

**WARSAW UNIVERSITY OF TECHNOLOGY**

DISCIPLINE OF SCIENCE INFORMATION  
AND COMMUNICATION TECHNOLOGY  
FIELD OF SCIENCE ENGINEERING AND TECHNOLOGY

# **Ph.D. Thesis**

Łukasz Lepak, MSc.

**Task automation using artificial intelligence methods**

**Supervisor**

Paweł Wawrzyński, PhD DSc

**WARSAW 2024**



## **Acknowledgements**

I would like to thank my PhD advisor, family, friends and coworkers, without whom the beautiful, exceptional journey presented in this dissertation would not be possible.



## **Task automation using artificial intelligence methods**

This dissertation presents a series of five publications focusing on task automation with artificial intelligence methods.

We present automation of four different tasks based on artificial intelligence methods. For day-ahead energy trading, we use reinforcement learning to create an automatic bidding strategy that outputs a collection of bids based on available information relevant to the trading process. This strategy allows the trading agent to maximize its profits. Also, it can be used in real-life scenarios. For keyword spotting in audio recordings, we test several similarity matching architectures on various datasets and show that creating a keyword detector for low-resource languages is possible. For on-line hyperparameter optimization in neural network learning methods, we propose a novel algorithm that optimizes hyperparameters on the fly during the training process, with experiments presenting its effectiveness against other popular learning algorithms. For simultaneous machine translation, we propose an architecture learned with reinforcement that automatically controls the translation delay while maintaining high translation quality. We compare it with state-of-the-art neural machine translation architectures, achieving similar results with limited context.

Based on the observations from the translation task, we also present a novel recurrent cell that allows for a deep transformation of its state and is safe from gradient propagation issues. Conducted experiments show that the proposed cell achieves comparable, often better results than popular recurrent cells while using fewer parameters.

Overall, we introduce four artificial intelligence-based automations, showcasing their generality and usability in various tasks and fields, and a novel recurrent unit that can be used as a part of such automations.

**Keywords:** reinforcement learning, neural networks, task automation, artificial intelligence applications

## **Automatyzacja zadań z wykorzystaniem metod sztucznej inteligencji**

Niniejsza rozprawa doktorska przedstawia serię pięciu publikacji dotyczących automatyzacji zadań z wykorzystaniem metod sztucznej inteligencji.

Przedstawiamy sposoby automatyzacji czterech różnych zadań oparte na metodach sztucznej inteligencji. W handlu energią elektryczną na rynku dnia następnego wykorzystujemy uczenie ze wzmocnieniem do stworzenia automatycznej strategii handlu, która generuje zbiór ofert w oparciu o dostępne informacje istotne z punktu widzenia handlu. Pokazujemy, że strategia ta pozwala agentowi maksymalizować zyski. Ponadto, może być ona stosowana w rzeczywistych zastosowaniach. W przypadku wykrywania słów kluczowych w nagraniach audio testujemy kilka architektur dopasowywania podobnych obiektów na różnych zbiorach danych i pokazujemy, że możliwe jest stworzenie detektora słów kluczowych dla języków z małą ilością dostępnych danych. W przypadku optymalizacji hiperparametrów on-line w metodach uczenia sieci neuronowych, proponujemy nowy algorytm, który optymalizuje wartości hiperparametrów podczas procesu uczenia się; eksperymenty prezentują jego skuteczność w porównaniu z innymi popularnymi algorytmami uczenia. W przypadku symultanicznego tłumaczenia maszynowego proponujemy architekturę uczoną ze wzmocnieniem, która automatycznie kontroluje opóźnienie tłumaczenia przy zachowaniu jego wysokiej jakości. Porównujemy ją z najnowocześniejszymi architekturami neuronowego tłumaczenia maszynowego, osiągając podobne wyniki przy ograniczonym kontekście.

Bazując na obserwacjach z zadania tłumaczenia maszynowego, prezentujemy także nową komórkę rekurencyjną, która pozwala na głęboką transformację stanu i jest odporna na problemy związane z propagacją gradientu. Przeprowadzone eksperymenty pokazują, że proponowana komórka osiąga porównywalne, a często lepsze wyniki niż popularne komórki rekurencyjne przy użyciu mniejszej liczby parametrów.

Podsumowując, przedstawiamy cztery automatyzacje oparte na sztucznej inteligencji, co potwierdza ich ogólność i użyteczność w różnych zadaniach i dziedzinach, a także nową jednostkę rekurencyjną, która może być używana jako część takich automatyzacji.

**Słowa kluczowe:** uczenie ze wzmocnieniem, sieci neuronowe, automatyzacja zadań, zastosowania sztucznej inteligencji

# Contents

- Acknowledgements . . . . . 3**
  
- 1. Introduction 9**
  - 1.1. Contributions . . . . . 10**
    - 1.1.1. Dissertation structure . . . . . 12
  - 1.2. List of publications . . . . . 13**
    - 1.2.1. Publications in the series . . . . . 13
    - 1.2.2. Publications not in the series . . . . . 14
  
- 2. Background 15**
  - 2.1. Reinforcement learning . . . . . 15**
  - 2.2. Neural architectures . . . . . 18**
    - 2.2.1. Feed-forward architectures . . . . . 18
    - 2.2.2. Recurrent architectures . . . . . 19
    - 2.2.3. Sequence-to-sequence architectures . . . . . 20
    - 2.2.4. Similarity ranking architectures . . . . . 21
  - 2.3. Neural network learning . . . . . 23**
    - 2.3.1. Classic learning methods . . . . . 24
    - 2.3.2. Adaptive gradient methods . . . . . 25
  
- 3. Day-ahead automated energy trading 29**
  - 3.1. Proposed solution . . . . . 30**
  
- 4. Polish keyword spotting in audio recordings 35**
  - 4.1. Research project results . . . . . 36**
  
- 5. On-line hyperparameter tuning in neural network learning algorithms 39**
  - 5.1. Proposed approach . . . . . 39**
  
- 6. Simultaneous machine translation 43**
  - 6.1. Proposed solution . . . . . 43**

<b>7. Deep state transformation in recurrent neural networks</b>	<b>47</b>
<b>8. Final remarks</b>	<b>49</b>
<b>9. Other achievements</b>	<b>51</b>
<b>Bibliography</b>	<b>53</b>
<b>Appendices</b>	<b>59</b>
<b>A. List of Abbreviations</b>	<b>60</b>
<b>B. List of Publications</b>	<b>61</b>
<b>B.1. Reinforcement learning meets microeconomics: Learning to designate price-dependent supply and demand for automated trading . . . . .</b>	<b>62</b>
<b>B.2. Generalisation gap of keyword spotters in a cross-speaker low-resource scenario . . . . .</b>	<b>97</b>
<b>B.3. Automatic hyperparameter tuning in on-line learning: Classic Momentum and ADAM . . . . .</b>	<b>124</b>
<b>B.4. Reinforcement Learning for on-line Sequence Transformation</b>	<b>133</b>
<b>B.5. Least Redundant Gated Recurrent Neural Network . . . . .</b>	<b>141</b>



# 1. Introduction

The popularity of artificial intelligence (AI) has grown significantly over the last few years. The number of solutions and applications based on AI methods is steadily increasing across various industries, with their market value and research and development spending larger than ever (Bharadiya et al., 2023). Many of these solutions focus on automating tasks and processes.

We interact with various automated processes daily. When driving a car or crossing the street, we go through traffic lights, which automatically control the traffic flow. In some buildings, like shopping malls, doors open automatically when someone approaches them. Elevators automatically maintain the order of floors they are supposed to stop at based on the requests they receive. Smartphones create personalized recommendations based on the user's preferences and observe their usage patterns, adapting to them by optimizing various operations like battery charging. While checking e-mail, spam messages are filtered not to clutter the inboxes.

Some tasks are always expected to do the same actions and produce the same result whenever their preconditions are met. For instance, when someone approaches automatic doors, they are expected to open. The main difficulty of automating such processes is preparing and connecting the components needed to make them operational.

On the other hand, some tasks require different actions based on circumstances, and different results may be produced. A good example is the previously mentioned e-mail spam filter - whether the email lands in the main inbox or the spam folder depends on the contents of the analyzed message. Here, the automation difficulty starts with creating its logic so that expected results are achieved. The rules for such automations may be based on some criteria, and the result of their operation depends on the fulfillment of these criteria. Not all automations, however, may be covered by a set of explicit rules, thus requiring different, approximate methods to achieve expected results.

Automations for many tasks can be created using artificial intelligence methods, ranging from classic machine learning methods like support vector machines (SVM) (Cortes and Vapnik, 1995), or gradient boosting (Friedman, 2001), to modern architectures like Transformers (Vaswani et al., 2017) or large language models

(LLMs) (Achiam et al., 2023; Anil et al., 2023; Touvron et al., 2023). The choice of an appropriate approach depends on the automated task. For example, the e-mail spam filter can be implemented by processing the message using natural language processing (NLP) techniques and classifying the result using a chosen classification approach, like neural networks. Nowadays, customer support services are often implemented using AI-based chatbots (Adam et al., 2021) for faster response times and quick resolution of simple cases. Artificial intelligence is also widely used in recommender systems (Zhang et al., 2021), which are used to create personalized content based on the users' preferences.

## 1.1. Contributions

This dissertation, based on a series of publications listed in Section 1.2.1, focuses on task automation using artificial intelligence methods. We automate the following tasks:

- day-ahead energy market trading,
- simultaneous machine translation (SMT),
- Polish keyword spotting in audio recordings,
- on-line hyperparameter tuning in neural network learning algorithms.

Based on our observations regarding the SMT task, we also introduce a novel recurrent neural module, which can be used to automate various sequential data tasks like sequence classification (i.e., deciding whether the review is positive or negative), machine translation, or language modeling.

**Energy trading.** For the day-ahead energy market trading task, we formulate the following hypothesis:

**Hypothesis 1.** *Utilizing information relevant to the day-ahead energy market trading process to automate bid creation increases the trading agent's profits and allows the strategy to be used in real-life scenarios.*

To verify Hypothesis 1, we designed a parametric trading strategy that outputs a collection of bids (multiple buy and sell bids for every hour, representing the supply and demand curves of the trading agent). We used reinforcement learning to train a policy, which translates relevant information - market, weather forecast, battery, and time data - into parameters of the aforementioned strategy. Our experiments show that the proposed strategy achieves the best profits compared to the trading strategy that outputs only one bid of each type and a simple parametric strategy that uses no external information and creates bids at statistically beneficial hours. The

proposed strategy also efficiently manages its battery storage, never being forced to make unwanted transactions, and can be used in real-life scenarios.

**Keyword spotting.** Keyword spotting in audio recordings involves finding fragments of provided audio data at which words from the provided list of words, known as keywords, are spoken. The hypothesis for this task is:

**Hypothesis 2.** *Finding keywords in call center recordings in a low-resource language is possible using audio similarity matching models.*

To verify Hypothesis 2, we presented the research project results, done for a Polish bank, to spot keywords in call center recordings. We prepared and evaluated different similarity ranking models on various datasets in English and Polish. Despite a significant performance gap between English and Polish detectors, we created a Polish keyword detector with good results on some words, which can be useful in time-consuming call center audio verification.

**On-line hyperparameter tuning.** The task of finding the best hyperparameter combination for the neural network learning algorithms is often time-consuming and requires significant computing power. Even if the default values are provided, they are not enough in most cases, and major performance improvements can be achieved by tuning them. Also, the optimal hyperparameter values can change as the training process progresses. Based on these observations, we set the hypothesis:

**Hypothesis 3.** *The short- and long-term influence of the learning algorithm’s hyperparameters on the training process can be used to optimize these hyperparameters on the fly without prior knowledge of the solved problem, thus improving the learning performance.*

For the validation of Hypothesis 3, we introduced Autonomous Stochastic Descent with Momentum, version 2 (ASDM2) method, which automatically tunes the hyperparameters of classic momentum (CM) (Polyak, 1964) and ADAM (Kingma and Ba, 2014) learning algorithms while they are running. Experiments on shallow classifiers and deep autoencoders show that the proposed method achieves the best results across popular state-of-the-art learning algorithms, beating base methods’ results with their hyperparameters optimized manually.

**Simultaneous machine translation.** The simultaneous machine translation task can be defined as transforming the input sequence into the output sequence, creating output tokens while input tokens are still being read. An important factor of such translation is the delay between reading the input and generating the translated output. Here, we set the hypothesis:

**Hypothesis 4.** *The translation delay in a simultaneous machine translation can be automatically controlled while maintaining high translation quality for sequences of arbitrary length.*

For Hypothesis 4, we introduced an architecture that learns with reinforcement and can control the translation delay automatically. We compared it with state-of-the-art neural machine translation (NMT) architectures on machine translation tasks. While the NMT architectures generate the output sequence after reading the whole input sequence, our solution produces output tokens during the processing of the input sequence, with the delay being controlled automatically. The results show that the proposed architecture achieves similar results to the NMT architectures, with stronger performance on longer sequences, preserving the context of long sequences.

**Recurrent neural module.** Recurrent neural networks (RNNs) are the main components of many systems processing sequential data. However, most of them do not allow deep state transformations due to their structure, which limits the capabilities of capturing state relations. We experienced such limitations while creating the SMT system from the previous paragraph. Based on this, we formulate the hypothesis:

**Hypothesis 5.** *Deep, arbitrary transformation of states in a recurrent neural network allows to achieve results comparable with state-of-the-art methods while mitigating training problems and being memory-efficient.*

To test Hypothesis 5, we introduced a recurrent neural architecture called Deep Memory Update (DMU), allowing any feed-forward neural network to model the state transformation. It also includes mechanisms to cope with gradient vanishing and exploding during training, a common issue for RNNs (Bengio et al., 1994). Conducted experiments show that the DMU module surpasses or at least matches the results of state-of-the-art RNNs on various synthetic and real data-based tasks.

### 1.1.1. Dissertation structure

The structure of this dissertation is as follows:

- Chapter 2 presents AI methods used in this dissertation - reinforcement learning, neural architectures, and neural network learning algorithms,
- Chapters 3-6 details tasks introduced in Section 1.1 and introduces our methods of automating them,
- Chapter 7 describes the novel recurrent neural unit, which addresses the problems encountered during automating sequence-based tasks,

- Chapter 8 briefly discusses the obtained results and concludes the dissertation.

## 1.2. List of publications

### 1.2.1. Publications in the series

This dissertation is based on five papers (four published, one accepted for publication):

- **[P1] Łukasz Lepak**, Paweł Wawrzyński. “Reinforcement learning meets microeconomics: Learning to designate price-dependent supply and demand for automated trading”, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2024 (*accepted for publication*)

**Contribution:**

The PhD candidate gathered the required data, designed, developed, tested, and analyzed the proposed trading strategy, and prepared the manuscript.

Ministerial score: **140**

Percentage of contribution: **75%**

- **[P2] Łukasz Lepak**, Kacper Radzikowski, Robert Nowak, Karol J. Piczak. “Generalisation gap of keyword spotters in a cross-speaker low-resource scenario”, Sensors, MDPI, 2021

**Contribution:**

The PhD candidate developed, tested and analyzed created solutions, gathered the required data, and prepared the manuscript.

Ministerial score: **100**

Impact factor: **3.4**

Percentage of contribution: **40%**

- **[P3]** Paweł Wawrzyński, Paweł Zawistowski, **Łukasz Lepak**. “Automatic hyperparameter tuning in on-line learning: Classic Momentum and ADAM”, 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020

**Contribution:**

The PhD candidate developed, tested, and analyzed different versions of the proposed algorithm.

Ministerial score: **140**

Percentage of contribution: **40%**

- **[P4]** Grzegorz Rypeść, **Łukasz Lepak**, Paweł Wawrzyński. “Reinforcement Learning for on-line Sequence Transformation”, 2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS), IEEE, 2022

**Contribution:**

The PhD candidate assisted in developing, testing, and analyzing the proposed system, reviewed the literature, prepared the manuscript, and presented the method at the conference.

Ministerial score: **70**

Percentage of contribution: **50%**

- **[P5]** Łukasz Neumann, **Łukasz Lepak**, Paweł Wawrzyński. “Least Redundant Gated Recurrent Neural Network”, 2023 International Joint Conference on Neural Networks (IJCNN), IEEE, 2023

**Contribution:**

The PhD candidate developed models to which the proposed solution was compared, and conducted part of the experiments.

Ministerial score: **140**

Percentage of contribution: **20%**

**1.2.2. Publications not in the series**

- **Łukasz Lepak**, Paweł Wawrzyński. “Sztuczna inteligencja handlująca energią na rynku dnia następnego”, Raport Otwarcia IV Kongresu Elektryki Polskiej, Stowarzyszenie Elektryków Polskich, 2024
- Bogdan Bednarski, **Łukasz Lepak**, Jakub Łyskawa, Paweł Pieńczuk, Maciej Rosoł, Ryszard Romaniuk. “Influence of IQT on research in ICT”, International Journal of Electronics and Telecommunications, 2022
- Kacper Radzikowski, Karol Chęciński, Mateusz Forc, **Łukasz Lepak**, Michał Jabłoński, Wiktor Kuśmirek, Bartłomiej Twardowski, Paweł Wawrzyński, Robert M. Nowak. “Widget detection on screenshots using computer vision and machine learning algorithms”, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, SPIE, 2019
- Paweł Wawrzyński, Paweł Zawistowski, **Łukasz Lepak**. “On the development of the ASDM method”, Polskie Porozumienie na rzecz Rozwoju Sztucznej Inteligencji, Politechnika Wrocławska, 2019

## 2. Background

### 2.1. Reinforcement learning

Reinforcement learning (Sutton and Barto, 2018) is one of the machine learning paradigms, alongside supervised and unsupervised learning. It describes an agent's learning to take optimal actions based on interactions with the dynamic environment over time. This framework is known as Markov Decision Processes (MDPs) (Howard, 1960). The properties of the MDP are:

- time,  $t = 1, 2, \dots$
- the state space,  $s_t \in S$ , defines the possible states of the environment,
- the action space,  $a_t \in A$ , defines the actions the agent can take,
- the state transition distribution,  $P_s(s_{t+1}|s_t, a_t)$ , defines the probability of the environment going into state  $s_{t+1}$  from state  $s_t$  after the agent takes the action  $a_t$ ,
- the reward function,  $r_t = R(s_t, s_{t+1}, a_t), r_t \in \mathbb{R}$ , quantifies the quality of the chosen action,
- the initial state distribution,  $P_0$ , defines the distribution of the environment's initial states,
- the terminal states set,  $S_T$ , contains the states after which the interactions cannot continue.

The interactions are divided into episodes, an episode being a sequence of states from the initial to the terminal state.

MDPs, in which an agent can access full environmental information, are called Fully Observable Markov Decision Processes (FOMDPs). MDPs, in which the information available to an agent is transformed by a certain function of an environment's state, are called Partially Observable Markov Decision Processes (POMDPs).

The agent chooses the action in the current state based on the distribution  $\pi(a|s)$  called the policy. The policy defines the probability of choosing an action  $a$  in a state  $s$ . Reinforcement learning methods aim to find an optimal policy based on environmental interactions. The optimal policy  $\pi^*$  is the one that maximizes the

expected discounted future rewards sum:

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left( \sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid \pi \right)$$

where  $\gamma \in [0, 1]$  is called the discount factor. The higher the discount factor, the longer the horizon of important future rewards.

The value function  $V^{\pi}$  of the policy  $\pi$  is defined as:

$$V^{\pi}(s) = \mathbb{E} \left( \sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s; \pi \right) = \mathbb{E} (r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t = s; \pi)$$

The value function is the expected discounted rewards sum starting from state  $s$  and taking actions according to the policy  $\pi$ . With the optimal policy, this function should have the maximum possible value for every possible state, as the optimal policy chooses optimal actions, resulting in the highest rewards (Bellman, 1952).

The action-value function  $Q^{\pi}$  of the policy  $\pi$  is defined as:

$$Q^{\pi}(s, a) = \mathbb{E} \left( \sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s, a_t = a; \pi \right) = \mathbb{E} (r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t = s, a_t = a; \pi)$$

The action-value function is the expected discounted rewards sum by taking action  $a$  at the state  $s$  and later taking actions according to the policy  $\pi$ . This allows for the action to be assessed in the context of the current state.

The policy of an agent can be represented in various ways. For example, in the simple Q-Learning algorithm (Watkins, 1989), the Q-function can be stored as a table for discrete and low-dimensional state and action spaces. The resulting policy, defined as  $\pi = \operatorname{argmax}_a Q(s_t, a)$ , is fully represented by such a table. However, the tabular approach becomes infeasible when the spaces are high-dimensional or continuous. In that case, approximators, like neural networks, can be used, and the policy becomes parametrized.

The major issue in reinforcement learning is the exploration versus exploitation trade-off. While learning, the agent needs to explore the outcomes of different actions, which would be impossible if it were to always choose the currently optimal action. To enable such exploration, the agent randomly gets an action different from the currently considered optimal one during the learning process. Many methods exist to control the exploration process, including the  $\epsilon$ -greedy exploration for discrete action spaces and different parametrized noise distributions for continuous action spaces.

Reinforcement learning algorithms can be divided into categories based on different criteria:



- interaction with the environment - during training (on-line) or based on registered interactions (off-line),
- number of agents to control in the environment - one (single-agent) or many (multi-agent),
- modeling of the environment behavior - no modeling of the environment behavior (model-free) or creating a model of the environment and using it as part of learning (model-based),
- base of learning - direct learning of the policy function (policy-based), learning the value function and deriving policy from it (value-based), or a combination of both (actor-critic),
- experience usage - using only interactions from the current policy (on-policy) or gathering and reusing interactions from different policies (off-policy).

Common RL algorithms include:

- Q-Learning [on-line, single-agent, model-free, value-based, off-policy] (Watkins, 1989),
- SARSA [on-line, single-agent, model-free, value-based, on-policy] (Rummery and Niranjan, 1994),
- REINFORCE [on-line, single-agent, model-free, policy-based, off-policy] (Williams, 1992),
- Deep Q-Network (DQN) [on-line, single-agent, model-free, value-based, off-policy] (Mnih et al., 2013),
- Advantage Actor-Critic (A2C) [on-line, single-agent, model-free, actor-critic, on-policy] (Mnih et al., 2016),
- Proximal Policy Optimization (PPO) [on-line, single-agent, model-free, actor-critic, on-policy] (Schulman et al., 2017),
- Deep Deterministic Policy Gradient (DDPG) [on-line, single-agent, model-free, actor-critic, off-policy] (Lillicrap et al., 2015),
- Twin Delayed Deep Deterministic Policy Gradient (TD3) [on-line, single-agent, model-free, actor-critic, off-policy] (Fujimoto et al., 2018),
- Soft Actor-Critic (SAC) [on-line, single-agent, model-free, actor-critic, off-policy] (Haarnoja et al., 2018),
- Value Decomposition Networks [on-line, multi-agent, model-free, value-based, off-policy] (Sunehag et al., 2017),
- QMIX [on-line, multi-agent, model-free, value-based, off-policy] (Rashid et al., 2020).

## 2.2. Neural architectures

### 2.2.1. Feed-forward architectures

**Dense neural networks.** One of the basic neural architectures are dense neural networks (also called fully-connected neural networks). They consist of dense layers, which perform the following operation:

$$y = \phi(Wx + b)$$

where  $x$  is an input vector,  $y$  is an output vector, the matrix  $W$  and the vector  $b$  are the layer's trainable parameters, and  $\phi$  is the activation function. The activation function introduces non-linearity into the otherwise linear operation, allowing for usage in non-linear approximation problems. Example activation functions include the logistic function, hyperbolic tangent, or rectified linear unit function (ReLU) defined as  $\phi(z) = \max\{0, z\}$ .

The two-layer perceptron is a dense neural network with one typically non-linear layer (called the hidden layer) and one output layer. Assuming the activation function is non-constant, bounded, and continuous, this architecture satisfies the universal approximation conditions (Hornik et al., 1989). This means that, for every function, it is possible to find a two-layer perceptron approximating this function with an arbitrary accuracy. However, a universal method for finding such a combination of neurons and their weights is unknown.

Two-layer perceptrons are a specific example of multi-layer perceptrons, consisting of several hidden layers and one output layer. As long as at least one hidden layer is non-linear, they satisfy the universal approximation conditions.

Multi-layer perceptrons can be used for many approximation tasks, most notably supervised learning tasks like classification or regression.

**Autoencoders.** Autoencoders are architectures used to learn encodings from the input data without labels, which fits the unsupervised learning paradigm. In general, autoencoders learn to reconstruct the provided input data. They consist of two main parts. The encoder is composed of layers with a decreasing number of neurons. It transforms the input vector into the vector called the encoding. The encoding is usually a lot smaller than the original input. The decoder reconstructs the input vector from the encoding. Its layer structure is usually reversed from the encoder structure, with further layers usually having an increasing number of neurons.

There are several different autoencoder architectures. Dense autoencoders consist of dense (non-linear) layers. Convolutional autoencoders include dense, convo-

lution, and pooling layers in the encoder, and dense, transposed convolution, and upsampling layers in the decoder. They are useful when the input data consists of multi-dimensional structured data like color images. Variational autoencoders (VAEs) (Kingma and Welling, 2013) encode input data as a distribution over the latent space. This allows for better generative properties.

### 2.2.2. Recurrent architectures

Recurrent neural networks process the data one step at a time, passing the state of transformations between each input sequence element. This makes it a useful tool for processing variable-length sequences.

**Base recurrent architecture.** The base recurrent architecture (Jordan, 1986; Elman, 1990) follows the equation:

$$h_t = \phi(W_x x_t + W_h h_{t-1} + b_h)$$

where  $x_t$  is the input vector at time  $t$ ,  $h_t$  is called the hidden state, and it is also the output vector at time  $t$ , matrices  $W_x, W_h$  and vector  $b_h$  are the trainable parameters, and  $\phi$  is the activation function. Leveraging the hidden state allows this architecture to gather information from previously processed inputs.

The main issue with learning base RNNs is the problem of gradient vanishing and gradient exploding (Bengio et al., 1994). This results from the method used to calculate gradients in such networks, Backpropagation Through Time (BPTT) (Robinson and Fallside, 1987). This approach represents the recurrent structure as a graph of networks connected through hidden states, each having its input and the weights shared between them. For this graph, the gradient for the weights can be calculated, accumulated, and applied using the chosen learning method. However, due to this representation, even a slight change in network parameters may cause the gradients to vanish or explode, making the learning process fail.

**Long Short-Term Memory.** Hochreiter and Schmidhuber (1997) introduced the Long Short-Term Memory (LSTM) network to address the gradient vanishing and exploding problem. Compared to base RNNs, it adds another state vector, called the cell state, and operates based on three gates:

- forget gate - decides what information from the cell state should be kept,
- input gate - decides what information from the current input should be included in the cell state,

- output gate - decides what information should be output from the cell state.

The LSTM cell follows the equations:

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\hat{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \hat{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned}$$

where  $c_t$  is the cell state,  $W_*, U_*, b_*$  are the trainable parameters,  $f_t$  is the forget gate activation vector,  $i_t$  is the input gate activation vector,  $o_t$  is the output gate activation vector,  $\hat{c}_t$  is the cell input activation vector,  $\sigma$  is the logistic activation function,  $\tanh$  is the hyperbolic tangent activation function, and  $\odot$  denotes the element-wise product. Due to its structure, LSTM significantly reduces gradient vanishing and exploding. However, it is more memory- and time-consuming due to more parameters and complex calculations.

**Gated Recurrent Unit.** Cho et al. (2014) proposed another approach with the Gated Recurrent Unit (GRU) network. Instead of three gates, this architecture uses the update gate (matching the input gate from LSTM) and the reset gate (matching the forget gate from LSTM). This simplified structure reduces the memory consumption of the network while still achieving results comparable to those of LSTM.

The GRU cell follows the equations:

$$\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}$$

where  $z_t$  is the update gate activation vector,  $r_t$  is the reset gate activation vector, and  $\hat{h}_t$  is the candidate for the new hidden state.

### 2.2.3. Sequence-to-sequence architectures

Sequence-to-sequence (Seq2Seq) architectures are used to create output sequences based on input sequences. These can be, for example, translating the

text from one language to another or converting words from audio recordings into text.

**Encoder-Decoder.** The basic Encoder-Decoder architecture (Sutskever et al., 2014; Cho et al., 2014) consists of two titular parts. The encoder is a neural network that transforms tokens from the input sequence into a fixed-sized vector. The decoder is a neural network that takes this fixed-sized vector and produces tokens that create the output sequence. The generation of output tokens ends when the network produces a special token, often called an End-of-Sequence (EOS) token. Both Sutskever et al. (2014) and Cho et al. (2014) used LSTMs as encoder and decoder networks.

The basic Encoder-Decoder architectures suffer from the bottleneck problem, as their encoders produce fixed-length vectors, which make storing all relevant information for decoding output sequences increasingly difficult with rising input sequence length. This was alleviated by introducing the attention mechanism (Bahdanau et al., 2014), allowing the model to focus on more relevant input tokens when generating the output sequence. This is achieved by calculating the weight coefficients of all the input hidden states, considering the previous output token or decoder state with the separate alignment model (learned together with encoder and decoder), and using those weights during decoding to highlight more important parts of the input sequence.

**Transformer.** As recurrent models often take a long time to train, another approach was proposed by Vaswani et al. (2017), known as the Transformer architecture. It is not a recurrent architecture, relying on attention and non-recurrent operations. It is currently the state-of-the-art architecture with the most widespread use in natural language processing tasks. It is also the base architecture for various LLM models, including the GPT architectures (Achiam et al., 2023). Beyond NLP, Transformers achieve great results in many other fields, including computer vision (Ramachandran et al., 2019) and reinforcement learning (Chen et al., 2021).

#### 2.2.4. Similarity ranking architectures

Metric learning involves optimizing the fixed-length vector representations of objects, called embeddings, based on a distance metric like Euclidean or Manhattan distances. One of the applications of metric learning is similarity learning, which aims to minimize the distance between similar objects (i.e., belonging to the same category) and maximize the distance between non-similar objects (i.e., from different categories).

**Siamese networks.** Siamese neural networks (Bromley et al., 1993) are one of the most common similarity ranking architectures. They consist of an embedder, a neural network that produces embeddings for objects. However, at least two embeddings must be prepared for comparison, hence the “Siamese” part. What sets apart the Siamese neural networks from other architectures is the loss function.

Contrastive loss minimizes the distance if the objects are similar and maximizes the distance up to a defined margin if the objects are non-similar. It is defined as follows:

$$l_{i,j}(\theta) = s_{i,j} \cdot d(n(x_i; \theta), n(x_j; \theta)) + (1 - s_{i,j}) \cdot \max\{0, m - d(n(x_i; \theta), n(x_j; \theta))\}$$

where  $x_i, x_j$  are input objects,  $n$  is the embedder parametrized with weights  $\theta$ ,  $d$  is the chosen distance metric,  $s_{i,j} = 1$  for similar objects and 0 for non-similar objects,  $m$  is a margin defining the minimum desired distance between non-similar objects, and  $l_{i,j}(\theta)$  is the loss function.

Triplet loss utilizes three different objects and is calculated as follows:

$$l_{i,j,k}(\theta) = \max\{0, d(n(x_i; \theta), n(x_j; \theta)) - d(n(x_i; \theta), n(x_k; \theta)) + m\}$$

where  $x_i$  is called anchor input,  $x_j$  is called positive input,  $x_k$  is called negative input, and  $m$  is a margin defining the minimum desired distance between a positive and a negative pair. Through the triplet loss, the distance between the anchor and positive inputs is minimized, and at the same time, the distance between the anchor and negative inputs is maximized.

**Prototypical networks.** Prototypical networks (Snell et al., 2017) propose a different approach to similarity learning by introducing class prototypes. The training data is split randomly into two subsets for each training episode - support and query. For every class in the support subset, a class prototype is calculated as a mean vector of several support embeddings from this class:

$$p^c = \frac{1}{k} \sum_{i=1}^k n(s_i^c; \theta)$$

where  $c$  is one of the classes,  $p^c$  is the class prototype,  $k$  is the number of objects used and  $s_i^c$  are the support objects from class  $c$ , and  $n$  is the embedder parametrized with weights  $\theta$ .

The class prototypes are later used as a comparison point for all query embeddings. The loss function for one episode of training is defined as:

$$L(\theta) = \frac{1}{C|Q|} \sum_{c=1}^C \sum_{i=1}^{|Q|} r(q_i, c) \cdot d(n(q_i; \theta), p^c) + (1 - r(q_i, c)) \cdot \max\{0, m - d(n(q_i; \theta), p^c)\}$$

where  $C$  is the number of support classes,  $Q$  is the query subset,  $q_i$  is the  $i$ -th query object,  $r(q_i, c) = 1$  if the  $i$ -th query object is of class  $c$ , 0 otherwise,  $d$  is the chosen distance metric, and  $m$  is the margin defining the minimum desired distance between the query objects and class prototypes from different classes. One epoch of prototype network training consists of several episodes with different support and query subsets.

## 2.3. Neural network learning

Neural network training aims to minimize the quality index:

$$\hat{q}(\theta) = \frac{1}{n} \sum_{i=1}^n q_i(\theta)$$

where  $\hat{q}$  is the global loss function (i.e. mean square error, cross-entropy, contrastive loss),  $\theta$  are the network weights,  $q_i$  is the loss function value for the  $i$ -th data sample, and  $n$  is the number of samples.

Commonly used learning algorithms utilize the gradient of the loss function with respect to the network parameters to update the weights. They are known as first-order methods. With a small enough dataset and network, calculating  $\nabla_{\theta} \hat{q}(\theta)$  is possible in a reasonable time, and a simple gradient descent method can be used. However, gradient calculation over the whole dataset in one step is infeasible in most cases.

**On-line learning.** In on-line learning, the gradient is calculated based on a small subset of a dataset called a mini-batch. It is assumed that:

$$\begin{aligned} \hat{q}(\theta) &= \mathbb{E} q_{\xi}(\theta) \\ \nabla_{\theta} \hat{q}(\theta) &= \mathbb{E} \nabla_{\theta} q_{\xi}(\theta) \end{aligned}$$

where  $\xi$  is the mini-batch, and  $q_{\xi}$  is the loss function calculated on that mini-batch. This makes the gradient calculated on a mini-batch an unbiased estimator of the global gradient, which can be used to optimize the weights of the learned network.

Mini-batches should be created to make them cover most of the data samples. A

single pass through all mini-batches is called an epoch. On-line learning is usually finished when the sufficient value of the loss function is reached (usually on a separate validation dataset), there is no significant improvement in loss value (also usually on the validation dataset), or the amount of training epochs has reached the defined limit.

### 2.3.1. Classic learning methods

**Stochastic gradient descent.** The simplest on-line learning algorithm is the stochastic gradient descent (Rosenblatt, 1958). It is defined as follows:

$$\theta_{t+1} = \theta_t - \beta \nabla_{\theta_t} q_{\xi_t}(\theta_t)$$

where  $\beta$  is the learning rate (also called the step size). The learning rate defines how much of the gradient should be used to update the weights. Low learning rates result in slow learning, while high learning rates can result in instability of the learning process and its failure. SGD convergence criteria exist for the learning rate (Robbins and Siegmund, 1971). However, using them yields low learning rates, so in most cases, it has to be optimized by the user, as no default value exists. The SGD method is prone to gradient fluctuation, as it stores no information on previous gradients. That is, with subsequent mini-batch gradients pointing in different directions, the network weights may jump from one part of the parameter space to another, causing spikes in the loss function values and potential instability.

**Momentum methods.** SGD algorithm can be easily improved by introducing momentum, which is responsible for gathering gradient changes:

$$\begin{aligned} m_t &= \lambda m_{t-1} - \beta g_t \\ \theta_{t+1} &= \theta_t + m_t \end{aligned} \tag{2.1}$$

where  $g_t$  is the gradient,  $m_t$  is the momentum component, and  $\lambda$  is the momentum decay factor, which dictates the strength of diminishing previous gradient changes influence on the current weight change. Momentum methods significantly reduce the gradient fluctuation issue from the SGD method, maintaining the trajectory of parameter changes through the momentum component.

Two main momentum methods differ in gradient calculation. The classic momentum algorithm (Polyak, 1964) calculates the gradient with respect to the current parameters:

$$g_t = \nabla_{\theta_t} q_{\xi_t}(\theta_t)$$



The second method, Nesterov accelerated gradient (NAG) (Nesterov, 1983), calculates the gradient after shifting the parameters with the momentum component:

$$g_t = \nabla_{\theta_t + \lambda m_{t-1}} q_{\xi_t}(\theta_t + \lambda m_{t-1})$$

This can be useful in situations when the learning process approaches instability. In that case, the NAG algorithm can react to it faster, as the gradient calculated with respect to shifted parameters will take the original parameters away from the instability.

### 2.3.2. Adaptive gradient methods

Current state-of-the-art gradient learning methods utilize various gradient adaptation techniques to improve the learning process in terms of performance and stability.

For notation clarity, algebraic operations on vectors in this subsection work in an element-wise way. For example,  $x^2$  produces a vector with squared elements of vector  $x$ .

**AdaGrad.** AdaGrad (Duchi et al., 2011) collects the sum of all squared gradients and uses this to adjust the set learning rate, making it different for every network parameter:

$$G_t = G_{t-1} + (\nabla_{\theta_t} q_{\xi_t}(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\beta}{\sqrt{G_t + \varepsilon}} \nabla_{\theta_t} q_{\xi_t}(\theta_t)$$

where  $\beta$  is the global learning rate,  $G_t$  is the all square gradients sum, and  $\varepsilon$  is a small constant preventing division by zero. With non-growing individual learning rates, AdaGrad usually ensures a stable training process. However, this sometimes results in bad training results due to a fast learning rate decrease, thus limiting network parameter changes.

**RMSprop.** To prevent an aggressive learning rate decrease, the RMSprop algorithm (Tieleman, 2012) replaced the sum of the squared gradients with exponential smoothing:

$$\bar{g}_t = \gamma \bar{g}_{t-1} + (1 - \gamma) (\nabla_{\theta_t} q_{\xi_t}(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\beta}{\sqrt{\bar{g}_t + \varepsilon}} \nabla_{\theta_t} q_{\xi_t}(\theta_t)$$

where  $\bar{g}_t$  is the exponential moving average of squared gradients, and  $\gamma$  is the smoothing factor. This allows the individual learning rates to change in any direction based on the received gradients, lowering the learning rate for higher gradients and increasing it for lower ones, making the training process stable and efficient.

**Adadelta.** Around the same time as RMSprop, Adadelta was introduced (Zeiler, 2012), also aiming at limiting Adagrad’s aggressive learning rate decrease mechanism:

$$\begin{aligned}\bar{g}_t &= \gamma\bar{g}_{t-1} + (1-\gamma)(\nabla_{\theta_t} q_{\xi_t}(\theta_t))^2 \\ \Delta\theta_t &= -\frac{\sqrt{\bar{\theta}_{t-1} + \varepsilon}}{\sqrt{\bar{g}_t + \varepsilon}} \nabla_{\theta_t} q_{\xi_t}(\theta_t) \\ \bar{\theta}_t &= \gamma\bar{\theta}_{t-1} + (1-\gamma)\Delta\theta_t^2 \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$

where  $\Delta\theta_t$  is the current parameter change, and  $\bar{\theta}_t$  is the exponential moving average of squared parameter changes. Adadelta used exponential smoothing similar to the RMSprop algorithm while also including a way of approximating the base learning rate.

**Adam.** Another adaptive gradient method is Adam, introduced by Kingma and Ba (2014). With all previously mentioned adaptive methods based on SGD, Adam is based on the CM algorithm:

$$\begin{aligned}m_t &= \gamma_1 m_{t-1} + (1-\gamma_1)\nabla_{\theta_t} q_{\xi_t}(\theta_t) \\ v_t &= \gamma_2 v_{t-1} + (1-\gamma_2)(\nabla_{\theta_t} q_{\xi_t}(\theta_t))^2 \\ \hat{m}_t &= \frac{m_t}{1-\gamma_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\gamma_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\beta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t\end{aligned}\tag{2.2}$$

where  $m_t, v_t$  are exponentially smoothed gradients and their squares,  $\hat{m}_t, \hat{v}_t$  are their corrections, and  $\gamma_1, \gamma_2$  are smoothing factors for  $m_t, v_t$ , respectively.  $m_t$  represents an estimate of gradient averages, while  $v_t$  estimates gradient uncentered variance. Their corrections are mostly needed in early training parts, as both are initialized with zeros. The influence of these corrections diminishes over time. As with previous adaptive methods, Adam utilizes individual learning rates, but its updates are based on momentum rather than single gradients.

Adam is one of the most commonly used neural network learning algorithms, considered state-of-the-art, thanks to its efficient operation compared to other algorithms, even on default hyperparameter values, and fast convergence.

Many modifications of the Adam algorithm are available. The authors of Adam also introduced AdaMax (Kingma and Ba, 2014), which replaces  $v$  calculation with the infinity norm and skips correction steps. Dozat (2016) introduced Nadam, which is based on the NAG method, adding similar modifications. Reddi et al. (2019) showed that methods utilizing squared roots of the exponential moving average of squared gradients (RMSprop, Adadelta, Adam, Nadam) do not converge on certain tasks, and proposed an AMSGRAD method, which uses  $\hat{m}_t = m_t$  and  $\hat{v}_t = \max\{\hat{v}_{t-1}, v_t\}$ .



### 3. Day-ahead automated energy trading

Day-ahead energy markets allow their participants to buy and sell energy the day before the actual energy exchange happens. Their main goal is to balance energy demand and generation, especially with the volatile renewable energy generation share rapidly growing in recent years (Çam, 2024). Participants of such markets can place buy and sell bids for every hour of the next day. In its Polish instance, these bids must be placed before 10.30 am the day preceding the energy exchange. Also, the Polish day-ahead energy market sets the minimum volume for bids at 0.1 MWh.

The single bid is defined as:

$$\langle type, hour, price, volume \rangle$$

where  $type \in \{buy, sell\}$  is the type of the bid,  $hour \in \{0, \dots, 23\}$  is the hour for which the bid is placed,  $price$  denotes the highest (for buy bid) or lowest (for sell bid) price for the bid to be realized and  $volume$  sets the amount of energy to be transferred when the bid is executed. The bids are placed without knowing the price of energy at each hour. After receiving all the bids, the market aggregator sorts them based on price, determines the hourly supply and demand equilibrium, and sets the market clearing price for each hour. Buy bids with prices not lower than the market price and sell bids with prices not higher than the market price are accepted and executed the next day at market prices.

We assume the trading agent to be a medium-sized entity (like a group of households or a small power plant) with battery energy storage, optional energy consumption, and optional renewable energy generation capabilities. The goal of this agent is to generate buy and sell bids for the day-ahead energy market that maximize its profits (or minimize its costs) by participation in day-ahead trading.

By the bidding strategy, we understand a method of creating bids for the market. An example of a simple parametric strategy is “buy  $x$  MWh of energy at 2 am, sell  $y$  MWh of energy at 4 pm”, as the energy is generally cheap at night and expensive in the afternoon. The parameters in such strategies can be optimized with various optimization techniques, including linear programming (Bakirtzis et al., 2007), genetic (Wen and David, 2001; Changsong et al., 2009) and evolutionary algorithms (Attaviriyanupap et al., 2005), or stochastic optimization (Liu et al., 2015). While

useful for simple strategies, these optimization techniques utilize little to no external information and cannot transform it into more meaningful bids. Instead, they rely on simple parameter optimization to maximize their efficiency on historical data.

For information transformation into bids, we can use reinforcement learning. There are various applications of reinforcement learning in energy systems (Jogunola et al., 2020; Yang et al., 2020; Perera and Kamalaruban, 2021), including on-line energy trading on local energy markets (Jogunola et al., 2021; Okwuibe et al., 2022; May and Huang, 2023) and bidding in day-ahead energy markets (Dong et al., 2021).

Dong et al. (2021) models trading on both day-ahead and hour-ahead energy markets as a Markov Decision Process, using RL to optimize the bidding strategy for profit maximization while managing a system of battery energy storages. They use a modified Q-Learning approach with Q-function approximation. However, they use very little external information for their strategy, which limits its ability to place meaningful bids. They treat one day as an episode, with hours as steps, so the between-day dynamics of the trading process are not accounted for. They place only a single bid each hour, with discrete actions, limiting possible volumes and prices. Because of this, such an approach could not be used in real-life scenarios, with uncertainty regarding energy prices and volatile energy generation.

### 3.1. Proposed solution

In publication P1, we proposed a bidding strategy that places a collection of buy and sell bids for every hour. These collections represent the trading agent's demand and supply curves, allowing it to exchange more energy when the price is beneficial. This would not be possible with a single buy and sell bid, as in this case, the bids get fully executed or not executed at all.

We define collections of buy and sell bids for every hour. The collection of sell bids for the hour  $h = 0, \dots, 23$  is defined as

$$\begin{aligned} n_s^h &= \lfloor c_q \exp(c_e a_h) / q_0 + 1/2 \rfloor \\ p_s^{h,i} &= c_p^h \exp(a_{h+24}) \left( 1 + \exp(a_{96}) \left( -(2a_{98} + 4)^{-1} + (i/n_s^h)^{2a_{98}+3} \right) \right) \end{aligned} \quad (3.1)$$

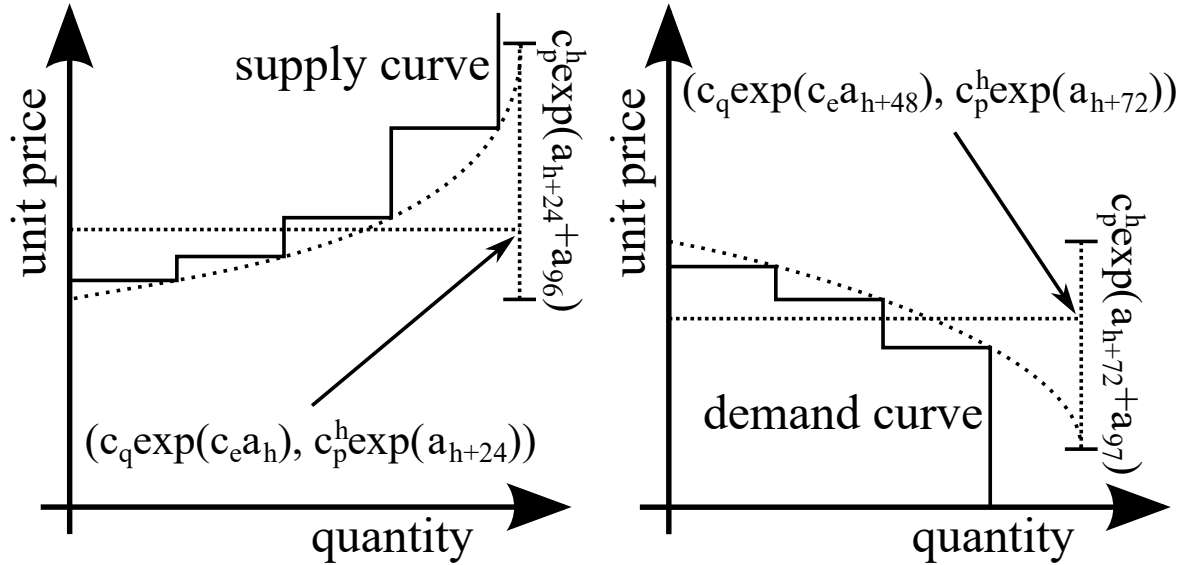
The collection of buy bids for the hour  $h = 0, \dots, 23$  is defined as

$$\begin{aligned} n_d^h &= \lfloor c_q \exp(c_e a_{h+48}) / q_0 + 1/2 \rfloor \\ p_d^{h,i} &= c_p^h \exp(a_{h+72}) \left( 1 + \exp(a_{97}) \left( (2a_{99} + 4)^{-1} - (i/n_d^h)^{2a_{99}+3} \right) \right) \end{aligned} \quad (3.2)$$

Here,  $\alpha_k$  denotes  $k$ -th coordinate of the action  $\alpha$ , and

- $a_h/a_{h+48}$  defines the width of the supply/demand curve, i.e., the number of sell/buy bids for the hour  $h$ ,
- $a_{h+24}/a_{h+72}$  defines the average height at which the supply/demand curve is located,
- $a_{h+24} + a_{96} / a_{h+72} + a_{97}$  defines vertical span of the supply/demand curve,
- $a_{98}/a_{99}$  defines convexity/concavity of the supply/demand curve,
- $c_q$  is the quantity scaling factor (we assume its value to be equal to the maximum hourly production of the installed sources),
- $c_p^h$  is the price scaling factor (we assume its value to be equal to the median price for hour  $h$  over the last 28 trading days),
- $c_e$  is the quantity exponent scaling factor (we assume  $c_e = 3$ ).

The representation of supply and demand curves defined by Equations 3.1 and 3.2 is presented in Figure 1.



**Figure 1.** Supply and demand defined by the proposed collections of bids approach.

To optimize the collection of bids strategy with RL, we parametrize it with parameters set by the trading agent’s policy. The policy is in the form of a feed-forward neural network. The policy translates the available information (observations) into strategy parameters (actions). We provide the trading agent with relevant energy trading process information:

- prices of energy at the current day for every hour (24 values) – these are the prices for the current day, for which the bids were created the day before; the agent does not know energy prices for the bids currently submitted. This gives an agent information about the price level at all hours.

- one-hot encoded information about the current month (12 values) and the current day of the week (7 values), giving the agent information about the current season and the week’s progress.
- cloudiness, wind speed, and temperature forecasts for each hour of the next day (72 values). They allow the trading agent to estimate its production capabilities, which can be used to place better bids.
- current relative battery charge (1 value), informing the agent about his current battery state of charge.
- estimated relative battery charge at midnight (1 value), an estimation of the battery state of charge at midnight. This estimation is quite accurate, as we know which bids are accepted for the current day (they were placed on the previous day), and with weather forecasts, we can estimate the renewable energy generation.

The reward is defined as  $r_t = 10^{-3}(p_t - \bar{p}_t - \rho_t)$ , where  $t$  represent a single day of trading operations,  $p_t$  is the achieved daily balance,  $\bar{p}_t$  is the reference balance, and  $\rho_t$  is the regularization penalty. The reference balance  $\bar{p}_t$  is a daily balance that would be achieved if the difference between daily produced and consumed energy was sold or bought at the average market price from that day. The regularization penalty  $\bar{p}_t$  prevents the strategy parameters from saturating at their bounds.

We can use any on-line RL algorithms that support continuous observations and actions to optimize the trading agent’s policy. Through preliminary experiments, we chose A2C as the best-performing and most stable algorithm.

The assumption of the trading agent’s size means it does not influence market prices through his operations, allowing us to utilize historical market data to represent the trading process in the simulation accurately.

We compare our proposed collection of bids strategy optimized with RL with its simpler form, also optimized with RL, where only a pair of one buy and sell bid is placed every hour, and with a simple parametric strategy buying and selling energy at statistically beneficial hours, optimized with the CMA-ES algorithm (Hansen, 2016). The experiments are run on a simulator of Polish day-ahead energy market operations. We test these strategies on four scenarios:

- an agent with battery energy storage only,
- an agent with battery energy storage and production capabilities,
- an agent with battery energy storage and consumption capabilities,
- an agent with battery energy storage, production, and consumption capabilities.

The results show that our proposed collection of bids strategy achieves the best results across all tested strategies in all scenarios. We also present the reasonable



behavior of the trained agent, with efficient battery storage control and no need to make sudden purchases due to bad control. Both the collection and pair of bids strategies achieve significantly better results than the simple parametric strategy, which uses no external information. This shows that a bidding strategy powered by information relevant to the trading process allows it to create better bids, resulting in higher returns or fewer losses.

Our proposed strategy can be deployed in real-life scenarios due to its generality and adaptability to the data. We are currently in advanced talks with business and industry partners about implementing our strategy in real operations. Also, the solution got the attention of the Polish electricians' community, being featured during the annual Polish Electrics Congress (Cieślik, 2024).



## 4. Polish keyword spotting in audio recordings

Keyword spotting in audio recordings involves finding the occurrences of certain phrases in audio recordings. This can be useful for faster processing of recordings in call centers, where the recorded calls are often checked later by independent auditors.

There are two main approaches to keyword spotting - speech-to-text conversion and similarity matching. The first one converts the audio recording into text, on which the keywords are found. While text availability makes keyword spotting much easier, it requires a significant amount of data (Amodei et al., 2016), which is not always possible. Also, any conversion errors done on a speech-to-text part cannot be fixed during keyword spotting. With Polish being a low-resource language, having significantly fewer data sources available than English, and the research project oriented towards call center recordings, which are often of low quality and hard to transcribe, we focus on the second approach.

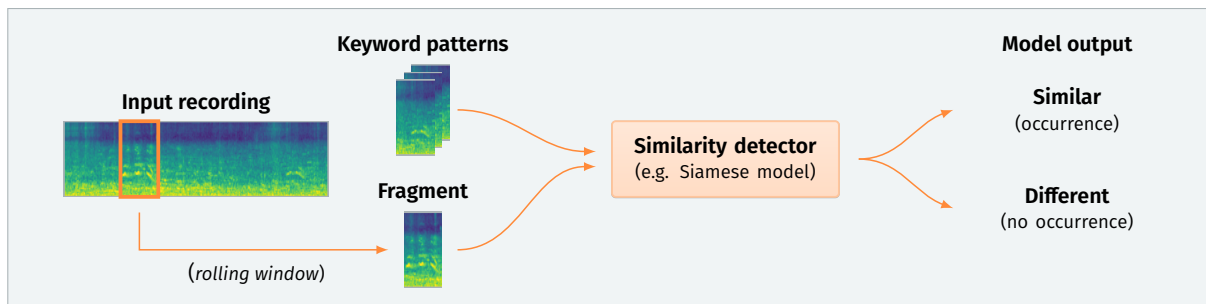
In audio similarity matching, the audio recording is split into small, possibly overlapping fragments with a rolling window of set size. These fragments are later compared with representative examples of the keywords, and if the fragment is considered similar by a model, a timestamp of its beginning is returned. For audio similarity models, the data amount requirements are considerably lower, and we can use few-shot similarity models like Siamese (Bromley et al., 1993) and prototypical (Snell et al., 2017) networks.

Dataset availability for audio data varies between languages. For English, which is a high-resource language, there is an abundance of datasets, including word utterances (Warden, 2018), read sentences (Panayotov et al., 2015; Ardila et al., 2019), or whole Wikipedia articles (Baumann et al., 2019). For low-resource languages like Polish, data availability is very limited. For example, as of now, the Common Voice dataset (Ardila et al., 2019) contains around 16 times fewer hours recorded for the Polish language than English. In 2020, when the research project was conducted, this difference was even more significant. While the availability of Polish audio data sources increases over time, there is still not enough labeled data, especially of word utterances, to enable the creation of the whole keyword spotting system.

## 4.1. Research project results

In publication P2, we described the keyword spotting research project results. We utilized various English and Polish datasets. We created most of the Polish audio data, with both actual and text-to-speech synthetic recordings. Access to the final keyword spotting evaluation dataset was strictly limited due to the bank’s security policies, so we tested our solutions on the gathered datasets and tested only the best promising ones on the bank dataset.

The general keyword spotting pipeline based on audio similarity matching is presented in Figure 2. We used Siamese (Bromley et al., 1993) and prototypical (Snell et al., 2017) networks on monolingual and cross-language scenarios with different training, pattern, and evaluation datasets. We also used the pre-trained speech embedding model from Google (Lin et al., 2020), with and without fine-tuning with our datasets and with or without results post-processing. The post-processing procedure involved standardizing similarity values between keywords and filtering out excessively spotted keywords.



**Figure 2.** The general pipeline of keyword spotting based on audio similarity matching.

The results show that all the tested models can achieve good results in the monolingual English case, with performance differing mostly based on the training dataset. In general, smaller training datasets that were more related to the keywords searched improved the results. The results of the Polish monolingual training were less than satisfactory, with models failing to detect most of the keywords’ utterances. In cross-language scenarios, with English training data and Polish pattern and evaluation data, results improved, although they were still unsatisfactory. The best results for the Polish datasets were achieved by the pre-trained speech embedding model, without fine-tuning and with results post-processing, with 0.69 f-score.

The best-performing model on Polish evaluation was chosen to be tested on a bank evaluation dataset. Here, the performance was disappointing but expected, with very little precision and less than desired recall, both with and without result post-processing. However, on keywords with more syllables, like “transakcja” or

“reklamacja”, the model achieved performance of around 75% precision, 10 – 20% recall. While not enough to fully automate the detection process, creating an audio similarity matching detector for low-resource language is possible, with it being a helpful tool for an auditor.



## 5. On-line hyperparameter tuning in neural network learning algorithms

In neural network learning, hyperparameter tuning is often one of the steps in improving the network’s performance on a given task. The number of hyperparameters to be optimized varies between the learning algorithms. While some algorithms provide default values for these hyperparameters, they do not always guarantee good enough results and must be tuned. In most cases, the most important parameter to optimize is the learning rate, but others may also significantly influence the resulting performance of the network.

Most algorithms can be used with hyperparameter schedulers, allowing their values to change based on a given formula or a set of rules. However, these schedulers do not account for inside training information, mostly relying on simple measures and rules. Common learning rate schedules decrease their value over time, depending on epochs passed or mini-batches processed.

Adaptive gradient methods, presented in Section 2.3.2, can be considered algorithms tuning their base methods’ hyperparameters on a per-weight basis. However, they require defining base learning rate and smoothing parameters, with their performance depending on the values of these hyperparameters.

Hyperparameter values can be optimized on preliminary runs using various approaches, including random search, grid search, and evolutionary algorithms. However, optimal hyperparameter values can also change during training, requiring an on-line optimization approach.

### 5.1. Proposed approach

In publication P3, we introduced Autonomous Stochastic Descent with Momentum, version 2 (ASDM2), a neural network learning algorithm based on the CM or Adam algorithm, depending on a chosen variant. The proposed method optimizes the base algorithm’s hyperparameters ( $\beta$  and  $\lambda$  in Equation 2.1,  $\beta$  and  $\gamma_1$  in Equation 2.2) on the fly during the training process. This algorithm refines the method presented by Wawrzyński (2017). It utilizes various estimators to capture these hyperparameters’ short-term and long-term influence on the learning process

by measuring their impact on the weights and the exponential moving average of weights. The exponential smoothing of weights represents their trend, with the smoothing coefficient also optimized on the fly. Hence, this long-term influence can be measured. Based on these estimators, a quality measure is defined to optimize the hyperparameters. Training stability and initialization formulas are also provided for the optimized hyperparameters. The ASDM2 algorithm requires several coefficients to operate. However, their values do not have to be optimized.

To optimize hyperparameters from Equations 2.1 and 2.2 while maintaining the desired short-term and long-term trend of network trainable parameters, the following quality index is proposed

$$Q_t = q_{\xi_t}(\bar{\theta}_t) + r_t^T(\theta_t - \theta_{t-1}) \quad (5.1)$$

where  $q$  is the loss function,  $\xi_t$  is the mini-batch,  $\theta_t$  are network trainable parameters,  $\bar{\theta}_t = \mu\bar{\theta}_{t-1} + (1-\mu)\theta_{t-1}$  are the network smoothed parameters, with  $\mu \in [0, 1)$  optimized to minimize  $q_{\xi_t}(\bar{\theta}_t)$ , and  $r_t = \nabla_{\bar{\theta}_t} q_{\xi_t}(\bar{\theta}_t)$  is treated as a constant.

The first term in Equation 5.1 is responsible for loss minimization for the smoothed parameters. The second term prevents the current network weights from fluctuating too much, preventing training instability. The ASDM2 algorithm minimizes the quality index from Equation 5.1 with respect to the optimized hyperparameters from Equations 2.1 and 2.2 by adjusting their values on the fly.

We compared the proposed algorithm to the popular learning algorithms - CM, NAG, Adadelta, AdaGrad, and Adam. We ran many hyperparameter settings for the momentum and adaptive gradient algorithms, and we also included results for the hyperparameter default values if the method implementation provided them. We tested the algorithms across ten different shallow neural network classifiers, three deep dense autoencoders, and a deep convolutional autoencoder. We were interested in the optimization capabilities of tested algorithms, so we focused on the training losses they achieved.

The results show that in almost all tested problems, any variant of ASDM2 achieved the best results. Both variants of ASDM2 generally achieved better results than their base algorithms, performing especially well on deep autoencoders. It can also be noted that tuning adaptive gradient methods' hyperparameters significantly improves their performance over default values, with their optimal values being orders of magnitude different from each other, depending on the architecture and task.

As the ASDM2 algorithm utilizes many estimators, its time and memory usage is higher than its base methods. The proposed method requires around 4 times more



memory than the CM method, and its one iteration is around 3-3.5 slower. However, it can be run only once, without prior hyperparameter optimization, saving much time. Also, the ASDM2 algorithm, through its recognition of the short-term and long-term influence of hyperparameters on the learning process, can adapt their values accordingly, improving the obtained optimization results.



## 6. Simultaneous machine translation

Simultaneous machine translation is a special case of a sequence-to-sequence transformation. Unlike NMT, which produces translated sequences after going through the whole input sequence, SMT methods create translated tokens while going through the input sequence, similar to human interpreters translating on the fly. This makes neural machine translation architectures like Encoder-Decoder (Bahdanau et al., 2014) or Transformers (Vaswani et al., 2017) not applicable without significant modifications. Also, a trade-off exists in SMT between delay and quality - the more input tokens we process, the more accurate output tokens become, at the cost of a longer generation delay. This trade-off needs to be optimized, usually with hyperparameters defining the usually constant delay between reading and writing tokens.

Ma et al. (2018) and Dalvi et al. (2018) propose modifications to NMT architectures for SMT tasks. Many SMT methods use reinforcement learning. Imitation learning is used in Grissom II et al. (2014) and Zheng et al. (2019). Gu et al. (2016) introduced a two-action framework where an interpreter agent can either read an input token or write an output token. The framework’s simplicity and generality make it a good baseline for SMT methods. Alinejad et al. (2018) expanded this framework with the third action, allowing the agent to predict the next input token instead of reading it.

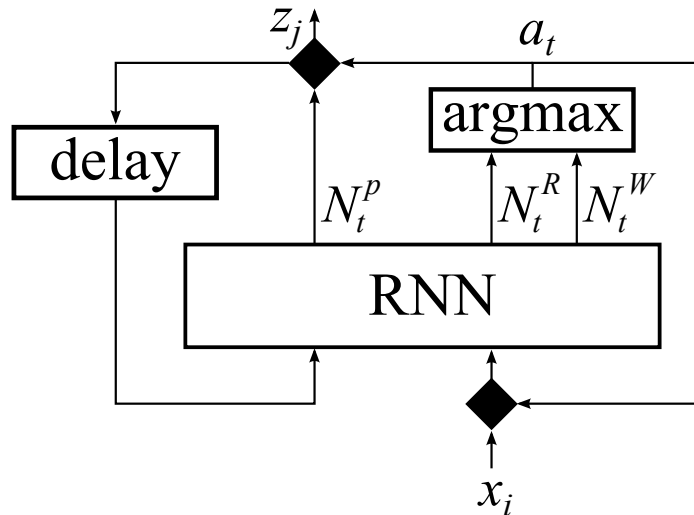
A delay-quality trade-off in SMT methods is usually controlled through hyperparameters. This necessitates finding the right values for such parameters with different types of sequences.

### 6.1. Proposed solution

In publication P4, we introduced a two-action SMT system called Reinforcement Learning for on-line Sequence Transformation (RLST). We model the simultaneous translation task as POMDP, with the agent receiving only the last read token or the last written token, depending on the previous action. The policy of this agent is a recurrent neural network (in the experiments, we used the GRU network, but other recurrent architectures can also be used). Its input consists of a read token and a written token from the previous step, with one of them set to the special NULL value, as only one action can be taken at any time. It produces the potential output

token and estimates of discounted rewards sum (returns) for read and write actions. We specify the training details of our proposed architecture, including optimization of translation quality and return estimates and weighting their losses, as their nominal values are of different scales. Including return estimates allows the RLST agent to control the delay-quality trade-off automatically, as it chooses the action that gives it the highest return during normal operation.

The architecture of the proposed system is presented in Figure 3. There,  $RNN$  denotes a policy of the agent in the form of the recurrent neural network,  $x_i$  is the  $i$ -th input token,  $z_j$  is the  $j$ -th output token created from the potential output token  $N_t^p$ ,  $N_t^R$  and  $N_t^W$  are expected returns for READ and WRITE actions, respectively, and  $a_t$  is the chosen action (READ or WRITE). If  $a_t = \text{READ}$ ,  $x_i$  is read, and if  $a_t = \text{WRITE}$ , the token  $z_j$  is created from  $N_t^p$ .



**Figure 3.** Architecture of the RLST system. The black squares represent passing/delaying  $x_i$  and outputting/skipping  $N_t^p$  depending on the action  $a_t$ .

We compare our proposed SMT architecture against NMT architectures, the Encoder-Decoder (Sutskever et al., 2014; Cho et al., 2014) and the Transformer (Vaswani et al., 2017), on machine learning task with several language pairs from Tatoeba and IWSLT2014 English-German pair. The comparison is done using the BLEU metric (Papineni et al., 2002) on a test subset, with the model to test chosen based on the highest BLEU score on the validation subset.

The results show that the proposed RLST approach achieves similar results to state-of-the-art NMT architectures, with significantly better performance on longer sequences, which implies it can store the context of longer sequences better than the other architectures. It is worth mentioning that RLST does this while controlling the delay and with limited input context. This proves that it can create

high-quality on-line translations comparable to state-of-the-art NMT solutions while automatically managing the delay-quality trade-off.



## 7. Deep state transformation in recurrent neural networks

While creating an SMT system described in publication P4, we noticed the shallow transformations of its recurrent unit state limited its performance. By enabling the translating agent to make any non-linear state transformation between each time step, the context of the already read and written tokens could be better registered, allowing the agent to achieve even better translation results.

Both LSTM and GRU cells allow only for a shallow, one-layer transformation of their states from one time step to another. While there are approaches to mitigate this issue, they either allow only a deep transformation of parts of the network (Graves, 2013), exhibit gradient propagation problems (Pascanu et al., 2013), or are very memory-consuming (Zilly et al., 2017).

In publication P5, we introduced the Deep Memory Update cell, which allows for its state’s arbitrary transformation between time steps. The cell follows the equations:

$$\begin{aligned} \langle z_t, \bar{h}_t \rangle &= FNN(x_t, h_{t-1}) \\ h_t &= h_{t-1} \odot \sigma(z_t) + \phi(\bar{h}_t) \odot (1 - \sigma(z_t)) \end{aligned}$$

where  $z_t$  is the memory preservation vector,  $\bar{h}_t$  is the hidden state change direction vector,  $FNN$  is the feed-forward neural network,  $\sigma$  is the unipolar soft step function (i.e. the logistic function), and  $\phi$  is the activation function (i.e. hyperbolic tangent) Using a feed-forward neural network that satisfies the universal approximation theorem (i.e., the two-layer perceptron from Section 2.2.1) allows the state to undergo arbitrary, non-linear transformation at any time.

The DMU cell structure makes it safe from gradient vanishing and exploding issues, which we proved mathematically. Also, the proposed cell has fewer parameters in its basic configuration than other recurrent architectures, as it only uses one gate.

To make the training process more stable, we proposed including a positive bias in the weight initialization of the DMU module to allow it to mostly preserve the memory state at the beginning of the training. We also introduced a learning rate

scaling method when the module is part of a larger network with the following equation:

$$\beta_{DMU} = \frac{\beta}{2n}$$

where  $\beta$  is the learning rate of the whole network,  $n$  is the number of layers in the FNN part of the cell and  $\beta_{DMU}$  is the learning rate for the DMU module. This scaling makes the DMU module learn slower but more stable as its depth increases, allowing the rest of the network to learn faster and improving the convergence time.

We test the DMU on several problems, including synthetic tasks from Hochreiter and Schmidhuber (1997) and problems with real-life data available, including neural machine translation. The results show that the proposed cell achieved comparable results to the other state-of-the-art recurrent modules, outmatching them in many cases. We also provide the learning rate ablation, showing that using our learning rate scaling method improves the results.

The DMU cell matches the performance of state-of-the-art recurrent architectures in various problems. In doing so, it is safe from gradient propagation problems, and its representation is memory-efficient, making it suitable for automating various tasks that utilize sequential data.



## 8. Final remarks

In this dissertation, we focused on task automation using artificial intelligence methods. We discussed four different tasks and presented approaches to automating them, and we also described a novel recurrent cell that can be used as a part of many task automations.

We used reinforcement learning to automate the trading process on a day-ahead energy market, enabling the trading agent to submit the collection of buy and sell bids every hour, allowing it to leverage price fluctuations and maximize its profits. The resulting bidding strategy can be used in real-life scenarios.

We proposed an SMT architecture learned with reinforcement, that is able to automatically control the delay of translation to maximize the translation quality without consuming the whole input sequence. We showed that it is able to match the performance of the state-of-the-art NMT architectures.

We prepared and tested various audio similarity matching systems for keyword spotting on a variety of datasets. We showed their strong performance in English and weak performance in Polish, with longer Polish words able to be identified more accurately.

We introduced a neural network learning algorithm that optimizes the learning hyperparameters on the fly by measuring their short-term and long-term influence on the learning process. The proposed method outperforms popular learning algorithms on the majority of tested problems.

Lastly, we designed a novel recurrent cell that is able to transform its state in an arbitrary way. This cell is also not susceptible to gradient propagation issues. We showed the performance of this cell to often outperform other recurrent networks while requiring fewer parameters.

In conclusion, we show that artificial intelligence methods can be used in tasks from various fields, often with very efficient operation and results.



## 9. Other achievements

Other achievements of the PhD candidate:

### Conference presentations

- “Reinforcement Learning for on-line Sequence Transformation”, 2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS 2022).
- “Widget detection on screenshots using computer vision and machine learning algorithms”, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019.

### Projects

- **Artificial intelligence applications in the energy sector**  
IX 2023 - currently  
Company: IDEAS NCBR  
PhD candidate scope of work: Research and development of different AI-based methods for the energy sector as part of the Learning in Control, Graphs and Networks team.
- **Complete coverage path algorithms for the cleaning robot**  
II 2023 - XII 2023  
Company: United Robots  
PhD candidate scope of work: Research and development of complete coverage path algorithms for the cleaning robot powered by machine learning methods.
- **AI-powered fashion stylist**  
II 2022 - II 2023  
Company: QuarticOn  
PhD candidate scope of work: Research and development on methods for AI-powered fashion stylist, technical advisory.
- **Recurrent neural networks for processing of sequential and graph data**  
I 2021 - XII 2023  
Project leader: Paweł Wawrzyński, PhD DSc.

PhD candidate scope of work: Development of a novel method for simultaneous machine translation and custom recurrent module, manuscript preparation (publications P2 and P5).

- **Simulating and analyzing methods of logistics networks for postal operators (LAS)**

II 2020 - XII 2020

Project leader: Rafał Biedrzycki, PhD DSc.

PhD candidate scope of work: Development of benchmark applications and simulators, code refactoring and optimization, code results unification in various programming languages, and benchmark reports.

- **Detection of words from a given list of terms in voice recordings together with an indication of the exact time stamp in order to improve the complaint handling process**

XI 2019 - XI 2020

Project leader: Robert Nowak, PhD DSc.

PhD candidate scope of work: Research and development of keyword spotting pipeline, preparation of a manuscript (publication P3) based on the project's outcome.

- **Identification of the GUI input-controls in the computer screenshots and movies**

XII 2018 - XII 2019

Project leader: Robert Nowak, PhD DSc.

PhD candidate scope of work: Research and development of widget detection methods based on computer vision and machine learning, manuscript preparation, and presentation.

## **Social activity**

- Presentation for AI science club "Golem" on applications of artificial intelligence in the energy sector. 09.05.2024

# Bibliography

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Alteschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Adam, M., Wessel, M., and Benlian, A. (2021). Ai-based chatbots in customer service and their effects on user compliance. *Electronic Markets*, 31(2):427–445.
- Alinejad, A., Siahbani, M., and Sarkar, A. (2018). Prediction improves simultaneous neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3022–3027.
- Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182. PMLR.
- Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. (2023). Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 1.
- Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. (2019). Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*.
- Attaviriyapap, P., Kita, H., Tanaka, E., and Hasegawa, J. (2005). New bidding strategy formulation for day-ahead energy and reserve markets based on evolutionary programming. *International Journal of Electrical Power & Energy Systems*, 27(3):157–167.
- Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Bakirtzis, A. G., Ziogos, N. P., Tellidou, A. C., and Bakirtzis, G. A. (2007). Electricity producer offering strategies in day-ahead energy market with step-wise offers. *IEEE Transactions on Power Systems*, 22(4):1804–1818.
- Baumann, T., Köhn, A., and Hennig, F. (2019). The spoken wikipedia corpus collection: Harvesting, alignment and an application to hyperlistening. *Language Resources and Evaluation*, 53:303–329.
- Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719.

- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.
- Bharadiya, J. P., Thomas, R. K., and Ahmed, F. (2023). Rise of artificial intelligence in business and industry. *Journal of Engineering Research and Reports*, 25(3):85–103.
- Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a " siamese" time delay neural network. *Advances in Neural Information Processing Systems*, 6.
- Changsong, C., Shanxu, D., Tao, C., Bangyin, L., and Jinjun, Y. (2009). Energy trading model for optimal microgrid scheduling based on genetic algorithm. In *2009 IEEE 6th International Power Electronics and Motion Control Conference*, pages 2136–2139. IEEE.
- Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems*, 34:15084–15097.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Cieślík, S. (2024). *Raport Otwarcia IV Kongresu Elektryki Polskiej*. Stowarzyszenie Elektryków Polskich.
- Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.
- Dalvi, F., Durrani, N., Sajjad, H., and Vogel, S. (2018). Incremental decoding and training methods for simultaneous translation in neural machine translation. *arXiv preprint arXiv:1806.03661*.
- Dong, Y., Dong, Z., Zhao, T., and Ding, Z. (2021). A strategic day-ahead bidding strategy and operation for battery energy storage system by reinforcement learning. *Electric Power Systems Research*, 196:107229.
- Dozat, T. (2016). Incorporating nesterov momentum into adam.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7).
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232.
- Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*,

- pages 1587–1596. PMLR.
- Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.
- Grissom II, A., He, H., Boyd-Graber, J., Morgan, J., and Daumé III, H. (2014). Don’t until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP)*, pages 1342–1352.
- Gu, J., Neubig, G., Cho, K., and Li, V. O. (2016). Learning to translate in real-time with neural machine translation. *arXiv preprint arXiv:1610.00388*.
- Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.
- Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.
- Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. John Wiley.
- Jogunola, O., Adebisi, B., Ikpehai, A., Popoola, S. I., Gui, G., Gačanin, H., and Ci, S. (2020). Consensus algorithms and deep reinforcement learning in energy market: A review. *IEEE Internet of Things Journal*, 8(6):4211–4227.
- Jogunola, O., Tsado, Y., Adebisi, B., and Nawaz, R. (2021). Trading strategy in a local energy market, a deep reinforcement learning approach. In *2021 IEEE Electrical Power and Energy Conference (EPEC)*, pages 347–352. IEEE.
- Jordan, M. (1986). Serial order: a parallel distributed processing approach. Technical report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- Lin, J., Kilgour, K., Roblek, D., and Sharifi, M. (2020). Training keyword spotters with limited and synthesized speech data. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7474–7478. IEEE.

- Liu, G., Xu, Y., and Tomsovic, K. (2015). Bidding strategy for microgrid in day-ahead market based on hybrid stochastic/robust optimization. *IEEE Transactions on Smart Grid*, 7(1):227–237.
- Ma, M., Huang, L., Xiong, H., Zheng, R., Liu, K., Zheng, B., Zhang, C., He, Z., Liu, H., Li, X., et al. (2018). Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. *arXiv preprint arXiv:1810.08398*.
- May, R. and Huang, P. (2023). A multi-agent reinforcement learning approach for investigating and optimising peer-to-peer prosumer energy markets. *Applied Energy*, 334:120705.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ . In *Dokl. akad. nauk Sssr*, volume 269, page 543.
- Okwuibe, G. C., Bhalodia, J., Gazafroudi, A. S., Brenner, T., Tzscheutschler, P., and Hamacher, T. (2022). Intelligent bidding strategies for prosumers in local energy markets based on reinforcement learning. *IEEE Access*, 10:113275–113293.
- Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE.
- Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.
- Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318. Pmlr.
- Perera, A. and Kamalaruban, P. (2021). Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618.
- Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.
- Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., and Shlens, J. (2019). Stand-alone self-attention in vision models. *Advances in Neural Information Processing Systems*, 32.
- Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S.



- (2020). Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51.
- Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.
- Robbins, H. and Siegmund, D. (1971). A convergence theorem for non negative almost supermartingales and some applications. In *Optimizing Methods in Statistics*, pages 233–257. Elsevier.
- Robinson, A. J. and Fallside, F. (1987). *The utility driven dynamic error propagation network*, volume 11. University of Cambridge Department of Engineering Cambridge.
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386.
- Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*, 30.
- Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Tieleman, T. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
- Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*.
- Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King’s

College, Cambridge, United Kingdom.

- Wawrzyński, P. (2017). Asd+ m: Automatic parameter tuning in stochastic optimization and on-line learning. *Neural Networks*, 96:1–10.
- Wen, F. and David, A. (2001). Strategic bidding for electricity supply in a day-ahead energy market. *Electric Power Systems Research*, 59(3):197–206.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.
- Yang, T., Zhao, L., Li, W., and Zomaya, A. Y. (2020). Reinforcement learning in sustainable energy and electric systems: A survey. *Annual Reviews in Control*, 49:145–163.
- Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.
- Zhang, Q., Lu, J., and Jin, Y. (2021). Artificial intelligence in recommender systems. *Complex & Intelligent Systems*, 7(1):439–457.
- Zheng, B., Zheng, R., Ma, M., and Huang, L. (2019). Simpler and faster learning of adaptive policies for simultaneous translation. *arXiv preprint arXiv:1909.01559*.
- Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2017). Recurrent highway networks. In *International Conference on Machine Learning*, pages 4189–4198. PMLR.
- Çam, E. (2024). Electricity 2024 – analysis and forecast to 2026. Technical report, International Energy Agency.

# **Appendices**

## A. List of Abbreviations

**A2C** – Advantage Actor-Critic  
**AI** – Artificial Intelligence  
**ASDM2** – Autonomous Stochastic Descent with Momentum, version 2  
**BPTT** – Backpropagation Through Time  
**CM** – Classic Momentum  
**DDPG** – Deep Deterministic Policy Gradient  
**DMU** – Deep Memory Update  
**DQN** – Deep Q-Network  
**EOS** – End of Sequence  
**FNN** – Fully-connected neural network  
**FOMDP** – Fully Observable Markov Decision Process  
**GPT** – Generative Pre-trained Transformer  
**GRU** – Gated Recurrent Unit  
**LLM** – Large Language Model  
**LSTM** – Long Short-Term Memory  
**MDP** – Markov Decision Process  
**NAG** – Nesterov Accelerated Gradient  
**NLP** – Natural Language Processing  
**NMT** – Neural Machine Translation  
**POMDP** – Partially Observable Markov Decision Process  
**PPO** – Proximal Policy Optimization  
**ReLU** – Rectified Linear Unit  
**RL** – Reinforcement Learning  
**RLST** – Reinforcement Learning for on-line Sequence Transformation  
**RNN** – Recurrent Neural Network  
**SAC** – Soft Actor-Critic  
**Seq2seq** – Sequence-to-sequence  
**SGD** – Stochastic Gradient Descent  
**SMT** – Simultaneous Machine Translation  
**SVM** – Support Vector Machine  
**TD3** – Twin Delayed Deep Deterministic Policy Gradient  
**VAE** – Variational Autoencoder

## **B. List of Publications**

## **B.1. Reinforcement learning meets microeconomics: Learning to designate price-dependent supply and demand for automated trading**

Title	Reinforcement learning meets microeconomics: Learning to designate price-dependent supply and demand for automated trading
Authors	Łukasz Lepak, Paweł Wawrzyński
Conference	<i>(accepted for publication)</i> European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2024 (ECML PKDD 2024)
Year	2024
Ministerial score	140

# Reinforcement learning meets microeconomics: Learning to designate price-dependent supply and demand for automated trading

Lukasz Lepak<sup>1,2</sup> (✉) and Paweł Wawrzyński<sup>2</sup>

<sup>1</sup> Warsaw University of Technology, Pl. Politechniki 1, 00-661 Warsaw, Poland  
lukasz.lepak.dokt@pw.edu.pl

<sup>2</sup> IDEAS NCBR, Chmielna 69, 00-801 Warsaw, Poland  
{lukasz.lepak,pawel.wawrzynski}@ideas-ncbr.pl

**Abstract.** The ongoing energy transition towards renewable sources increases the importance of energy exchanges and creates demand for automated trading tools on these exchanges. Day-ahead exchanges play a prominent role in this area. Participants in these exchanges place buy/sell bids collections before each trading day. However, machine learning-based approaches to automated trading are based on placing a single bid for each time instant. The bid is either executed or not, depending on the relation between the market price and the bid price. This is contrary to economic rationality, which usually requires buying more when the market price is lower and selling more when it is higher. Single bids do not allow the expression of such preferences. In this paper, we fill this gap and design a policy that translates the information available to the trading agent into price-dependent supply and demand curves. Also, we demonstrate how to train this policy with reinforcement learning and real-life data. Our proposed method is now being deployed in a real system for energy storage management. Here, we demonstrate how it performs in four data-driven simulations. The proposed method outperforms alternatives in all cases.

**Keywords:** Automated trading · Reinforcement learning · Energy market.

## 1 Introduction

In 2023, wind and solar energy represented 14.26% of global electricity generation, after these shares doubled in 5 preceding years [42]. The power of wind and sunlight reaching the Earth's surface is, to some extent, random. Therefore, while the rise of renewable energy sources presents the prospect of cheap and clean energy, it also exacerbates the problem of balancing power supply and demand.

In many countries, the main institution that balances volatile electricity supply and demand is a day-ahead energy market [13,14,27,30]. Every day, agents participating in this market place their buy and sell bids separately for every hour between 0 am and 11 pm the next day. Market clearing prices are then designated for each of these hours, and the bids are consequently executed or not, depending on the proposed prices.

Here, we consider an agent that (i) consumes electricity, (ii) produces electricity, and (iii) has electricity storage. What is of main interest here is a strategy for automated trading on a day-ahead energy market on behalf of this agent.

Reinforcement learning (RL) [32] is a natural tool to optimize a policy of sequential decision-making in dynamical, stochastic systems that elude modeling. RL has been applied to optimize strategies of on-line energy trading within local energy markets [4,15,20,21,24,28], real-time bidding for internet ads [6], stock market trading [18,41,10,38], power grid control [25,14,1], trading on the day-ahead energy market [8,9].

In existing studies on RL for automated trading, an action either selects a bid from a predefined set or directly defines parameters (type, price, and quantity) of a single bid or a pair (sell and buy) of bids.

The fact that for each bidding, the agent is able to submit only one or two bids is a serious limitation. Most electronic markets allow their participants to define many bids for each time interval. By submitting a collection of bids, the participant can define how much of the commodity he wishes to sell and/or buy, depending on the market price. The actual trading agents usually take advantage of this possibility since buying more when the price is low and selling more when the price is high usually results from economic rationality.

In this paper, we design a strategy that translates the information available to the trading agent into parameters of the supply and demand curves. These parameters are then translated into a collection of bids. The number of bids within the collection is variable. This strategy enables the trading agent to behave rationally in an economic sense, which is not possible when the strategy only produces single bids. We have designed our strategy with the day-ahead electricity market. However, it can also be applied to other electronic markets.

In this paper, we demonstrate the performance of our proposed automated trading strategy in several real data-based scenarios of the day-ahead electricity market trading. The strategy is currently being deployed in a real system for energy storage management.

The paper contributes as follows:

- We design a parametric automated trading strategy suitable for electronic markets with significant lags between bidding and its corresponding transaction. This strategy produces supply and demand curves by means of bid collections of variable sizes, thereby enabling the trading agent to behave rationally.
- We formalize a framework in which on-line RL can be applied to optimize a policy on the basis of recorded observations of the external environment without data on earlier decision-making.
- We apply reinforcement learning to optimize the above strategy and select the best algorithm for this purpose. The resulting strategy is fitted to the data and ready to use in real life.

## 2 Related Work

*Automated trading on the electricity market.* Research on automated trading on the electricity market covers various approaches. Some works introduce theoretical frameworks of bidding strategies [17,5,36]. Many authors propose various forms of parametric bidding strategies. These strategies are optimized with methods like linear program-



ming [3], genetic and evolutionary algorithms [37,2] or stochastic optimization [13,19]. However, as a more complex bidding strategy is expected and a more complex transformation of observations into bids is required, these techniques become less effective.

With the advent of electricity prosumers, energy microgrids, energy cooperatives, and flexible price-driven energy consumption, there is an increasing need for automated decision-making and control in various activities undertaken by the energy market participants. Strategies for these agents can be optimized with reinforcement learning. Various applications of RL in power systems are reviewed in [14,39,26]. The authors of [23] analyze bidding on a DA energy market as a zero-sum stochastic game played by energy producers willing to exercise their market power and keep their generators productive. RL is used there to optimize their bidding strategy. In [35], bidding on a DA energy market from the point of view of a flexible buyer (who charges a fleet of electric vehicles) is analyzed. His strategy is optimized with RL. A number of papers is devoted to peer-to-peer trading with electricity on a local, event-driven energy market, with RL applied to optimize the behavior of such peers [7,8,4,15,28]. RL and neural price predictions are used in [20] to optimize the scheduling of home appliances of private users. The authors assume that the electricity prices are changing and are known one hour ahead. The work [4] analyzes a similar setting in which the users also trade energy with each other. This setting is used in [28] to optimize the user strategies with multi-agent RL. The authors of [21] optimize peer-to-peer energy microgrid operations with multi-agent reinforcement learning, with their method generating higher net profits than simple fixed price biddings. Q-Learning and SARSA algorithms are used in [24] to create simple bidding strategies and test them on German real-life data.

The authors of [9] consider simultaneous trading on a DA and hour-ahead energy markets by an energy storage operator as a Markov Decision Process (MDP). The authors use RL to optimize a strategy of bidding on a DA energy market by a battery energy storage system. They use RL to optimize a strategy of bidding on a DA energy market by a battery energy storage system (BESS). However, the authors address the dynamics of that process only to a limited extent. Consecutive days are separate episodes, so the between-day dynamics of the market are not accounted for. Discrete actions define the parameters of the bids. They are not based on external observations such as weather forecasts. Also, only a single bid can be placed each hour. In the current paper, we address all of these limitations, which leads to significantly better performance of our proposed strategy and allows it to be deployed in real-life scenarios.

*Automated stock market trading.* In this area, the trading agent observes a set of time series of prices of different assets. The agent makes on-line decisions on buying these assets at the current prices in anticipation of their price increase or selling them in anticipation of their price decrease. Because the problem is formalized as an MDP, it is addressed with RL [10,40].

Additional related works are discussed in Appendix A of the supplementary material.

### 3 Problem definition

In this paper, we consider automated trading on the commodity markets with lags between biddings and their corresponding transactions. We specifically focus on the day-ahead energy market, understanding that other commodity markets could be approached alike, with some minor variations.

#### 3.1 Day-ahead electricity market

A trading agent is an entity such as a small- or medium-sized consumer of electricity e.g., a group of households connected together to the power network. We assume that it may consume electricity randomly, produce electricity with weather-dependent sources such as solar panels and windmills, and store energy in batteries.

The trading agent participates in the day-ahead energy market. Every day before 10.30 am<sup>3</sup> the agent submits bids for 24 separate biddings: for hours 0 am, 1 am, ..., 11 pm of the following day. Each bid is defined by the hour, type (sell/buy), price (per 1 kWh), and quantity (in kWhs). Any number of bids for each hour is acceptable. Right after the biddings close at 10.30 am, market prices are designated for each hour. The buy bids with prices higher than or equal to the market price will be executed at the market price. Likewise, the sell bids with prices lower than or equal to the market price will be executed at the market price. On the next day, at each hour, the agent consumes, produces, and transmits the energy to/from the power network according to its bids being executed. The net energy is transmitted to or released from the energy storage. When the agent tries to get energy from empty storage or put the energy into full storage, it actually exchanges it with the market and pays a special fine for that.

The problem is to designate the bids on behalf of the trading agent to maximize the profit gained (or minimize the cost incurred) from participation in the market.

#### 3.2 Reinforcement learning to bid

We adopt the general framework of reinforcement learning [32]. The objective is to optimize a policy that translates relevant available information into bids. The said information defines the state of the environment. It is relevant for future market prices, e.g., weather forecasts or the day of the week. Also, it is relevant to the current situation of the trading agent and its potential to produce and consume energy, e.g., battery charge and, again, weather forecasts.

Every day, the trading agent receives a reward equal to the financial net result of its bids (and fines). The goal is to optimize the policy to yield the largest possible sums of future discounted rewards in each environmental state the trading agent encounters.

### 4 Method

#### 4.1 Analysis

Within traditional microeconomics, we analyze the relation between the amount of goods the agent sells or buys and the unit price of these goods. If the agent is only able

<sup>3</sup> We take details from the specific DA market considered in the experimental study.

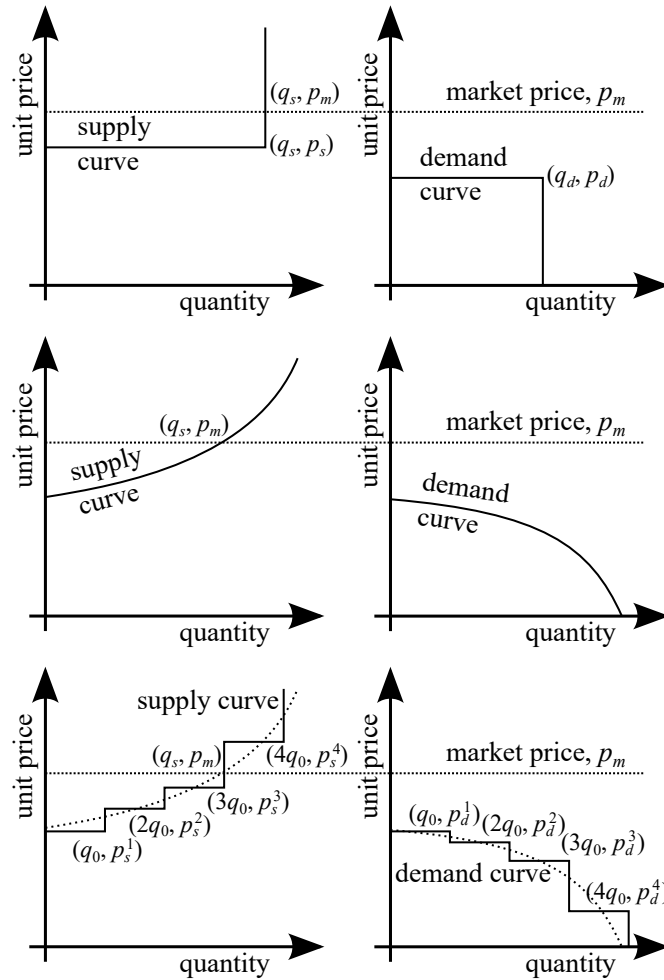


Fig. 1: *Top:* Supply and demand defined by a pair of bids; the agents sells  $q_s$  units at the unit price of  $p_m$ . *Middle:* Nondecreasing supply and nonincreasing demand. *Bottom:* Nondecreasing supply and nonincreasing demand as defined by a collection of bids.

to express its offered supply and demand in a pair of bids, the agent either sells/buys its defined quantity or not, depending on whether the market price is lower/higher than its defined threshold. The supply/demand curves that visualize these relations can be seen in the top part of Figure 1. To the best of our knowledge, placing a single bid, or a sell-and-buy pair of bids, at a time has only been considered in the literature of automated trading.

However, it is folklore of microeconomics [16] that a rational economic agent is most often willing to sell a higher quantity of commodity when its market price is higher. Also, the economic agent most likely is willing to buy a higher quantity of commodity when its market price is lower. For our considered trading agent, both the above cases create a lucrative opportunity to sell high and buy cheap. These typical preferences are depicted in the middle part of Figure 1, in the form of increasing supply curve and decreasing demand curve. How can the trading agent express such preferences with bids?

#### 4.2 Price-dependent supply and demand in bids

Let us consider, for a given hour  $h$ , a collection of sell bids

$$\langle \text{sell}, h, p_s^{h,i}, q_0 \rangle, \quad i = 1, \dots, n_s^h, \quad p_s^{h,i} \leq p_s^{h,i+1}, \quad (1)$$

where  $q_0 > 0$  is a certain constant quantity,  $n_s^h$  is the number of bids, and  $p_s^{h,i}$  are unit prices. Let  $p_m^h$  be a market price, and integer  $j$  be such that

$$p_s^{h,j} \leq p_m^h < p_s^{h,j+1}. \quad (2)$$

Then, only the first  $j$  bids are executed and the bidding agent sells a quantity of  $j q_0$  at the market price  $p_m^h$ . The above collection of bids (1) can thus be represented as a non-decreasing supply curve, similar to that depicted on the left-bottom part of Figure 1.

Any nondecreasing function can be approximated by a piecewise constant step function. Consequently, any reasonable preferences of selling can be approximately represented by the collection of bids (1). Moreover, for technical reasons, in most electronic markets, quantities can only be defined in bids as integer numbers (or as integer multiples of the minimum tradable quantity). Consequently, any supply curves feasible in the electronic market is a piecewise constant step function, and it can be represented in the form (1).

The above reasoning can be repeated, with similar conclusions, for demand. It can effectively be represented as a collection of bids in the form

$$\langle \text{buy}, h, p_d^{h,i}, q_0 \rangle, \quad i = 1, \dots, n_d^h, \quad p_d^{h,i} \geq p_d^{h,i+1}, \quad (3)$$

where  $n_d^h$  is the number of bids, and  $p_d^{h,i}$  are unit prices.

#### 4.3 Parametric representation of a collection of bids

In order to apply reinforcement learning to learn to designate collections of bids in the form (1) and (3), we need a way to translate vectors of predefined dimension into bid

collections of variable size. We design this translation as follows. Let the action space be 100-dimensional,  $a \in [-1, 1]^{100}$ . Coordinates of a single action define all bids for the whole day. The collection of sell bids for the hour  $h = 0, \dots, 23$  is given by (1) with

$$\begin{aligned} n_s^h &= \lfloor c_q \exp(c_e a_h) / q_0 + 1/2 \rfloor & (4) \\ p_s^{h,i} &= c_p^h \exp(a_{h+24}) \left( 1 + \exp(a_{96}) \left( -(2a_{98} + 4)^{-1} + (i/n_s^h)^{2a_{98}+3} \right) \right) & (5) \end{aligned}$$

The collection of buy bids for the hour  $h = 0, \dots, 23$  is given by (3) with

$$\begin{aligned} n_d^h &= \lfloor c_q \exp(c_e a_{h+48}) / q_0 + 1/2 \rfloor & (6) \\ p_d^{h,i} &= c_p^h \exp(a_{h+72}) \left( 1 + \exp(a_{97}) \left( (2a_{99} + 4)^{-1} - (i/n_d^h)^{2a_{99}+3} \right) \right) & (7) \end{aligned}$$

where  $a_k$  denotes  $k$ -th coordinate of the action  $a$ , and

- $a_h/a_{h+48}$  defines the width of the supply/demand curve, i.e., the number of sell/buy bids for the hour  $h$ ,
- $a_{h+24}/a_{h+72}$  defines the average height at which the supply/demand curve is located,
- $a_{h+24} + a_{96} / a_{h+72} + a_{97}$  defines vertical span of the supply/demand curve,
- $a_{98}/a_{99}$  defines convexity/concavity of the supply/demand curve,
- $c_q$  — quantity scaling factor (we assume its value equal to the maximum hourly production of the installed sources),
- $c_p^h$  — price scaling factor (we assume its value equal to the median price for hour  $h$  over the last 28 days),
- $c_e$  — quantity exponent scaling factor (we assume  $c_e = 3$ ).

The resulting supply and demand curves are depicted in Figure 2. Note that the above symbols, except  $q_0, c_q, c_e$ , depend on  $t$ , but we skip this dependence in the notation.

The supply and demand curves above are designed symmetrically. Thus, let us only analyze  $p_s^{h,i}$  (5). The term

$$-(2a_{98} + 4)^{-1} + (i/n_s^h)^{2a_{98}+3} \quad (8)$$

makes the supply curve an increasing power function with the exponent  $2a_{98} + 3$  controlling the convexity/concavity of the curve; for  $a_{98} \in [-1, 1]$  the exponent is in the  $[1, 5]$  interval. The component  $-(2a_{98} + 4)^{-1}$  makes the average of (8) over  $i \in [0, n_s^h]$  equal to zero. The term  $\exp(a_{96})$  controls a vertical span of the supply curve. The values of  $a_{96}$  and  $a_{98}$  do not impact the average height at which the supply curve is located, which is designated only by the term  $c_p^h \exp(a_{h+24})$ .

The widths and vertical locations of the curves are specified separately for different hours by their corresponding action coordinates. However, the vertical span of these curves and their convexity/concavity are specified for all hours by the same action coordinates  $a_{96} \dots a_{99}$ . This parameter sharing is intended to maintain a low enough dimensionality of the action space.

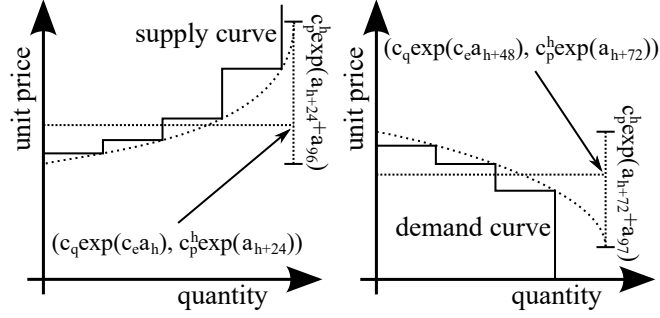


Fig. 2: Supply and demand defined by our proposed collections of bids.

#### 4.4 Bidding policy

In general in reinforcement learning, a *policy*,  $\pi$ , is a probability distribution of actions conditioned on states:

$$a_t \sim \pi(\cdot | s_t), \quad (9)$$

where  $s_t$  and  $a_t$  are, respectively, the state and the action at the instant  $t$  of discrete time.<sup>4</sup> We adopt a policy in the form

$$a_t = g^1(s_t; \theta) + \xi_t \circ \exp(g^2(s_t; \theta)), \quad \xi_t \sim \mathcal{N}(0, I), \quad (10)$$

where  $g^1$  and  $g^2$  are two vectors produced by the  $g$  neural network which is fed with the state  $s_t$  and parameterized by the vector  $\theta$  of trained weights; “ $\circ$ ” denotes the Hadamard (elementwise) product;  $\xi_t$  denotes random normal noise.

#### 4.5 Bidding policy optimization with reinforcement learning

Participation in the day-ahead market can be naturally modeled as a Markov Decision Process in which the state,  $s_t$ , of the environment at time  $t = 1, 2, \dots$  is a vector composed of two sub-vectors, *uncontrollable* variables  $s_t^u$ , and *controllable* variables  $s_t^c$ . The uncontrollable state variables denote external conditions like weather forecasts. They evolve according to an unknown stationary conditional probability

$$s_{t+1}^u \sim P(\cdot | s_t^u). \quad (11)$$

The controllable variables  $s_t^c$  are directly determined by the actions  $a_t$  taken and the uncontrollable state coordinates that is

$$s_{t+1}^c = f(s_t^c, a_t, s_t^u, s_{t+1}^u), \quad (12)$$

<sup>4</sup> In Section 4.3,  $a_k$  denoted  $k$ -th coordinate of action and here  $a_t$  denotes action at the time  $t$ .

where  $f$  is known. The key controllable state variable is the power storage charge. It trivially results from the agent's bids (actions) and uncontrollable variables: market prices and the agent's own energy production and consumption.

The critical assumption that allows us to distinguish uncontrollable and controllable variables is that the trading agent is small enough not to impact the market prices. Therefore, we may simulate its bidding and determine whether the bids are executed based on the recorded market prices. If the agent was large enough to actually impact the market prices, then this simulation would not be realistic, at least without an elaborate model of the impact of this agent on the market prices.

Note that the above-defined division of state variables into controllable and uncontrollable is unusual. In a typical MDP, we assume that the state changes according to

$$s_{t+1} \sim P_s(\cdot | s_t, a_t), \quad (13)$$

where the conditional probability  $P_s$  may be quite difficult to analyze and estimate. Therefore, a strategy of choosing actions cannot be evaluated without bias within a simulation based on a model of  $P_s$ .

Based on a recorded trajectory of uncontrollable states,  $(s_t^u : t = 1, \dots, T)$ , we can designate a strategy of selecting actions  $a_t$  based on states  $s_t$  and evaluate this strategy in a simulation with the record  $(s_t^u : t = 1, \dots, T)$  replayed. This valuation will be an unbiased estimate of the performance of this strategy deployed in reality. Furthermore, we can replay this record repeatedly and simulate episodes of on-line RL just using  $f$  (12) to designate consecutive values of  $s_t^u$ .

In order to optimize the strategy (10), we may use any algorithm of on-line reinforcement learning [33] e.g., A2C [22], PPO [31] or SAC [11]. In the experiments below, we used the A2C algorithm, which showed the best stability by far. Our comparison of RL algorithms is presented in Appendix G of the supplementary material. A training consists of a sequence of simulated trials in which the trajectory of uncontrollable states is just replayed from the data, and the corresponding trajectory of controllable states is designated based on the uncontrollable states, the actions selected, and the function  $f$  (12).

#### 4.6 Alternative bidding strategies

In order to verify our proposed bidding strategy, we compare it to two more intuitive ones.

*Simple arbitrage strategy.* Perhaps the simplest conceivable bidding strategy is to buy energy when it is cheap, keep it in the battery, and sell it when it is expensive. On most days, the market value of electricity is the lowest at 2 am, and it is the highest at 10 am. Therefore, our reference simple arbitrage strategy assumes placing the two bids:

$$\langle \text{buy}, 2am, +\infty, \theta_1 - \hat{l} \rangle, \quad \langle \text{sell}, 10am, -\infty, \theta_2 \rangle, \quad (14)$$

where  $\hat{l}$  is an estimated storage state of charge at 0 am, and  $\theta_1, \theta_2$  are optimized parameters. We apply the CMA-ES evolutionary algorithm [12] for their optimization.

*Pair of bids strategy.* A simple approach to bidding on the day-ahead electricity market, which also involves reinforcement learning, is to present just two bids for each hour  $h = 0am, \dots, 11pm$ , namely

$$\langle \text{buy}, h, p_d^h, n_d^h q_0 \rangle, \quad \langle \text{sell}, h, p_s^h, n_s^h q_0 \rangle, \quad (15)$$

where  $p_d^h, n_d^h, p_s^h$  and  $n_s^h$  are defined by an action,  $a \in [-1, 1]^{96}$ , as follows:

$$n_d^h = \lfloor c_q \exp(c_e a_{h+48}) / q_0 + 1/2 \rfloor, \quad p_d^{h,i} = c_p^h \exp(a_{h+72}), \quad (16)$$

$$n_s^h = \lfloor c_q \exp(c_e a_h) / q_0 + 1/2 \rfloor, \quad p_s^{h,i} = c_p^h \exp(a_{h+24}). \quad (17)$$

For comparison, see  $n_d^h$  (6),  $p_d^{h,i}$  (7),  $n_s^h$  (4),  $p_s^{h,i}$  (5). The collection of bids strategy introduced in Section 4.2 would be equivalent to (16) and (17), if all buy bids for a given hour had equal price and all sell bids for a given hour had equal price. In our simulations, we use the same reinforcement learning setup to train strategies that place the above pairs of bids and the collections of bids introduced in Section 4.3.

## 5 Simulations

### 5.1 Simulation environment

Experiments are conducted using a custom environment simulating day-ahead energy market operations. This simulator is based on real-life data from the Polish market. It allows for customization of various market settings, such as a bid creation time, a scale of the trading agent (defined by the number of households), or its solar and wind energy generation capabilities. The environment is based on the Gymnasium environment interface [34], making it compatible with popular reinforcement learning libraries, including Stable-Baselines3 [29], which we use as our source of RL algorithms.

We provide details and parameters on the simulation environment, the trading agent's energy consumption and production profile, and weather forecast randomization in Appendices B-E of the supplementary material.

We run our experiments by replaying the events that occurred in the years 2016-2019. We selected this period as preceding the COVID-19 pandemic, which destabilized markets. The runs involve replaying original price data and weather data. In order to diversify every replay and thus avoid overfitting to the data, we randomize weather forecasts and electricity demand according to their statistical profile.

During the simulation, the trading agent may be forced to buy missing energy or sell excess energy immediately. It happens when the agent sells or uses energy it does not have or buys energy it does not have room for. The agent is being penalized for such events. Immediate buying is realized for double the current market price, and immediate selling is realized for half the current market price so that the agent has the incentive to better plan its bids instead of relying on instant buys or sells. Also, we do not include market entry and transaction fees, as they are fixed costs independent of the bidding strategy.



## 5.2 Experiments

Reinforcement learning is used to optimize the bidding policy for a collection of bids parameterized as in Section 4.3, later referred to as COLLECTION. It utilizes data from 2016 to the third quarter of 2018 as the training set, data from the fourth quarter of 2018 as the validation set, and data from 2019 as the testing set. The training is done in randomly generated intervals from the training set, which are 90 days long. Periodically, evaluation is done on a single validation interval 90 days long. After the training timesteps budget is depleted, the model for which the highest reward on validation interval was achieved is evaluated on the single testing interval 365 days long. Common parameters used for the RL experiments are available in Table 2 of the supplementary material.

The observation of the environment’s state (117 values) is passed to the agent at bid placing time and contains the following information:

- prices of energy at the current day for every hour (24 values) – these are the prices for the current day, for which the bids were created the day before; the agent does not know energy prices for the bids currently submitted,
- current relative battery charge (1 value),
- estimated relative battery charge at midnight (1 value),
- one-hot encoded information about the current month (12 values),
- one-hot encoded information about the current day of the week (7 values),
- cloudiness, wind speed, and temperature forecasts for each hour of the next day (72 values).

Rewards are computed as

$$r_t = 10^{-3} (p_t - \bar{p}_t - \rho_t), \quad (18)$$

where  $p_t$  is the daily profit from selling and buying energy,  $\bar{p}_t$  is a reference profit, and  $\rho_t$  is a regularizing penalty. The reference profit  $\bar{p}_t$  is a daily profit that would be achieved if the difference between daily produced and consumed energy was sold or bought at the average market price from that day. The reference profit is not trivial to achieve since the agent mostly consumes energy when it is expensive and produces energy when it is cheap. The regularizing penalty

$$\rho_t = \sum_{i=1}^{\dim(a_t)} [|a_{t,i}| > 0.99] \quad (19)$$

where  $[condition]$  equals 1 if the condition is true, else 0, prevents the action coordinates from saturating at their bounds. The effect of regularization on the performance of tested strategies is presented and discussed in Appendix I of the supplementary material.

We compare the collection of bids strategy to the strategy to the alternative strategies presented in Section 4.6. The simple arbitrage strategy is later referred to as ARBITRAGE, and the pair of bids strategy is later referred to as PAIR.

We also applied the algorithm from [9], later referred to as FARL, which is a conceptually different approach to optimize a bidding strategy. FARL considers each day

a 24-step episode and places a single sell/buy bid at each hour. FARL is based on the assumption that each bid is placed when market prices for preceding hours are known. This assumption is wrong for any day-ahead electricity market we are aware of. We used this algorithm to produce bids for consecutive hours without access to the market prices of previous biddings. We fed it with the same training, evaluation, and test data as discussed above. However, when used this way, it was unable to produce even remotely reasonable strategy. Implementation details, parameters, and discussion about the FARL algorithm are provided in Appendix F of the supplementary material.

### 5.3 Different operation scenarios

We tested the proposed collection of bids strategy in comparison to the alternatives from Section 4.6 in the following scenarios:

- an agent has an energy storage only (BES),
- an agent has an energy storage and production capabilities (BES+PROD),
- an agent has an energy storage and consumes energy (BES+CON),
- an agent has an energy storage, produces and consumes electricity (ALL).

### 5.4 Results

Scenario \ strategy	ARBITRAGE	PAIR	COLLECTION	Reference
BES	13251.29 ± 6238.36	30791.40 ± 851.31	<b>32826.71</b> ± 1127.88	0.00
BES+PROD	17578.67 ± 8039.00	28388.18 ± 765.45	<b>29170.47</b> ± 1630.18	37470.07
BES+CON	3446.76 ± 7042.91	28485.64 ± 667.74	<b>28547.43</b> ± 1154.93	-45089.87
ALL	16217.54 ± 6677.35	30203.35 ± 644.51	<b>31036.12</b> ± 1310.28	-7619.80

Table 1: Differences between achieved balances and the reference profit for the tested strategies in different scenarios; last column contains the reference.

Table 1 presents differences between total profits achieved by tested strategies and the total reference profit described above; the last column contains the reference. It is seen that depending on the scenario, the reference varies a lot because the trading agent either sells the energy produced, buys the energy consumed or does both or neither. The proposed collection of bids strategy achieved the best profits, beating the pair of bids strategy in all tested scenarios. The pair of bids strategy achieved reasonable results but slightly worse than the proposed strategy.

Of all tested scenarios, the collection of bids achieved the best advantage over the pair of bids strategy in the battery-only scenario. Here, the agent earns money solely based on bids created, without any production or consumption to include in the bids. It is noticeable that the collection of bids strategy is able to adapt to these circumstances, making the biggest buys when the energy price is low and the biggest sells when the energy price is high, with some additional smaller transactions also happening in beneficial hours. This means that the collection of bids strategy is able to recognize significant price fluctuations, allowing it to capitalize on occasional prices.

In all of the tested scenarios, both strategies were able to adapt to the circumstances, buying enough energy when only consumption was active and selling surpluses of energy when only production was active. Immediate transactions due to lack or excess of energy were, in fact, very rare.

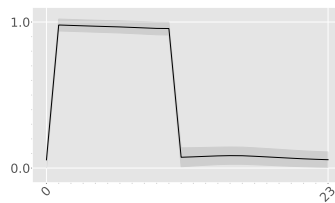


Fig. 3: Mean hourly relative battery charge. Strategy: COLLECTION. Scenario: ALL.

In Figure 3, the mean hourly relative charge for the battery is presented. These were calculated for the COLLECTION strategy based on the test run that achieved the best profit. The proposed strategy is able to make the best use of its available capacities, with smooth transitions between hours, indicative of reasonable bid creation. It is seen that the battery is charged at night, which means that the agent buys energy when it is cheap. The battery is discharged at about 10 am, which means that the agent sells energy when it is the most expensive. The PAIR strategy is generally able to leverage that regularity and achieve reasonable profits. However, our proposed COLLECTION strategy is also able to leverage unpredictable variations of prices to the agent's benefit: It buys more when the prices are unexpectedly low and sells more when the prices are unexpectedly high.

The supplementary material attached to this paper contains the following:

- Appendix A - additional related works
- Appendix B - details and parameters of the simulation environment
- Appendix C - model of the trading agent's energy consumption
- Appendix D - model of the trading agent's energy production
- Appendix E - model for creating weather forecasts from real weather data
- Appendix F - description of adapting the FEARL algorithm [9] to our simulation environment
- Appendix G - comparison of other RL algorithms (PPO, SAC, TD3) together with their hyperparameters
- Appendix H - detailed results for the pair of bids strategy
- Appendix I - study of using regularization in the collection of bids and the pair of bids strategies
- Plots for the collection of bids and the pair of bids strategies with different scenarios.

## 6 Conclusions

In this paper, we have proposed a parametrization of supply and demand curves, which allows for multiple sell and buy bids at each time, thus introducing increased flexibility and efficiency to automated trading on electronic markets. We have described a framework for optimization of this parametrized bidding strategy on a day-ahead energy market based on simulations and real-life data. We have used reinforcement learning to optimize this strategy and have compared it with different strategies. The proposed collection of bids strategy achieved the best results, getting the highest financial profit while showing reasonable behavior with battery management and bid placement.

The proposed strategy's generality and adaptability to data allow it to be deployed in real life. Indeed, the strategy is now being deployed in a system for energy storage management.

**Data availability.** Data resources are available at the following link:  
[https://github.com/Bestest96/ecml24\\_rl4trade](https://github.com/Bestest96/ecml24_rl4trade).

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Antonopoulos, I., Robu, V., Couraud, B., Kirli, D., Norbu, S., Kiprakis, A., Flynn, D., Elizondo-Gonzalez, S., Wattam, S.: Artificial intelligence and machine learning approaches to energy demand-side response: A systematic review. *Renewable and Sustainable Energy Reviews* **130**, 109899 (2020)
2. Attaviriyapap, P., Kita, H., Tanaka, E., Hasegawa, J.: New bidding strategy formulation for day-ahead energy and reserve markets based on evolutionary programming. *International Journal of Electrical Power & Energy Systems* **27**(3), 157–167 (2005)
3. Bakirtzis, A.G., Ziogos, N.P., Tellidou, A.C., Bakirtzis, G.A.: Electricity producer offering strategies in day-ahead energy market with step-wise offers. *IEEE Transactions on Power Systems* **22**(4), 1804–1818 (2007)
4. Bose, S., Kremers, E., Mengelkamp, E.M., Eberbach, J., Weinhardt, C.: Reinforcement learning in local energy markets. *Energy Informatics* **4**(1), 1–21 (2021)
5. Castellini, M., Di Corato, L., Moretto, M., Vergalli, S.: Energy exchange among heterogeneous prosumers under price uncertainty. *Energy Economics* **104**, 105647 (2021)
6. Chen, S., Xu, Q., Zhang, L., Jin, Y., Li, W., Mo, L.: Model-based reinforcement learning for auto-bidding in display advertising. In: *Autonomous Agents and Multiagent Systems (AAMAS)* (2023)
7. Chen, T., Su, W.: Indirect customer-to-customer energy trading with reinforcement learning. *IEEE Transactions on Smart Grid* **10**(4), 4338–4348 (2018)
8. Chen, T., Su, W.: Local energy trading behavior modeling with deep reinforcement learning. *IEEE access* **6**, 62806–62814 (2018)
9. Dong, Y., Dong, Z., Zhao, T., Ding, Z.: A strategic day-ahead bidding strategy and operation for battery energy storage system by reinforcement learning. *Electric Power Systems Research* **196**, 107229 (2021)

10. Gao, S., Wang, Y., Yang, X.: Stockformer: Learning hybrid trading machines with predictive coding. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 4766–4774 (2023)
11. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning (ICML). pp. 1861–1870 (2018)
12. Hansen, N.: The cma evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772 (2016)
13. Iria, J.P., Soares, F.J., Matos, M.A.: Trading small prosumers flexibility in the day-ahead energy market. In: 2017 IEEE Power & Energy Society General Meeting. pp. 1–5. IEEE (2017)
14. Jogunola, O., Adebisi, B., Ikpehai, A., Popoola, S.I., Gui, G., Gačanin, H., Ci, S.: Consensus algorithms and deep reinforcement learning in energy market: A review. IEEE Internet of Things Journal **8**(6), 4211–4227 (2020)
15. Jogunola, O., Tsado, Y., Adebisi, B., Nawaz, R.: Trading strategy in a local energy market, a deep reinforcement learning approach. In: 2021 IEEE Electrical Power and Energy Conference (EPEC). pp. 347–352. IEEE (2021)
16. Kolmar, M.: Principles of Microeconomics. Springer Nature Switzerland AG (2021)
17. Lamont, J.W., Rajan, S.: Strategic bidding in an energy brokerage. IEEE transactions on power systems **12**(4), 1729–1733 (1997)
18. Lee, N., Moon, J.: Transformer actor-critic with regularization: Automated stock trading using reinforcement learning. In: Autonomous Agents and Multiagent Systems (AAMAS) (2023)
19. Liu, G., Xu, Y., Tomsovic, K.: Bidding strategy for microgrid in day-ahead market based on hybrid stochastic/robust optimization. IEEE Transactions on Smart Grid **7**(1), 227–237 (2015)
20. Lu, R., Hong, S.H., Yu, M.: Demand response for home energy management using reinforcement learning and artificial neural network. IEEE Transactions on Smart Grid **10**(6), 6629–6639 (2019)
21. May, R., Huang, P.: A multi-agent reinforcement learning approach for investigating and optimising peer-to-peer prosumer energy markets. Applied Energy **334**, 120705 (2023)
22. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning (2016), arXiv:1602.01783
23. Nanduri, V., Das, T.K.: A reinforcement learning model to assess market power under auction-based energy pricing. IEEE transactions on Power Systems **22**(1), 85–95 (2007)
24. Okwuibe, G.C., Bhalodia, J., Gazafroudi, A.S., Brenner, T., Tzscheutschler, P., Hamacher, T.: Intelligent bidding strategies for prosumers in local energy markets based on reinforcement learning. IEEE Access **10**, 113275–113293 (2022)
25. Orfanoudakis, S., Chalkiadakis, G.: A novel aggregation framework for the efficient integration of distributed energy resources in the smart grid. In: Autonomous Agents and Multiagent Systems (AAMAS) (2023)
26. Perera, A., Kamalaruban, P.: Applications of reinforcement learning in energy systems. Renewable and Sustainable Energy Reviews **137**, 110618 (2021)
27. Prabavathi, M., Gnanadass, R.: Energy bidding strategies for restructured electricity market. International Journal of Electrical Power & Energy Systems **64**, 956–966 (2015)
28. Qiu, D., Wang, J., Wang, J., Strbac, G.: Multi-agent reinforcement learning for automated peer-to-peer energy trading in double-side auction market. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 2913–2920 (2021)
29. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)

30. Rahimiyan, M., Baringo, L.: Strategic bidding for a virtual power plant in the day-ahead and real-time markets: A price-taker robust optimization approach. *IEEE Transactions on Power Systems* **31**(4), 2676–2687 (2015)
31. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017), arXiv:1707.06347
32. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. Second edition. The MIT Press (2018)
33. Sutton, R.S., Singh, S.P., McAllester, D.A.: Comparing policy-gradient algorithms (2001)
34. Towers, M., Terry, J.K., Kwiatkowski, A., Balis, J.U., Cola, G.d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Shen, A.T.J., Younis, O.G.: *Gymnasium* (Mar 2023). <https://doi.org/10.5281/zenodo.8127026>, <https://zenodo.org/record/8127025>
35. Vandael, S., Claessens, B., Ernst, D., Holvoet, T., Deconinck, G.: Reinforcement learning of heuristic ev fleet charging in a day-ahead electricity market. *IEEE Transactions on Smart Grid* **6**(4), 1795–1805 (2015)
36. Vytelingum, P., Ramchurn, S.D., Voice, T.D., Rogers, A., Jennings, N.R.: Trading agents for the smart electricity grid. In: *The Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)* (10/05/10 - 14/05/10). pp. 897–904 (2010), <https://eprints.soton.ac.uk/268361/>, event Dates: May 10-14, 2010
37. Wen, F., David, A.: Strategic bidding for electricity supply in a day-ahead energy market. *Electric Power Systems Research* **59**(3), 197–206 (2001)
38. Yang, M., Zhu, M., Liang, Q., Zheng, X., Wang, M.: Spotlight news driven quantitative trading based on trajectory optimization. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 4930–4939 (2023)
39. Yang, T., Zhao, L., Li, W., Zomaya, A.Y.: Reinforcement learning in sustainable energy and electric systems: A survey. *Annual Reviews in Control* **49**, 145–163 (2020)
40. Zhong, Y., Bergstrom, Y., Ward, A.: Data-driven market-making via model-free learning. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 4461–4468 (2020)
41. Ziyi Xu, Xue Cheng, Y.H.: Performance of deep reinforcement learning for high frequency market making on actual tick data. In: *Autonomous Agents and Multiagent Systems (AAMAS)* (2022)
42. Çam, E.: *Electricity 2024 – analysis and forecast to 2026*. Tech. rep., International Energy Agency (2024)

# Reinforcement learning meets microeconomics: Learning to designate price-dependent supply and demand for automated trading Supplementary material

Lukasz Lepak<sup>1,2</sup> (✉) and Paweł Wawrzyński<sup>2</sup>

<sup>1</sup> Warsaw University of Technology, Pl. Politechniki 1, 00-661 Warsaw, Poland  
lukasz.lepak.dokt@pw.edu.pl

<sup>2</sup> IDEAS NCBR, Chmielna 69, 00-801 Warsaw, Poland  
{lukasz.lepak,pawel.wawrzynski}@ideas-ncbr.pl

## A Other related work

*Automated stock market trading.* As asset prices are driven by market news, the transformer may be applied to handle this news and impact the transactions [6]. However, on the energy market, the asset (energy) can not be bought or sold at the current price. Prices for a whole day are only revealed at the same time after the bidding.

*Online auto-bidding on advertising markets.* In this area, trading agents are repeatedly offered an opportunity to show an advertisement to a defined internet user [3,1,4]. The agents bid, and then the user is presented with the advertisement of the agent that bid the highest. The problem is for the agent to translate the user's features into the bidding price to maximize the expected net utility of the advertising. However, on the energy market, the agent needs to place a collection of bids at each time, which are interdependent within each collection and between consecutive collections.

## B Simulation environment

Table 1 depicts common environment settings used in our experiments. We set the action scheduling time to match the Polish day-ahead energy market. Battery and solar panel efficiencies reflect the efficiencies of real-life batteries and solar panels. Wind energy and solar energy limits are tuned so that daily energy production in the environment averages around 1 MWh. The number of households is set to 100 to scale the simulation for a medium-sized manufacturing facility, an energy cooperative, or a small power plant.

In our experiments, we use real historical data from the following sources:

- energy prices – Polish day-ahead energy market (Fixing I),
- weather data – Polish Meteorology and Water Management Institute,
- average energy consumption – Polish Central Statistical Office.

Common parameters used for the experiments with RL strategies are presented in Table 2.

Action scheduling time	10.30 am
Battery capacity	2 MWh
Battery efficiency	90%
Maximum solar panels' power	120 kW
Maximum wind turbine power	50 kW
Maximum wind turbine operation speed	11 m/s
Mean household power consumption	224 W
Number of households	100

Table 1: Parameters of the simulation environment used for experiments.

### C Energy consumption

Energy consumption for the given hour ( $E_c^h$ ) is calculated as follows:

$$E_c^h = n \cdot E_{c,avg}^h \cdot |1 + \rho| \quad (1)$$

where  $E_{c,avg}^h$  is the average energy consumption per one household for the given hour,  $n$  is the number of households, and  $\rho \sim \mathcal{N}(0, 0.03)$  allows the resulting energy consumption to differ each day while maintaining the average value. Equation (1) is prepared to scale well with the changing number of households.

### D Energy production

Solar energy production for the given hour ( $E_s^h$ ) is based on cloudiness value from the actual weather data and is calculated as follows:

$$E_s^h = s_{max} \cdot (2/3) \max\{-\cos(2\pi h/24) - 0.5 \cos(2\pi d/365), 0\} \cdot (1 - c/8) \quad (2)$$

where  $s_{max}$  is the maximum solar energy generation,  $c \in \{0, 1, \dots, 7, 8\}$  is the cloudiness value in Oktas (0 - clear sky, 8 - heavy overcast) taken from the weather data and  $d$  is the day of the year.

Wind energy production for the given hour ( $E_w^h$ ) is based on the actual wind speed value from the weather data and is calculated as follows:

$$E_w^h = w_{max} \cdot \frac{ws}{ws_{max}} \cdot [ws \leq ws_{max}] \quad (3)$$

where  $w_{max}$  is the maximum wind energy generation,  $ws$  is the wind speed,  $ws_{max}$  is the maximum wind speed for which the wind turbines are still operational, and  $[cond] = 1$  when  $cond$  is true, else 0.

### E Weather forecasts model generation

We start each day at 10 am of the previous day in the actual data. For every hour from 11 am of the previous day to 11 pm of the currently forecasted day, we generate the



Algorithm	A2C
Timesteps	4 500 000
Evaluation frequency	9 000
Episode length	90
Action space	$[-1, 1]^{100}$
Observation space	117 normalized
Reward space	$(-\infty, \infty)$
Learning rate ( <i>learning_rate</i> )	0.0001
Number of update steps ( <i>n_steps</i> )	90
Discount ( <i>gamma</i> )	0.9
GAE coefficient ( <i>gae_lambda</i> )	0.9
Entropy coefficient ( <i>ent_coef</i> )	0.0
Value function coefficient ( <i>vf_coef</i> )	0.5
RMSprop as optimizer ( <i>use_rms_prop</i> )	True
RMSprop epsilon ( <i>rms_prop_eps</i> )	0.00001
Use gSDE ( <i>use_sde</i> )	False
Hidden layers neurons ( <i>net_arch</i> )	300
Log standard deviation init ( <i>log_std_init</i> )	-1
Normalize input ( <i>normalize_images</i> )	False
Activation function ( <i>activation_fn</i> )	tanh
Orthogonal initialization ( <i>ortho_init</i> )	True

Table 2: Parameters of the A2C algorithm used for experiments. Names in brackets are taken from the Stable-Baselines3 library [5]. Parameters not present in this table use default values. Neural network architecture is the same for actor and critic networks.

forecasts as follows:

$$\begin{aligned}\epsilon_t &\sim \mathcal{N}(0, \sigma^2/24) \\ d_t &= \sum_{i=1}^t \epsilon_i \\ x_t^{forecast} &= x_t^{actual} + d_t\end{aligned}\tag{4}$$

where  $\sigma$  is an accuracy of a 24-hour forecast,  $d_t$  is a deviation for index  $t$  and  $x_t^{actual}$ ,  $x_t^{forecast}$  are actual and forecasted weather for index  $t$ , respectively. For cloudiness, we assume  $\sigma = 2$  Oktas; for wind speed, we assume  $\sigma = 1$  m/s, and for temperature we assume  $\sigma = 2$  °C. Here,  $t = 0$  denotes 10 am of the previous day, and we are interested in  $t \in [14, 37]$ , i.e., next-day forecasts. Cloudiness forecasts are clipped and projected to the nearest integers, while wind speed forecasts are clipped to be at least zero.

## F FARL

The FARL algorithm from [2] optimizes discrete actions. Thus, in order to use it with our environment, we performed action discretization in the following way:

- six capacity levels ( $\{0, 1, 2, 3, 4, 5\}$ ), denoting the number of batteries we want to load (for buy bids) or unload (for sell bids), seven price levels ( $\{0, \frac{c_p^h}{3}, \frac{2c_p^h}{3}, \frac{3c_p^h}{3}, \frac{4c_p^h}{3}, \frac{5c_p^h}{3}, +\infty\}$ ), where  $c_p^h$  is the price scaling factor, equal to the median price for hour  $h$  over the last 28 days), separate bids for buying and selling, which gives  $6 \cdot 7 \cdot 2 = 84$  different actions. Note that many actions refer to doing nothing, e.g., selling/buying zero capacity at different prices.

We used observations matching the original paper, which are:

- price from the same hour of the previous day ( $p_p$ , normalized to range  $[-1, 1]$ )
- action from the previous hour ( $a_{t-1}$ , volume normalized to range  $[-1, 1]$  (+ for buy bid, - for sell bid), price normalized to range  $[-1, 1]$ )
- relative battery states' estimates at 0 am of the trading day ( $b_t$ , range  $[0, 1]$  for each battery)
- current hour ( $t$ , provided as a one-hot vector with 1 on the current hour index (0 – 23), else 0)

Note that in order to produce bids for each hour of the trading day, this algorithm is run in a 24-step episode. Also, this algorithm does not take into account any external information, such as weather forecasts. We have also prepared a wrapper for our original environment, which converts actions and observations and allows the FARL algorithm to be executed with timesteps representing one hour instead of one day.

Parameters of the FARL algorithm are presented in Table 3. The discount is set to 1 to match it with the original paper. The number of timesteps is set to match the number of days seen throughout the training, as the environment for FARL uses hours as timesteps instead of days.

Timesteps	108 000 000
Evaluation frequency	2400
Episode length	24
Action space	84, discrete
Observation space	32, normalized
Reward space	$(-\infty, \infty)$
Learning rates ( $\alpha, \beta$ )	0.0001
Discount ( $\gamma$ )	1.0
Exploration rate ( $\epsilon$ )	1.0 - 0.1, linear decrease during 10% of timesteps, later constant

Table 3: Parameters of the FARL algorithm used for experiments.

## G Comparison of RL algorithms

We tested four RL algorithms: A2C, PPO, SAC, and TD3. Parameters for the A2C algorithm are in Table 2, and the rest of the algorithms' parameters are presented in Tables 4-6. We tested them on the collection of bids strategy on the ALL scenario. Table 7 presents the results of all tested algorithms. The A2C algorithm achieved the best results, with PPO being behind but still over the reference profit described in the main part of the paper. SAC and TD3 algorithms achieved disappointing results, failing to learn how to bid efficiently on the simulated market. The TD3 algorithm performed very poorly, resulting in significant money loss.

We interpret the above results as follows. A2C and PPO are based on  $n$ -step returns, while SAC and TD3 are based on 1-step returns.  $n$ -step returns compensate for non-Markovian characteristics of the market environment. Also, action is highly multi-dimensional in the considered problem. Being a purely on-line algorithm, A2C worked better with such actions than PPO. The latter algorithm performs more than one policy optimization step based on the collected experience. It is, therefore, not a purely on-line algorithm.

## H Pair of bids strategy

Figure 3 presents the mean hourly relative battery charges from the best run of the pair of bids strategy. The strategy works reasonably well, buying the energy when it is cheap and selling it when it is expensive. In that regard, it behaves similarly to the collection of bids strategy. However, it is not able to benefit from random price variations. This may be seen in Figure 4, where strategy usually places similar buy and sell thresholds and amounts at given hours. Because of this, when the price change is significant, the strategy will either sell or buy the whole requested volume at the beneficial price or do nothing at all. That is why the collection of bids strategy improves on this approach, being able to place multiple bids at the same hour and profit from the good price at least partially.

Learning rate ( <i>learning_rate</i> )	0.0001
Number of update steps ( <i>n_steps</i> )	900
Discount ( <i>gamma</i> )	0.9
GAE coefficient ( <i>gae_lambda</i> )	0.9
Entropy coefficient ( <i>ent_coef</i> )	0.0
Batch size ( <i>batch_size</i> )	128
Number of epochs ( <i>n_epochs</i> )	10
Clipping parameter ( <i>clip_range</i> )	0.2
Advantage normalization ( <i>normalize_advantage</i> )	True
Use gSDE ( <i>use_sde</i> )	False
Hidden layers neurons ( <i>net_arch</i> )	300
Log standard deviation init ( <i>log_std_init</i> )	-1
Normalize input ( <i>normalize_images</i> )	False
Activation function ( <i>activation_fn</i> )	tanh
Orthogonal initialization ( <i>ortho_init</i> )	True

Table 4: Parameters of the PPO algorithm used for experiments. Names in brackets are taken from the Stable-Baselines3 library [5]. Parameters not present in this table use default values.

Learning rate ( <i>learning_rate</i> )	0.0001
Discount ( <i>gamma</i> )	0.9
Buffer size ( <i>buffer_size</i> )	1 000 000
Learning start ( <i>learning_starts</i> )	9000
Batch size ( <i>batch_size</i> )	256
Polyak coefficient ( <i>tau</i> )	0.01
Training frequency ( <i>train_freq</i> )	1
Gradient steps ( <i>gradient_steps</i> )	1
Entropy coefficient ( <i>ent_coef</i> )	auto
Target entropy ( <i>target_entropy</i> )	auto
Use gSDE ( <i>use_sde</i> )	False
Hidden layers neurons ( <i>net_arch</i> )	300
Log standard deviation init ( <i>log_std_init</i> )	-1
Normalize input ( <i>normalize_images</i> )	False
Activation function ( <i>activation_fn</i> )	ReLU

Table 5: Parameters of the SAC algorithm used for experiments. Names in brackets are taken from the Stable-Baselines3 library [5]. Parameters not present in this table use default values.

Learning rate ( <i>learning_rate</i> )	0.0001
Discount ( <i>gamma</i> )	0.9
Buffer size ( <i>buffer_size</i> )	1 000 000
Learning start ( <i>learning_starts</i> )	900
Batch size ( <i>batch_size</i> )	256
Polyak coefficient ( <i>tau</i> )	0.001
Training frequency ( <i>train_freq</i> )	90
Gradient steps ( <i>gradient_steps</i> )	-1
Policy delay ( <i>policy_delay</i> )	2
Target policy noise ( <i>target_policy_noise</i> )	0.2
Target noise clip ( <i>target_noise_clip</i> )	0.5
Hidden layers neurons ( <i>net_arch</i> )	300
Normalize input ( <i>normalize_images</i> )	False
Activation function ( <i>activation_fn</i> )	ReLU

Table 6: Parameters of the TD3 algorithm used for experiments. Names in brackets are taken from the Stable-Baselines3 library [5]. Parameters not present in this table use default values.

A2C	<b>31036.12</b> $\pm$ 1310.28
PPO	24689.69 $\pm$ 1454.73
SAC	-17677.83 $\pm$ 4642.92
TD3	-116953.57 $\pm$ 28423.53

Table 7: Differences between achieved balances and the reference profit for all tested RL algorithms on the collection of bids strategy.

## I Regularization

In the reward function, we’ve introduced the regularization penalty to prevent the action coordinates from saturating at their bounds. If, at a certain learning stage, the agent learns that a certain coordinate of its action should be as small/large as possible, then the agent starts to produce its limit value with decreasing amount of exploratory noise. If, in another learning stage, this limit value is not optimal anymore, the agent is unable to find it out. This phenomenon is well known; different environments suffer from it, more or less. We have adopted regularization as a technique to cope with this phenomenon. Table 8 presents the results of tested strategies with and without this regularization penalty. The performance of the collection of bids strategy degrades slightly, with bigger uncertainty, while the pair of bids strategy achieves very similar results. The pair of bids strategy mostly produces sell bids with minimal prices and buy bids with maximal prices to ensure these bids will be realized. The minimal/maximal prices naturally correspond to saturated action coordinates. Therefore, the strategy does not suffer from such saturation. However, the collection of bids strategy actually approximates supply and demand curves, and saturation of some of their determinants degenerates the curves. This is especially harmful when one of the parameters  $a_{96} \dots a_{99}$  saturates since they are shared by each hour of the trading day.

Figure 5 presents the mean hourly relative battery charge from the best run of the collection of bids strategy without regularization, and Figure 6 presents the parameters

of created bids. We can clearly see that the collection of bids strategy without regularization is less stable, however still viable. Figures 7 and 8 present the same data for the pair of bids strategy without regularization.

COLLECTION_ALL	31036.12 $\pm$ 1310.28
COLLECTION_ALL_NOREG	29301.68 $\pm$ 2216.16
PAIR_ALL	30203.35 $\pm$ 644.51
PAIR_ALL_NOREG	30445.04 $\pm$ 1145.73

Table 8: Differences between achieved balances and the reference profit for the collection of bids strategy and the pair of bids strategy, with and without regularization, for the ALL scenario.

## References

1. Chen, S., Xu, Q., Zhang, L., Jin, Y., Li, W., Mo, L.: Model-based reinforcement learning for auto-bidding in display advertising. In: Autonomous Agents and Multiagent Systems (AAMAS) (2023)
2. Dong, Y., Dong, Z., Zhao, T., Ding, Z.: A strategic day-ahead bidding strategy and operation for battery energy storage system by reinforcement learning. *Electric Power Systems Research* **196**, 107229 (2021)
3. Liu, X., Shen, W.: Auto-bidding with budget and roi constrained buyers. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 2817–2825 (2023)
4. Mehta, A.: Auction design in an autobidding setting: Randomization improves efficiency beyond vcg. In: ACM Web Conference. pp. 173–181 (2022)
5. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research* **22**(268), 1–8 (2021)
6. Yang, M., Zhu, M., Liang, Q., Zheng, X., Wang, M.: Spotlight news driven quantitative trading based on trajectory optimization. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 4930–4939 (2023)

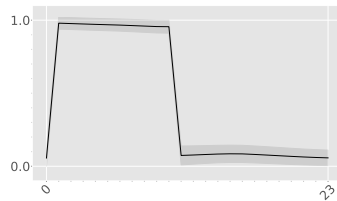


Fig. 1: Mean hourly relative battery charges in the simulation for the collection of bids strategy.

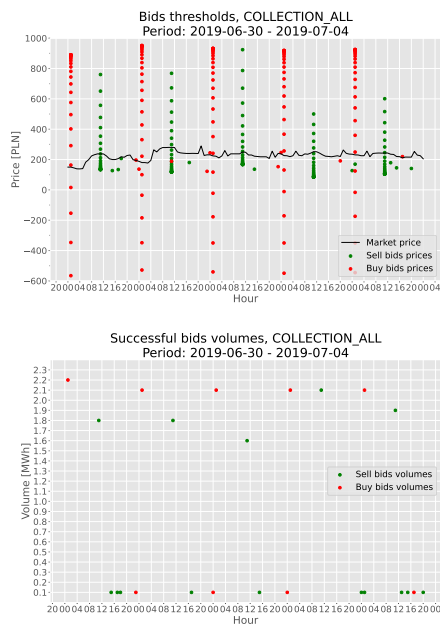


Fig. 2: The collection of bids strategy. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

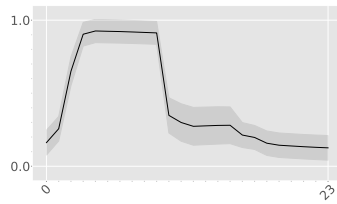


Fig. 3: Mean hourly relative battery charges in the simulation for the pair of bids strategy.

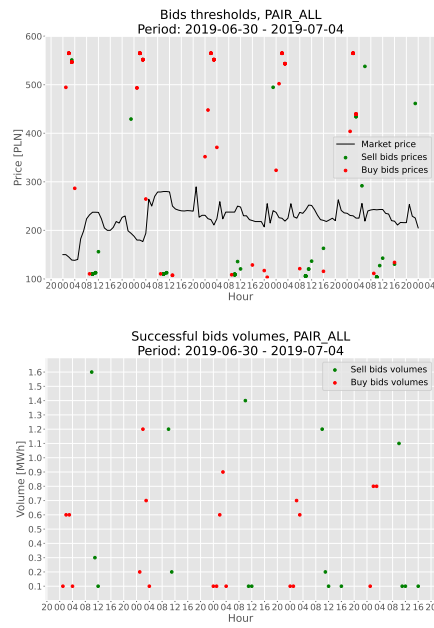


Fig. 4: The pair of bids strategy. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.



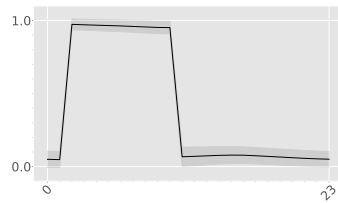


Fig. 5: Mean hourly relative battery charges in the simulation for the collection of bids strategy without regularization.

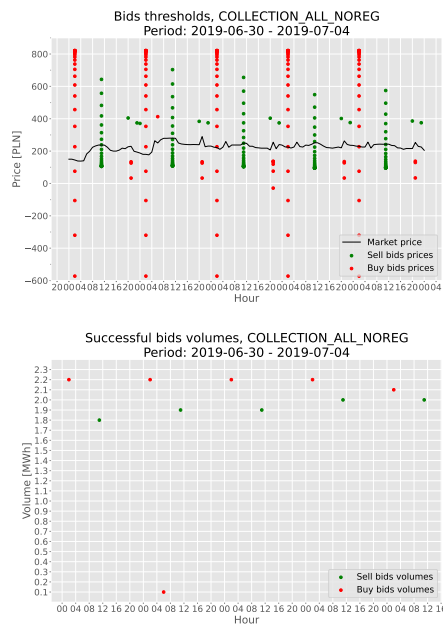


Fig. 6: The collection of bids strategy without regularization. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

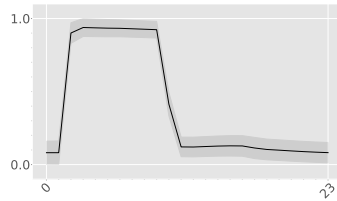


Fig. 7: Mean hourly relative battery charges in the simulation for the pair of bids strategy without regularization.

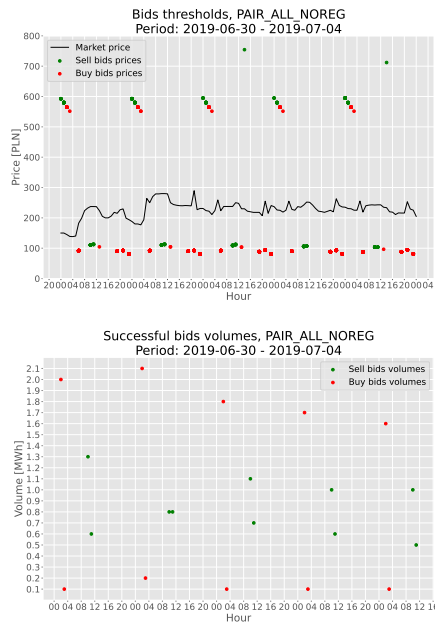


Fig. 8: The pair of bids strategy without regularization. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

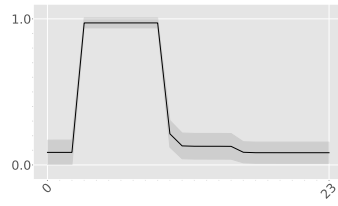


Fig. 9: Mean hourly relative battery charges in the simulation for the collection of bids strategy with battery, without production and consumption.

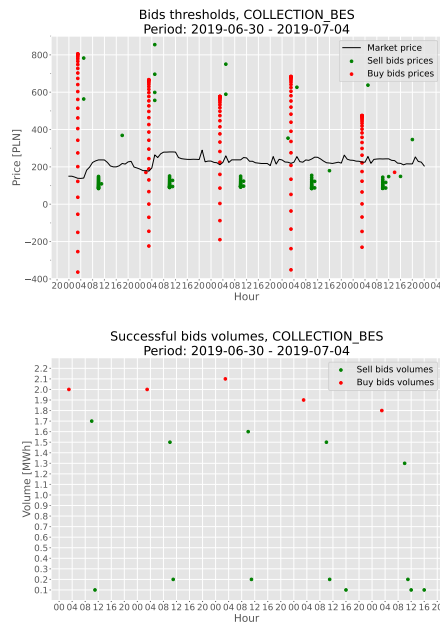


Fig. 10: The collection of bids strategy with battery, without production and consumption. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

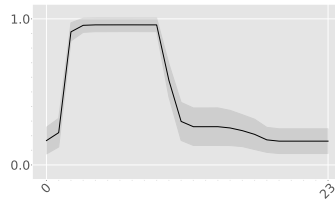


Fig. 11: Mean hourly relative battery charges in the simulation for the pair of bids strategy with battery, without production and consumption.

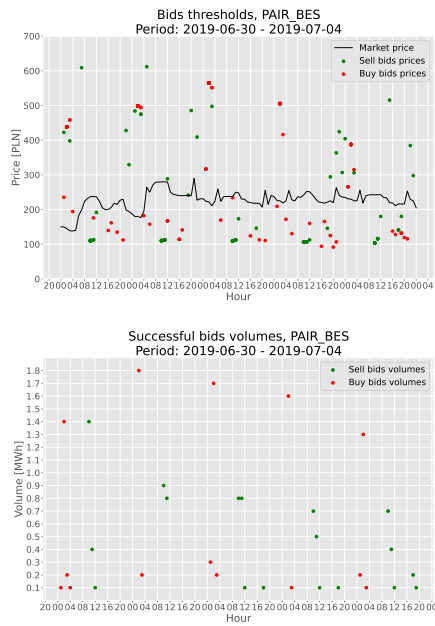


Fig. 12: The pair of bids strategy with battery, without production and consumption. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

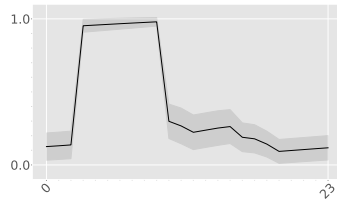


Fig. 13: Mean hourly relative battery charges in the simulation for the collection of bids strategy with battery and production, without consumption.

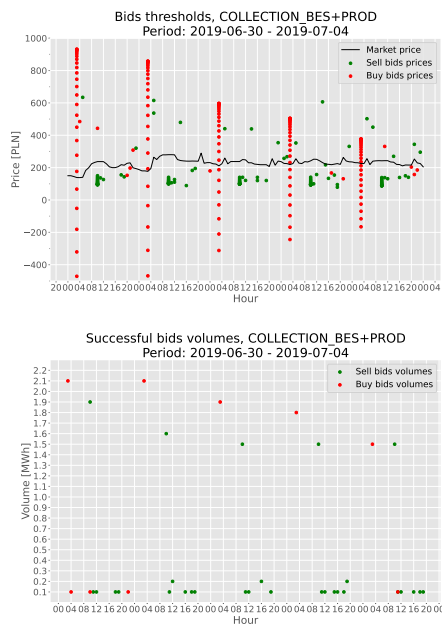


Fig. 14: The collection of bids strategy with battery and production, without consumption. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

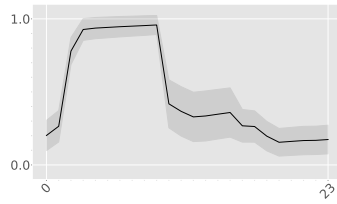


Fig. 15: Mean hourly relative battery charges in the simulation for the pair of bids strategy with battery and production, without consumption.

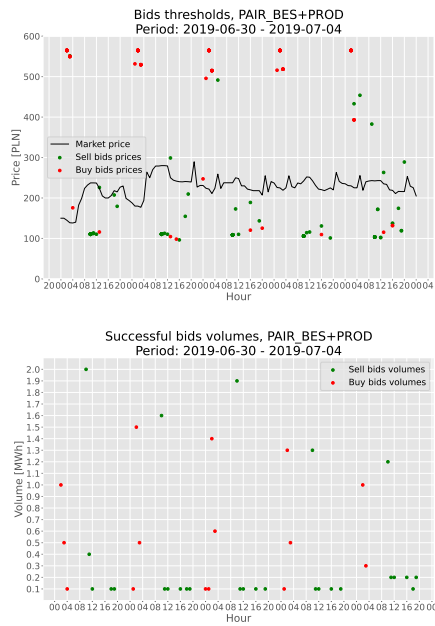


Fig. 16: The pair of bids strategy with battery and production, without consumption. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

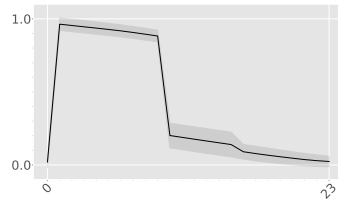


Fig. 17: Mean hourly relative battery charges in the simulation for the collection of bids strategy with battery and consumption, without production.

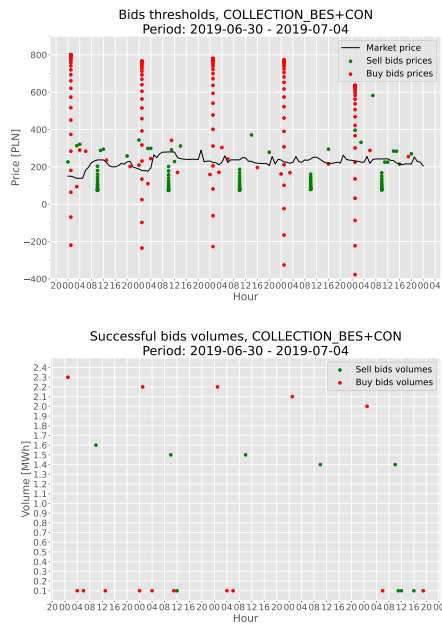


Fig. 18: The collection of bids strategy with battery and consumption, without production. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.

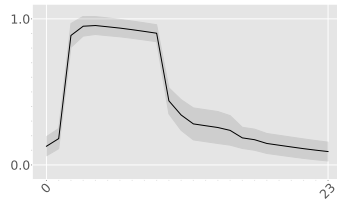


Fig. 19: Mean hourly relative battery charges in the simulation for the pair of bids strategy with battery and consumption, without production.

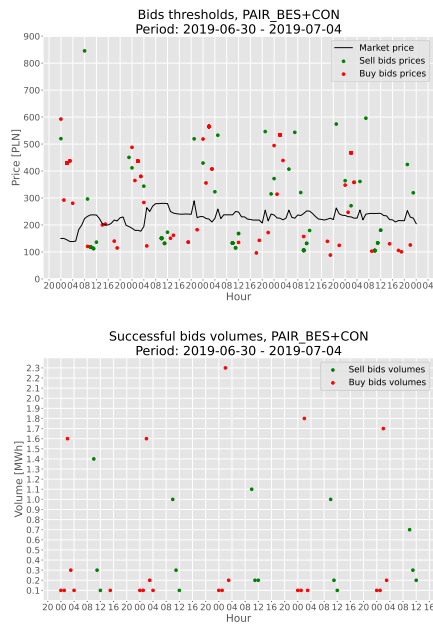


Fig. 20: The pair of bids strategy with battery and consumption, without production. Data was gathered from the test run with the best profit. *Up*: Price thresholds. *Down*: Volumes for successful bids.






## **B.2. Generalisation gap of keyword spotters in a cross-speaker low-resource scenario**

Title	Generalisation gap of keyword spotters in a cross-speaker low-resource scenario
Authors	Łukasz Lepak, Kacper Radzikowski, Robert Nowak, Karol J. Piczak
Journal	MDPI Sensors
Volume(Issue)	21(24)
Pages	8313
Year	2021
DOI	10.3390/s21248313
Ministerial score	100

Article

# Generalisation Gap of Keyword Spotters in a Cross-Speaker Low-Resource Scenario

Lukasz Lepak <sup>1,\*</sup> , Kacper Radzikowski <sup>1,2</sup>, Robert Nowak <sup>1</sup>  and Karol J. Piczak <sup>3,\*</sup> 

<sup>1</sup> Institute of Computer Science, Faculty of Electronics and Information Technology, Warsaw University of Technology, 00-665 Warsaw, Poland; kradziko@fuji.waseda.jp (K.R.); robert.nowak@pw.edu.pl (R.N.)

<sup>2</sup> Graduate School of Information, Production and Systems, Waseda University, Tokyo 808-0135, Japan

<sup>3</sup> Institute of Computer Science and Computational Mathematics, Jagiellonian University, 30-348 Krakow, Poland

\* Correspondence: lukasz.lepak.dokt@pw.edu.pl (L.L.); karol.piczak@uj.edu.pl (K.J.P.)

**Abstract:** Models for keyword spotting in continuous recordings can significantly improve the experience of navigating vast libraries of audio recordings. In this paper, we describe the development of such a keyword spotting system detecting regions of interest in Polish call centre conversations. Unfortunately, in spite of recent advancements in automatic speech recognition systems, human-level transcription accuracy reported on English benchmarks does not reflect the performance achievable in low-resource languages, such as Polish. Therefore, in this work, we shift our focus from complete speech-to-text conversion to acoustic similarity matching in the hope of reducing the demand for data annotation. As our primary approach, we evaluate Siamese and prototypical neural networks trained on several datasets of English and Polish recordings. While we obtain usable results in English, our models' performance remains unsatisfactory when applied to Polish speech, both after mono- and cross-lingual training. This performance gap shows that generalisation with limited training resources is a significant obstacle for actual deployments in low-resource languages. As a potential countermeasure, we implement a detector using audio embeddings generated with a generic pre-trained model provided by Google. It has a much more favourable profile when applied in a cross-lingual setup to detect Polish audio patterns. Nevertheless, despite these promising results, its performance on out-of-distribution data are still far from stellar. It would indicate that, in spite of the richness of internal representations created by more generic models, such speech embeddings are not entirely malleable to cross-language transfer.

**Keywords:** keyword spotting; speech embedding; siamese networks; automatic speech recognition



**Citation:** Lepak, L.; Radzikowski, K.; Nowak, R.; Piczak, K.J. Generalisation Gap of Keyword Spotters in a Cross-Speaker Low-Resource Scenario. *Sensors* **2021**, *21*, 8313. <https://doi.org/10.3390/s21248313>

Academic Editors: Leon Rothkrantz and Chimán Kwan

Received: 20 October 2021

Accepted: 10 December 2021

Published: 12 December 2021

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

### 1.1. Keyword Spotting

The goal of a keyword spotter is to detect words of interest in continuous audio recordings. These recordings can be provided either as prerecorded files of considerable length (offline processing) or real-time streaming data (online processing). Nowadays, keyword spotting is more often associated with the latter scenario. The purpose of such a real-time detector is to catch utterances of a specific wake word provided by the user and activate a fully functional conversation with a personal digital assistant.

However, offline usage of keyword spotting can be of great help when navigating vast libraries of audio recordings, pinpointing short regions of interest without the need to listen through the entire conversation. This feature is especially applicable to the mundane task of reviewing lengthy call centre recordings.

### 1.2. Paper Overview

In this paper, we describe a case study of developing such a proof-of-concept solution for spotting keywords in call centre recordings of a Polish bank. The system's goal was to provide rapid localisation of predefined words of interest when reviewing call centre conversations with the client, which would help bank employees process clients' complaints more efficiently.

### 1.3. Contributions

Our main contributions in this work can be summarised as follows:

- We create a number of experimental protocols (mono- and cross-lingual) for keyword spotting in continuous audio recordings.
- We prepare additional Polish datasets for training and evaluation purposes. The recordings sourced from YouTube clips are available on request.
- We evaluate similarity ranking models (Siamese and prototypical networks) in practical keyword detection tasks.
- We report the results on using the Google speech embedder on Polish data.
- We compare the embedding spaces generated by different combinations of the models and training data.
- We highlight the gap between popular keyword classification benchmarks and performance in more practical tasks (analysing authentic call centre recordings).

### 1.4. Approaches to Keyword Spotting

In general, there are two main approaches that we can utilise for creating keyword spotters: speech-to-text conversion or audio similarity matching. Both of them have their strengths and weaknesses.

#### 1.4.1. Speech-to-Text Conversion

The first approach consists of a full-scale automatic speech recognition (ASR) system transforming incoming audio streams into a textual representation. In this way, searching for regions of interest is converted to a trivial problem of finding words in a text stream. This approach has several advantages.

First of all, the generated output has a much more desirable form. Instead of only locating particular keywords, the system provides the user with a complete transcription of the reviewed conversation. Such an output form can then be utilised for numerous additional purposes.

A textual representation also helps when extending the dictionary of the system. As long as the speech recogniser generates a high-quality transcription, localising completely new keywords is trivial and can be performed ex-post.

Typical ASR systems have another advantage over audio content similarity matching, as they combine acoustic and language modelling information. Processing whole sentences enables the model to refine its predictions and eliminate implausible responses based on the probability estimate of the generated word sequence. Language modelling can thus help lower the word error rate (WER) in comparison to the processing of words in isolation.

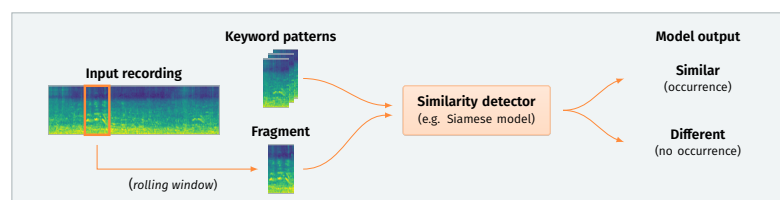
Unfortunately, the ASR-based approach has several drawbacks. For instance, any errors introduced during the transcription phase are final and cannot be fixed in post hoc analysis by adjusting the detection threshold. Another considerable drawback of training a complete ASR system is the data intensity of this process. Various estimates place the requirements on training data availability at 2000 h of transcribed recordings for a viable system [1] and as much as 10,000 h for production-quality results [2].

While the high cost of model training on such a scale [2] effectively prohibits any further experimentation with the baseline systems, it is not the most problematic factor. Unfortunately, in mid-2021, there were no publicly available resources of transcribed Polish speech that would remotely meet the required scope of such a project. Consequently, using

a complete ASR system is still not a viable option when processing Polish speech, despite the numerous advantages of this approach.

#### 1.4.2. Audio Similarity Matching

Due to the aforementioned problems with data availability, we have opted for the second option, i.e., developing a keyword detection system based on the perceptual similarity of acoustic fragments. Figure 1 presents a schematic depiction of this approach. The main principle of such a system lies in sequentially processing small fragments of an input recording. Each extracted fragment is then compared to a set of predefined exemplary patterns representing specific keywords. If a significant similarity is found, the system reports detection of a given keyword at a particular point in time.



**Figure 1.** Keyword detection based on audio similarity matching. This schematic depicts a typical pipeline. An input recording is processed sequentially with a short rolling window (e.g., 800 ms). The system compares each extracted fragment to a set of exemplary patterns predefined for each keyword. The model then outputs a decision based on the distance between the fragment and each pattern. If the distance is sufficiently small, a keyword occurrence is emitted for a given timestamp.

As our similarity detector, we employ a Siamese neural network model. It is a metric-learning model that has been widely used in similarity ranking problems and is specifically tailored to problems where examples of new classes are scarce, i.e., to few-shot learning [3]. We also compare Siamese models with prototypical networks [4], another solution popular in the domain of few-shot learning.

The main advantage of such similarity-based approaches lies in more moderate data requirements, both in terms of quantity and quality of annotation. Instead of using detailed transcripts that are costly to procure, we can train detectors by providing pairs of word utterances with a binary “match”/“no match” label, a much more convenient data acquisition setup.

Additionally, focusing on acoustic similarity allows for recognition of proper names that language modelling might filter out due to their rareness. The patterns might even represent data other than voice, as shown by Wang et al. [5]. By computing similarity in the domain much closer to the raw data representation, it would be also possible to search for fragments based on information usually lost during conversion, such as emotion, the rhythm of speech, and tone of voice. However, in this work, we focus solely on phonetic matching of keywords.

#### 1.5. Research Hypothesis

Our research hypothesis is that keyword spotting models that employ acoustic similarity matching should allow for keyword retrieval from call centre conversations in a low-resource language setting.

#### 1.6. Research Outcome

Our experiments show that, with a relatively diverse pool of examples used to train the acoustic embedding model, the performance of acoustic similarity matching is satisfactory when evaluated on synthetic benchmarks. Unfortunately, while such training resources are obtainable for the English language, we were unsuccessful in creating an equivalently

robust model for Polish speech. This performance gap underlines how problematic it is to create keyword spotting systems for low-resource languages.

However, as the pattern matching in such systems relies solely on acoustic similarity metrics, it should be, at least partially, language-agnostic—as evidenced by the results of Wang et al. [5]. Therefore, a plausible solution would be to train a model on a high-resource language and use it directly on the low-resource language, hoping that such perceptual matching will be sufficiently accurate, despite the language difference. To this end, we perform several experiments by initially training models on various standard English datasets and using Polish patterns in further detection steps. Unfortunately, such models exhibit weak generalisation when switching between languages, even when augmented with additional Polish data during training. This problem could indicate that the internal representation created by these models is highly specialised and attuned to the particular dataset.

As a way to potentially overcome this issue, we also evaluate an approach that utilises more generic speech embeddings created with a model provided by Google [6] that was pre-trained on 200 million English audio clips from YouTube. Our analysis shows that this embedder creates a more nuanced internal representation of the keyword classes, capturing additional variability factors present in the dataset. It is also better at detecting Polish patterns, despite being trained on non-Polish speech. Although the handicap of an enormous training dataset definitely helps in the few-shot cross-language setup, the results are still far from stellar. A final evaluation on out-of-distribution target recordings shows viability only for a very limited subset of potential Polish keywords.

## 1.7. Related Works

### 1.7.1. Keyword Spotting

Since the advent of deep learning models in speech processing, researchers have proposed numerous variants of such approaches for keyword spotting. Many of these models have been smaller-footprint versions of typical video and audio processing solutions. Unfortunately, most initial evaluations of models employing convolutional neural networks [7] and recurrent neural networks [8] used proprietary recordings for training and validation. More recently, by introducing the Speech Commands dataset [9], Warden established a popular public framework for benchmarking keyword spotting systems.

The most prevalent evaluation protocol for Speech Commands v1 employs an 80:10:10 split between training, validation, and testing data, selecting 10 of the 30 words in the dataset as target commands. The remaining 20 words create a single negative category (“unknown word”), while additional ambient recordings serve as a silent class, resulting in 12 classification options. Standard training data augmentation consists of random shifts and noise injection.

The baseline method provided by the author [9] attains a top-1 classification accuracy of 85.4% with a simple convolutional architecture [7] operating on 40-dimensional log-mel filter bank features. Slightly better performance is achievable with convolutional neural networks using 40-dimensional MFCC features [10]. By combining a broad hyperparameter sweep with depthwise separable convolutions, Zhang et al. were able to bring the accuracy up to 95% even with resource-constrained architectures [11].

In a similar vein, de Andrade et al. [12] applied a recurrent neural attention model to the task of speech command recognition achieving an accuracy of 94.1% on Speech Commands v1 with 80-band mel spectrograms. Zeng and Xiao [13] reported similar levels for baseline LSTM/CNN architectures, which they were able to improve up to 96.2% through the application of BiLSTM layers to a DenseNet model. What is even more noteworthy, these kinds of results are attainable even with low-footprint models suitable for speech recognition at the edge with less than 100,000 parameters (96.3% with EdgeSpeechNets [14], 96.1% reported for CNNs with temporal convolution over spectrograms [15], 96.4% with parametrized sinc-convolutions [16], and 97.5% with a bigger version of MatchboxNet [17]). When the number of parameters is not an issue, current state-of-the-art approaches can

achieve even higher accuracy levels with MoEx (*moment exchange*) feature regularisation on wide ResNet models (98.0% with WRN-28-10 consisting of 36.5 million parameters) [18].

Considering these stellar results, it might seem that keyword spotting is a solved problem. However, the accuracy of classifying isolated excerpts from the Speech Commands dataset might not translate to actual performance when working on continuous audio streams. Real scenarios involve unknown word boundaries, more variance in noisy conditions, and a highly imbalanced class distribution between trigger words and background information. This fact was brought up even in the original Speech Commands paper [9]. It has shown that a baseline model, with 88.2% accuracy of classifying isolated words from version 2 of the dataset, detects keywords with a 46.0% true positive rate when applied to a synthetically generated one-hour audio stream. While some papers were able to report much more promising results in this area of streaming keyword spotting, they were either limited to a detection of a single wake-word [19] or used proprietary datasets of unprecedented scope, e.g., 1 million training utterances of “OK/Hey Google” [20,21] and recordings of several hundred thousand subjects interacting with Alexa devices [22].

### 1.7.2. Speech Recognition in Low-Resource Settings

Over the years, many successful approaches to English ASR [2,23–29] have been proposed in the form of various deep learning models. Unfortunately, these models were trained on datasets containing hundreds or even thousands of hours of transcribed English speech. Simple adoption of these techniques to Polish ASR systems is thus impossible.

There were some previous attempts to create speech recognition systems specifically for the Polish language. However, they are mostly several years old and based on non-neural network methods. Some examples include systems based on hidden Markov models [30], k-nearest neighbours [31], and speech n-grams [32]. Results achieved with these methods are not satisfactory; therefore, they were not considered further in our project.

A recent development in global ASR research is the introduction of models trained on large amounts of unlabelled data. These methods swap the effort in data collection for computational time. By analysing vast collections of raw recordings, they can create more robust and powerful speech representations that can also facilitate knowledge transfer to low-resource languages.

An example of this approach is the wav2vec 2.0 framework introduced by Facebook [33]. It uses a contrastive self-supervised learning procedure to create latent speech representations. After pre-training on unlabelled data, the connectionist temporal classification (CTC) loss is used to refine the model further on downstream tasks with labelled data. The main advantage of this approach is the tremendous drop in the requirements for data annotation. Wav2vec 2.0 can achieve a WER of 2.9% on LibriSpeech *test-clean* with as little as one hour of labelled data. For such an outcome to occur, a high price has to be paid when training the model. The most performant variant consisting of 24 transformer blocks needs the equivalent of roughly two GPU years of training on a V100 to process the 53,200 h of LibriVox recordings.

An extension of this concept has led to the introduction of XLSR-53 [34], a large model pre-trained on 56,000 h of recordings in 53 languages. It has shown remarkable reductions in phone error rates when evaluated on various languages of the Common Voice dataset, with as little as one hour of labelled data required for fine-tuning the final model. Based on these results, XLSR-53 seems to be the most promising approach for speech recognition in low-resource languages to date. Due to its complexity, recreating such a model requires a considerable implementation effort and access to a GPU cluster. Unfortunately, Facebook released the pre-trained model only near the very end of 2020, after the conclusion of our project’s development. Nevertheless, for future endeavours in Polish ASR, it is an interesting possibility to explore.

However, in our work, we have opted for the help of a different pre-trained solution. In 2020, a team from Google Research released a speech embedding model trained on 200 million 2-s audio clips from YouTube [6]. This convolutional model aims to convert an

audio stream into a stream of 96-dimensional feature vectors. Each generated embedding vector encodes speech content in windows of approximately 800 ms, every 80 ms. To ensure the reusability of these embeddings, the model was pre-trained on an arbitrary set of 5000 keywords split randomly over 125 keyword spotters sharing the embedder backbone. After 40 GPU days of training, the embedding backbone has been released on TensorFlow Hub for reuse. The authors have shown that these generic embeddings could limit the amount of data needed for creating a robust Speech Commands keyword classifier. They were also successful in partially substituting real training data with synthesised speech. Therefore, we have chosen this approach as a potential solution for our data availability problem.

### 1.7.3. Voice Datasets

Various voice datasets suitable for speech recognition and keyword spotting tasks are used for training neural network models. While many datasets are strictly proprietary, a handful of them are distributed with more permissive licenses.

One of the most popular in this area is the already mentioned Speech Commands by Google [9]. Its second version consists of 35 different words in English, spoken by 2618 different speakers, totalling 105,829 utterances. The Speech Commands dataset is available for download from the TensorFlow Datasets catalogue.

Common Voice [35], an initiative of the Mozilla Foundation, is on the way to become the most significant publicly available dataset of voice recordings. It contains various sentences read by volunteers in many languages and is freely distributed under the Creative Commons Zero license. Everyone can contribute to the development of this dataset by recording their voice through a browser or a mobile phone. The quantity of sentences varies between languages, with English having the most significant share.

Initially, the dataset incorporated new languages when a sufficient number of recordings was amassed. Hence, the Polish version was publicly distributed for the first time only in Common Voice Corpus 5.1, in the later stages of our project's timeline. Nevertheless, the total length of validated recordings, reaching approximately 100 h, would be insufficient for training a complete ASR system either way. In mid-2021, more than 70 languages are publicly available. However, the distribution of recordings between languages remains very uneven.

Common Voice Single Word Target Segment was a spin-off dataset published alongside Corpus 5.1, resembling an extension of the Speech Commands concept across multiple languages. It contains recordings of spoken digits, as well as the words *yes*, *no*, *hey* and *Firefox*. The number of speakers varies greatly by language.

Another popular voice dataset is LibriSpeech [36]. It is a collection of around 1000 h of sentences in English. Recordings are annotated on a sentence rather than word level, making it more suitable for speech recognition tasks than keyword spotting.

The Spoken Wikipedia Corpus [37] is a collection of Wikipedia articles read by volunteers. It contains 182 h of word-aligned transcriptions in English, 249 h in German, and 79 h in Dutch. Based on this dataset, Wang et al. [5] have successfully employed prototypical networks for few-shot keyword detection, also in a cross-language evaluation. However, their problem formulation assumed that, in the detection phase, the keyword spotter is provided with exemplary patterns belonging to the same speaker as the speaker in the analysed recording. This availability of very similar patterns is in contrast to our project's assumptions. We expect keyword searching to function equally well in a cross-speaker regime, generalising to completely new speakers not encountered in the training set.

When deploying speech recognition models, some problematic aspects concerning voice datasets have to be considered. For instance, although the datasets described in this section use permissive licenses, many other datasets may come with limits on their research and commercial usage.

Another very significant problem is language availability. Numerous datasets are available in English. However, finding a good dataset in other languages is often not easy, as they tend to have much less data or insufficient quality. This deficiency makes speech

recognition and keyword spotting tasks outside English very difficult. Although several authors, apart from the Common Voice project [35], have prepared Polish voice corpora, they are either limited in recording length [38–40], not easily downloadable for offline use [41,42] or proprietary [43]. A potentially interesting dataset is SNUV (Spelling and NUmbers Voice database) [44] that contains 220 h of Polish speakers reading numbers and spelling words. However, due to the lack of complete word utterances (only spelling), its usability in training keyword spotters might be limited.

## 2. Materials and Methods

During the development of our keyword spotting system, we have conducted several experiments with various training and evaluation datasets, both in English and Polish. All the experiments in keyword detection adhere to the same pattern.

First, we prepare the datasets by preprocessing the recordings into mel spectrograms and dividing the data into three parts, described in Section 2.1:

- training data—single-word utterances used for training the acoustic similarity model,
- search patterns—keyword examples used as templates for comparison with fragments of the analysed evaluation recording,
- evaluation recordings—longer recordings used to assess how well the model detects keywords.

Following this step, we train the acoustic similarity model on single-word utterances. In our work, we consider two different types of models that we train from scratch: a Siamese convolutional neural network (*Siamese*) and a prototypical network (*Prototypical*). We also use a pre-trained speech embedder (*Google*) for comparison. Section 2.2 describes in more detail the architectures of these models and the training procedure.

After training, the similarity models work by calculating distances between a provided recording fragment and predefined keyword patterns. For each time step, this mode of operation provides distinct similarity values for each of the analysed keywords.

Therefore, as the last step, we use a detector to aggregate all this information into actual detection decisions based on the configured similarity threshold level. In this phase, we also introduce some temporal smoothing and filtering to make the predictions more robust. We provide more detailed detector configurations in Section 2.3.

### 2.1. Datasets and Data Preprocessing

The main goal of our system was to detect 22 pre-selected keywords in Polish call centre recordings provided by our industrial partner. These recordings came as an unlabelled collection of couple hundred conversations of differing lengths (from minutes to more than an hour) registered with standard call centre equipment at 8 kHz. Unfortunately, privacy concerns with this dataset resulted in stringent local access policies that proved to be problematic in combination with the ongoing pandemic restrictions. Therefore, we could effectively use this dataset only to evaluate the final system, with potentially significant domain shift.

In this less than ideal scenario, we had to resort to several other datasets, with different characteristics, for training and interim validation. Some of the datasets contain real audio recordings, while others are generated as synthetic mixtures. The original sampling rates of the datasets are varied, but we initially downsample all data to 8 kHz, which is the sample rate of the target recordings. We provide specific parameters of spectrogram preprocessing for each of the model types in Table 1.



**Table 1.** Parameters of mel spectrogram preprocessing for each model type.

	Siamese	Prototypical	Google
Sample rate	8 kHz	8 kHz	16 kHz
Segment duration	0.8 s	0.8 s	0.775 s
Mel bands	40	40	32
FFT window	512	512	400
Hop length	160	160	160
Centring	false	false	false

It is worth noting that the Google model has its processing sample rate set to 16 kHz. This setting aims to match the original training configuration [6]. However, as all our source recordings are initially downsampled to 8 kHz, the available information is the same for all models. Moreover, this discrepancy is also negligible as, effectively, the pre-trained speech embedder only uses log-mel features up to 3.8 kHz [6].

### 2.1.1. Training Data

This section provides short descriptions of the training datasets. We use training folds for creating the similarity model and validation data for interim monitoring. Test folds either serve as the basis for evaluation mixtures or are discarded. We initially downsample all recordings to 8 kHz. Where needed, we also refactor the datasets for the purpose of prototypical training by further splitting the data into *query* and *support* subsets by selecting speakers with a 75:25 ratio. Table 2 presents the aggregated statistics for the datasets.

**Table 2.** Statistics of the training datasets.

Dataset	Language	Samples	Classes	Speakers	Acronym
Speech Commands v1	English	64,727	30	1881	SC <sub>1EN</sub>
Speech Commands v2	English	105,829	35	2618	SC <sub>2EN</sub>
Speech Commands “delta”	English	7967	5	663	$\Delta$ SC <sub>EN</sub>
Common Voice Single Word	English	26,070	14	3519	CV <sub>EN</sub>
Common Voice Single Word	Polish	898	14	86	CV <sub>PL</sub>
Spoken Wikipedia Corpus	English	429,354	4657	372	SWC <sub>EN</sub>
Warsaw University of Technology	Polish	1058	36	29	WUT <sub>PL</sub>
Text-to-Speech	Polish	39,809	5687	7	TTS <sub>PL</sub>

#### Speech Commands (SC<sub>1EN</sub>, SC<sub>2EN</sub>, $\Delta$ SC<sub>EN</sub>)

Speech Commands datasets [9] are defined as follows: SC 1 contains words from 30 classes, SC 2 is a superset of SC 1 extended to 35 classes, while  $\Delta$ SC contains words from the five classes that are present exclusively in SC 2. All recordings contain an utterance of a single word. We follow the original structure of the dataset by using the pre-defined data splits.

#### Common Voice Single Word (CV<sub>EN</sub>, CV<sub>PL</sub>)

Common Voice Single Word datasets [35] are available both in English and Polish. Each recording contains a single word. We prepare the dataset by splitting it into *train*, *validation*, and *test* folds with a 80:10:10 ratio.

#### Spoken Wikipedia Corpus (SWC<sub>EN</sub>)

Spoken Wikipedia Corpus [37] contains English Wikipedia articles read by volunteers. Based on the provided annotations, we extract single word fragments that fit our constraints: they have a sample duration between 200 and 1000 ms, a keyword length of at least three characters, at least five unique speakers for a given keyword, and at least ten samples in total. We limit the number of samples per single keyword/speaker combination to five occurrences. We then split the created collection into 80:20 training and validation folds.

#### Warsaw University of Technology ( $WUT_{PL}$ )

The Warsaw University of Technology dataset is a small collection of single word utterances based on our list of target Polish keywords extended with 14 classes with high acoustic similarity. The word examples are recorded with personal devices by employees of the Artificial Intelligence Division at the Warsaw University of Technology and the personnel of mBank SA. The dataset is used exclusively for training purposes.

#### Text-to-Speech ( $TTS_{PL}$ )

The Text-to-Speech keywords dataset contains Polish spoken words generated with Azure and Google text-to-speech services. We use two forms of this dataset. For pattern matching, we limit the keywords to the 22 classes of the target evaluation. For training purposes, we extend the vocabulary with additional words found while scraping the bank's website. This process results in a total of 5687 classes generated with seven TTS voices.

#### Combined Datasets ( $ALL_{EN}$ , $ALL_{PL}$ )

Combined datasets, as the name suggests, are composed from the datasets described earlier. For the English language, we merge Speech Commands v2 ( $SC_{2EN}$ ), English Common Voice Single Word ( $CV_{EN}$ ) and Spoken Wikipedia Corpus ( $SWC_{EN}$ ). For the Polish language, the combined dataset consists of the Polish Common Voice Single Word ( $CV_{PL}$ ), Warsaw University of Technology ( $WUT_{PL}$ ) and Text-to-Speech recordings ( $TTS_{PL}$ ).

#### 2.1.2. Search Patterns

Herein, we briefly describe the collections of recordings that we use in our experiments as keyword templates in the matching process. For each keyword, we use 10 recordings as templates.

#### Speech Commands ( $SC_{1EN}$ , $\Delta SC_{EN}$ )

Speech Commands search patterns are generated directly from the respective  $SC_{1EN}$  and  $\Delta SC_{EN}$  datasets. For each keyword, we select ten random examples from the datasets' merged training and validation folds. The datasets consist of 30 and 5 classes, respectively.

#### Target Keywords ( $KW_{PL}$ )

We create these search patterns based on the Warsaw University of Technology and Text-to-Speech recordings. The dataset contains ten examples for each of the 22 target keywords, although only 19 keywords are actually present in the evaluation recordings.

#### 2.1.3. Evaluation Recordings

This section describes the datasets we use for evaluation purposes.

#### Synthetic Mixtures ( $SC_{1EN,mix}$ , $\Delta SC_{EN,mix}$ , $KW_{PL,mix}$ )

We create synthetic evaluation recordings by combining the utterances from the Speech Commands datasets ( $SC_{1EN}$  and  $\Delta SC_{EN}$ ) as a single continuous audio stream. The utterances come from the respective test folds. In each case, we place 20 keyword samples per recording with random delays between each occurrence. We generate 50 random evaluation mixtures in this way. For the Polish language, we generate these mixtures based on 1136 hand-annotated keyword examples from YouTube audio clips. All these synthetic mixtures contain only keywords detected in a given scenario, and are merged into a continuous audio stream by us.

#### Speech Commands Overlay on VoxCeleb (Semi-Synthetic) ( $SC_{1EN,Vox}$ )

This evaluation dataset consists of Speech Commands keywords overlaid on various backgrounds in the form of VoxCeleb [45] conversations. For each evaluation recording, we use a single keyword occurrence and two conversation fragments with a total length of approximately 15 s. We generate 20 recordings for each keyword.

### Target Keywords in YouTube Recordings (Authentic) ( $KW_{PL,real}$ )

As our most realistic evaluation protocol, we use authentic fragments of continuous speech extracted from various YouTube audio clips. Each fragment contains at least one target keyword and lasts from a couple of seconds to more than a minute. We annotated these fragments by hand. The recordings are separate from the synthetic mixtures ( $KW_{PL,mix}$ ). These recordings are real and contain numerous background words which are not the keywords we wish to detect. In total, we use 344 recordings with 511 keyword occurrences.

### Call Centre Target Keywords ( $CC_{PL,real}$ )

This dataset was provided by our industrial partner. It consists of several hundred recordings from the bank's call centre, with conversation lengths varying from seconds to hours, sparsely distributed occurrences of keywords and noise typical for call centre recordings. The recordings are provided in a typical call centre quality, an 8 kHz sample rate, and are unlabelled. During the project's time frame, the keyword labelling was performed for about 20% of the provided recordings. The availability of the dataset was strictly limited due to stringent privacy regulations, so we only tested our final approaches on it, as it required much coordination and effort to launch it on the bank's infrastructure.

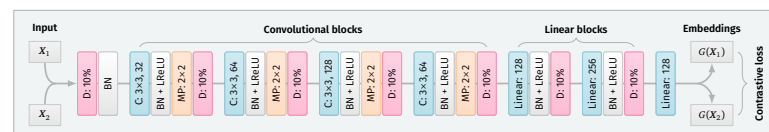
## 2.2. Similarity Ranking Models

In this part, we briefly provide the specifications for our model and training procedures.

### 2.2.1. Siamese Convolutional Neural Network

As our primary similarity ranking model, we use a Siamese neural network. The goal of this model is to map pairs of examples into pairs of embedding vectors. During optimisation, embeddings of examples belonging to the same class are placed close together in the embedding space.

The network consists of two identical convolutional branches serving as a speech embedder. In practice, we employ only one instantiation of the convolutional embedder as both branches are identical clones, with shared parameters. The convolutional embedder uses a VGG-like architecture, depicted in Figure 2, with four convolutional blocks, two dense layers, and a linear embedding. Each block combines  $3 \times 3$  padded convolutions with batch normalisation, Leaky ReLU activations, max pooling, and 10% dropout. We use 32–64–128–64 filters, accordingly. After each dense layer (128 and 256 neurons), we add batch normalisation, Leaky ReLU, and 10% dropout. The final embedding has an output size of 128 features. We normalise the input recordings with 10% dropout and single-channel batch normalisation.



**Figure 2.** The architecture of the embedding model. Layer types denoted as: D—dropout, BN—batch normalisation, C—convolutional, MP—max-pooling, LReLU—LeakyReLU. Both the Siamese and prototypical models use the same embedder architecture, albeit with different loss functions and input formulation.

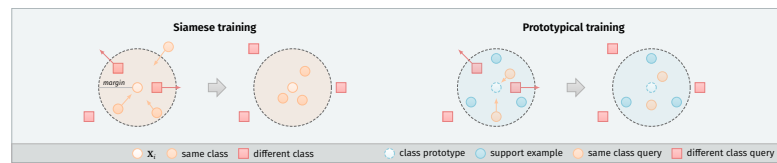
The Siamese model training is based on the contrastive loss, i.e., for pairs of training examples  $\langle X_i, X_j \rangle$  created with a given selection strategy  $((i, j) \in \mathbb{P})$ , we calculate the loss value according to the following formula:

$$\mathcal{L}(\mathbf{W}, \mathbf{X}, \mathbf{Y}) = \frac{1}{|\mathbb{P}|} \sum_{(i,j) \in \mathbb{P}} \left( (1 - Y_{(i,j)}) \cdot D_{\mathbf{W}}(X_i, X_j) + Y_{(i,j)} \cdot \max(0, m - \sqrt{D_{\mathbf{W}}(X_i, X_j)})^2 \right) \quad (1)$$

where  $\mathbf{W}$  specifies the weights of the embedder,  $Y_{(i,j)}$  term defines if the examples  $(\mathbf{X}_i, \mathbf{X}_j)$  are similar ( $Y_{(i,j)} = 0$ ) or dissimilar ( $Y_{(i,j)} = 1$ ),  $D_{\mathbf{W}}$  is the similarity (distance) function defined for the embeddings  $\langle G_{\mathbf{W}}(\mathbf{X}_i), G_{\mathbf{W}}(\mathbf{X}_j) \rangle$ , and  $m$  is the margin value. In our case, we use squared Euclidean distances, i.e.,:

$$D_{\mathbf{W}}(\mathbf{X}_i, \mathbf{X}_j) = \|G_{\mathbf{W}}(\mathbf{X}_i) - G_{\mathbf{W}}(\mathbf{X}_j)\|_2^2 \quad (2)$$

This loss formulation keeps the embeddings of the same class closer together. On the other hand, the embeddings of different keyword classes are pushed away, so that they do not fall inside the margin, as illustrated in Figure 3.



**Figure 3.** Overview of the Siamese and prototypical training approaches. Siamese networks use a contrastive loss to make pairwise comparisons between examples from the dataset  $\mathbf{X}$ . Examples from the same class are attracted to each other, while the examples from different classes are pushed back if their distance is lower than the defined margin  $m$ . The prototypical approach uses additional support examples  $\mathbf{S}$ . Their mean vector defines the class prototype. Query examples  $\mathbf{Q}$  of the same class are attracted to this prototype, while queries of different classes are pushed back in the same way as with Siamese training.

We train the model for 100 epochs, using the Adam optimiser with default hyper-parameters and a learning rate of 0.001, with a contrastive loss margin of 1.0. We use 50,000 training examples per epoch. In each epoch, we generate 200 training episodes, consisting of 25 classes per episode and 10 samples per class. Our pair selection procedure uses the *hard negative* variant, i.e., we first create all possible positive pairs (matching keywords) from the samples in the current episode, and then we generate an equal number of negative pairs selecting examples with the smallest distances.

### 2.2.2. Prototypical Network

In prototypical network experiments, we use the same embedder architecture as in Section 2.2.1. The main difference between the Siamese and prototypical approaches lies in the introduction of class prototypes. In Siamese models, we calculate the distances for similar/dissimilar pairs of individual examples from the dataset, whereas prototypical networks divide the training data into *support* ( $\mathbf{S}$ ) and *query* ( $\mathbf{Q}$ ) subsets. In each episode of training, a class prototype is created for each of the  $C$  selected classes by computing the mean value of the  $K$  vectors from the support subset:

$$\bar{\mathbf{S}}_c = \frac{1}{K} \sum_{i=1}^K \mathbf{S}_{c_i}, \quad (3)$$

where  $\mathbf{S}_c$  denotes support examples of class  $c$ . The pair comparison is then performed between such class prototypes and all query examples selected in the given episode, resulting in a loss function:

$$\mathcal{L}(\mathbf{W}, \mathbf{S}, \mathbf{Q}, \mathbf{Y}, C) = \frac{1}{C|\mathbf{Q}|} \sum_{c=1}^C \sum_{i=1}^{|\mathbf{Q}|} \left( (1 - Y_{c_i}) \cdot D_{\mathbf{W}}(\bar{\mathbf{S}}_c, \mathbf{Q}_i) + Y_{c_i} \cdot \max\left(0, m - \sqrt{D_{\mathbf{W}}(\bar{\mathbf{S}}_c, \mathbf{Q}_i)}\right) \right)^2, \quad (4)$$

where  $Y_{c_i}$  defines if the query example  $\mathbf{Q}_i$  is of the same class  $c$  as the prototype  $\bar{\mathbf{S}}_c$  ( $Y_{c_i} = 0$ ) or different ( $Y_{c_i} = 1$ ). A more intuitive depiction of this procedure is presented in Figure 3.

Using this loss formulation, we train the prototypical model for 200 epochs comprising 100 episodes each, by optimising the prototypical loss with squared Euclidean distances and a margin of 1.0. In each prototypical training episode, we take 5 support samples and 15 query samples per class. Each episode consists of 25 classes. This setup results in the same total number of parameter updates for the Siamese and prototypical approaches.

### 2.2.3. Google Speech Embedder

In the experiments involving the Google speech embedder, we use the pre-trained model provided as version 1 on the TensorFlow Hub ([https://tfhub.dev/google/speech\\_embedding/1](https://tfhub.dev/google/speech_embedding/1), accessed on 10 December 2021). When fine-tuning the network, we use the same training approach as in Section 2.2.1, but with 50 epochs and 10 classes per episode.

## 2.3. Detector Settings

Using the similarity models described in Section 2.2, we generate distances between consecutive fragments of the analysed recording and each of the provided keyword templates (we use 10 templates per keyword class). This way, we obtain multiple values for each single time step, telling us how closely the current fragment resembles these various patterns. To aggregate such data into more meaningful detection decisions, we use various detection policies that we briefly describe in this section.

### 2.3.1. Common Detection Pipeline

In most of the experiments, we use the same detection pipeline. We analyse each frame of the recording, i.e., we employ a step size of 1. The actual frame size in milliseconds is determined by the spectrogram processing settings defined in Table 1. We then transform the generated distances with a median filter with a window length of 5 frames, obtaining smoothed similarity scores for each search pattern.

Based on these scores, we find the lowest average across all the analysed classes. If the average score falls below the threshold value defined for a given policy, we emit a detection marker for a given keyword at this particular time step. These marker emissions are then smoothed with a median filter applied across 15 frames.

Finally, if any keyword generates a continuous sequence of markers exceeding our minimum length of 25 consecutive frames, we return a detection at a given time step. We also introduce a minimal distance of 10 frames between two consecutive occurrences of the same keyword.

When evaluating the results of the system, we count the detection as a true positive if it falls at a time step representing the middle of the keyword occurrence, with a collar of 1 s. We adjust the policy threshold values based on the specifics of each model. For instance, for the Siamese model, we evaluate all values from 0 to 1 with a step size of 0.05, as this range allows us to generate a complete precision–recall curve.

### 2.3.2. Additional Post-Processing for the Google Speech Embedder

In standard experiments with the pre-trained embedder model, we use the same approach as described in Section 2.3.1, albeit with a minimum length of consecutive matches reduced to 10 frames. However, our experiments show that the similarity scores returned by direct distance calculations on speech embeddings generated by the Google model have a different scale and are less uniformly distributed across the different keyword classes.

Therefore, we introduce an additional post-processing step to our detection pipeline in the form of an *exciter* module. The role of this module is to make the similarity values more uniform across different keywords and filter out superfluous detections. To achieve this goal, we perform several operations.

First, we standardise the values of the distances in the detection matrix, visualised as input data in Figure 4. We perform this standardisation per each of the 10 patterns (rows). Then, we filter out all the values above the 5th percentile, leaving only the responses for the

time steps with the closest matches. This way, we obtain several time series representing filtered and standardised similarity scores for single patterns.



**Figure 4.** Post-processing of the similarity scores generated by the pre-trained embedder. The first matrix, denoted as *input*, shows an example of raw distances between the embeddings of keyword patterns and recording fragments. Each row corresponds to a single pattern, while each column represents a single time step. Brighter colours indicate smaller distances (closer matches). The second matrix shows the same values after row-wise standardisation and filtering through the *exciter* module.

After that, we use an envelope follower to perform excitation filtering on each of the time series. Therefore, after encountering a distance value below our threshold (signifying a potentially close match for a given template), we sustain this information in time with a decaying impulse response for up to 50 steps. When we find another close match, the decaying response is restarted. The result of this process is depicted in Figure 4.

#### 2.4. Statistical Analysis of the Results

After running the experiments, we conduct a statistical analysis of the obtained results. For every experiment, we report training, patterns and evaluation datasets that were used. As the performance metrics, we use the Area Under Precision-Recall Curve (AUPRC) and F-score. The AUPRC is calculated based on the precision and recall values for different detection thresholds. Precision tells us how many of the reported keyword detections are correct, while recall shows us how many of the actual keyword utterances are retrieved. The AUPRC is provided with both micro- and macro-averaging. Micro-averaging takes into account the sizes of every keyword class we wish to detect by averaging over all the examples as a whole. This approach might be better suited to our problem, as we mainly deal with imbalanced datasets. On the other hand, macro-averaging, which aggregates the AUPRC values by first calculating them separately for each class, may also be helpful when interpreting the results. The F-score is defined as  $F = \frac{\text{precision} + \text{recall}}{2}$ , and it combines the precision and recall results into one metric, with higher values considered better.

### 3. Results

#### 3.1. Isolated Words Classification Benchmark

Before proceeding with the actual keyword detection experiments, we performed a baseline verification to make sure that our embedder architecture (*Siamese*), described in Section 2.2.1, is sufficiently performant when processing audio data. To this end, we recreate the Speech Commands evaluation protocol [9] of classifying single-word utterances as one of the 12 possible classes. This is the only experiment where we do not downsample the input recordings. Instead, we maintain the original 16 kHz sampling rate, adjusting the spectrogram preprocessing settings accordingly.

Our convolutional model achieves a 94.5% top-1 classification accuracy on the Speech Commands v1 dataset. This performance is on par with similar models processing mel spectrograms, as presented in Table 3. While more sophisticated models can achieve better classification results, we deemed the differences not significant enough to warrant the trade-off these models introduce in terms of complexity and training time.

**Table 3.** Top-1 classification accuracy for isolated words of Speech Commands v1.

Authors	Method	Accuracy
Warden [9]	Baseline CNN model	85.4%
Tang et al. [10]	CNN with MFCC features	90.2%
de Andrade et al. [12]	CNN with 80-band mel spectrograms	94.1%
Zhang et al. [11]	Depthwise separable CNN	95.4%
Choi et al. [15]	CNN with temporal convolution	96.1%
Zeng et al. [13]	DenseNet with BiLSTM layers	96.2%
Lin et al. [14]	EdgeSpeechNets	96.3%
Mittermaier et al. [16]	Parametrized sinc-convolutions	96.4%
Majumdar et al. [17]	MatchboxNet	97.5%
Li et al. [18]	Wide-ResNet with MoEx	98.0%
Ours (Siamese)	CNN with dropout, 40-band mel spectrograms	94.5%
Ours (Siamese), 8 kHz	Same as above, downsampled recordings	92.0%

We also verify the same architecture on a downsampled version of the Speech Commands recordings. As expected, this impairs the accuracy of the evaluated model. However, the results show that 8 kHz recordings can still provide sufficient information for proper classification.

### 3.2. Keyword Detection in a Monolingual Setup

As a first step in evaluating keyword detectors on continuous audio data streams, we analyse our Siamese approach in monolingual scenarios, i.e., by training and validating the model on recordings of the same language. For selected experiments, we also compare its performance to a prototypical network.

In each evaluation setting, we create a separate *precision-recall curve* (PRC) for each of the analysed keywords. The PRC shows detection performance at different values of the similarity threshold. We aggregate this information across classes using either micro- (*instance*) or macro- (*class*) averaging, a common approach in multi-class problems [46].

To summarise the performance of a model with a single value, we report the *Area Under Precision-Recall Curve* (AUPRC). We also highlight the best F-score achieved by the model across different threshold values. Table 4 presents the results obtained in the analysed monolingual scenarios.

Our initial verification, described in Section 3.1, has shown that a Siamese convolutional neural network can effectively differentiate between utterances of different keywords of the Speech Commands dataset. We also confirm this capability in a streaming evaluation by employing the Siamese model with standard detection settings on a synthetic mixture of Speech Commands keywords (SC 1<sub>EN,mix</sub>). When the vocabulary present in the target recording consists solely of the expected keywords, the model can achieve an outstanding performance of 91.3% micro-AUPRC and an F-score of 0.94. This result shows that our detection approach allows for properly recognising keywords with shifted word boundaries.

To measure the robustness of the detector to background distractors, we evaluate it on Speech Commands keywords mixed into fragments of conversations of the VoxCeleb dataset (SC 1<sub>EN,Vox</sub>). In this case, the detection accuracy drops to a level of 60.4% micro-AUPRC. Nevertheless, the performance is satisfactory for a cross-speaker keyword spotter as the system can correctly highlight more than 2/3 of keyword occurrences while maintaining a precision of 55%. Results at this level would be usable for prospective users of such a tool. Unfortunately, we have to admit that the semi-synthetic nature of this evaluation oversimplifies the task presented to the detector, making it an upper bound on achievable accuracy. Due to the lack of adequate data, we could not verify how such detectors would cope with more natural keyword occurrences and diverse recording conditions.

**Table 4.** Keyword detection performance for models trained and evaluated in monolingual scenarios. The Area Under Precision–Recall Curve is based on the keyword classes present in the search patterns. We report it either with micro- or macro-averaging. The F-score column represents the best result achieved by the model across different settings of the detection threshold. All models use the default detector configuration.

Model	Training <sup>1</sup> (# KW)	Patterns <sup>2</sup>	Evaluation <sup>3</sup> (# KW)	AUPRC		F-Score
				Micro	Macro	
<b>English → English</b>						
Siamese	SC 1 <sub>EN</sub> (30)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> (30)	91.3%	91.3%	0.94
Siamese	SC 2 <sub>EN</sub> , CV <sub>EN</sub> (37)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> (30)	87.9%	88.0%	0.92
Siamese	SC 1 <sub>EN</sub> , SWC <sub>EN</sub> (4K+)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> (30)	69.4%	65.6%	0.79
Siamese	ALL <sub>EN</sub> (4K+)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> (30)	70.8%	67.4%	0.81
Prototypical	SC 1 <sub>EN</sub> (30)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> (30)	80.2%	81.5%	0.88
Siamese	SC 1 <sub>EN</sub> (30)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> (30)	60.4%	65.0%	0.62
Siamese	SC 2 <sub>EN</sub> , CV <sub>EN</sub> (37)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> (30)	55.2%	62.4%	0.57
Siamese	SC 1 <sub>EN</sub> , SWC <sub>EN</sub> (4K+)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> (30)	38.0%	43.4%	0.47
Siamese	ALL <sub>EN</sub> (4K+)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> (30)	43.7%	46.9%	0.50
Prototypical	SC 1 <sub>EN</sub> (30)	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> (30)	40.8%	50.2%	0.54
Siamese	SC 1 <sub>EN</sub> (30)	ΔSC <sub>EN</sub>	ΔSC <sub>EN,mix</sub> (5)	48.3%	53.1%	0.65
Prototypical	SC 1 <sub>EN</sub> (30)	ΔSC <sub>EN</sub>	ΔSC <sub>EN,mix</sub> (5)	39.6%	42.3%	0.59
<b>Polish → Polish</b>						
Siamese	ALL <sub>PL</sub> (5K+)	KW <sub>PL</sub>	KW <sub>PL,mix</sub> (22)	8.8%	1.3%	0.20
Prototypical	ALL <sub>PL</sub> (5K+)	KW <sub>PL</sub>	KW <sub>PL,mix</sub> (22)	5.9%	1.9%	0.22
Siamese	ALL <sub>PL</sub> (5K+)	KW <sub>PL</sub>	KW <sub>PL,real</sub> (22)	0.0%	0.0%	0.02
Prototypical	ALL <sub>PL</sub> (5K+)	KW <sub>PL</sub>	KW <sub>PL,real</sub> (22)	0.0%	0.0%	0.01

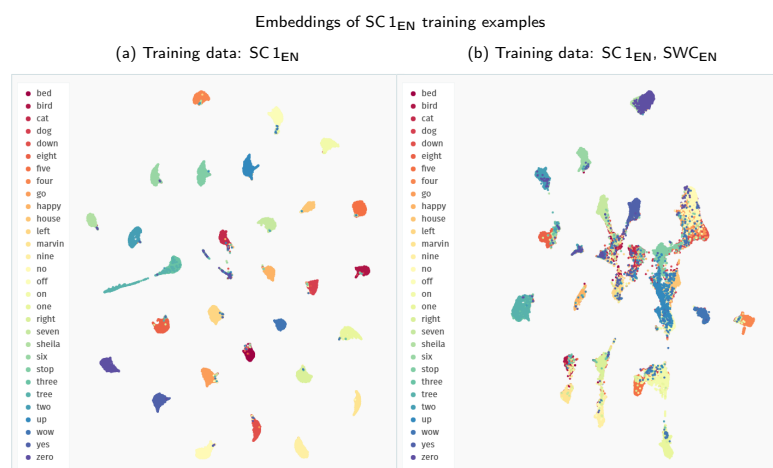
1: Training data consists of utterances from Speech Commands (SC 1<sub>EN</sub>, SC 2<sub>EN</sub>), Common Voice (CV<sub>EN</sub>), and Spoken Wikipedia Corpus (SWC<sub>EN</sub>). We also use a combined dataset (ALL<sub>EN</sub>). For the Polish language, we use all accessible training data (ALL<sub>PL</sub>), i.e., CV<sub>PL</sub>, WUT<sub>PL</sub> and TTS<sub>PL</sub>. Number of keyword classes (# KW) is denoted with a subscript. 2: English search patterns are extracted from Speech Commands training data (SC 1<sub>EN</sub>) or from the subset present only in the second version of the dataset (ΔSC<sub>EN</sub>). Polish templates come from the target keywords dataset (KW<sub>PL</sub>). We use 10 examples for each keyword class. 3: English models are evaluated with 30 keyword classes, either on fully synthetic mixtures of Speech Commands utterances (SC 1<sub>EN,mix</sub>) or Speech Commands keywords overlaid on VoxCeleb recordings (SC 1<sub>EN,Vox</sub>). We also show the performance on the “delta” dataset mixtures (ΔSC<sub>EN,mix</sub>, i.e., 5 classes). Polish evaluations assess 22 keywords either in synthetic mixtures (KW<sub>PL,mix</sub>) or actual YouTube audio streams (KW<sub>PL,real</sub>).

On the other hand, our problem setting assumes that the detector can only be trained on copious amounts of generic keywords. After that, it should cope well with limited examples of target keywords, especially since the end-user can extend the vocabulary after the system’s deployment. We assess this aspect with an experiment using keywords not occurring in the original training data, isolated from the second version of the Speech Commands dataset (ΔSC<sub>EN,mix</sub>). This evaluation scenario confirms that few-shot learning is quite difficult. A drop of the F-score to 0.65 on synthetic mixtures indicates that, when combined with more natural evaluation settings, detection of completely new keywords might be problematic. A more reasonable approach would require at least some retraining with the extended vocabulary.

Interestingly, across all the experiments, the extension of the Speech Commands training data with other datasets proves detrimental to the model’s performance. We analyse this phenomenon more closely in Figure 5 by visualising the embedding space created with Siamese models trained solely on the Speech Commands data and in combination with the Spoken Wikipedia Corpus recordings. The model trained only on Speech Commands utterances maps the examples from this dataset to groups with much clearer separability between the particular keywords. Apart from some stray confusions, the only intermixing of keywords happens for the “three-tree” pair, showing that the model is indeed focusing on the acoustic similarity of the provided samples. The inclusion of more diverse training examples from the Spoken Wikipedia Corpus prohibits the model from learning an equally



discriminative mapping for the Speech Commands keywords. The created groups of examples show much more bleed between keywords. Unfortunately, we were unable to devise a simple mitigation technique for this issue. It is possible that a more nuanced training procedure could create a more robust representation using all of the available data.



**Figure 5.** Comparison of the UMAP visualisations of the Speech Commands training examples (SC<sub>1EN</sub>) processed through the Siamese embedder: (a) embeddings generated with a model trained only on the Speech Commands data; (b) embeddings generated with a model trained on recordings both from the Speech Commands and the Spoken Wikipedia Corpus datasets. (We employ a zero minimum distance between embedded points. Other visualisation settings use standard values of the *umap-learn* Python package, i.e., 15 neighbours with a Euclidean metric for Uniform Manifold Approximation and Projection.)

We also extend our investigation with additional experiments employing a prototypical network model instead of the Siamese embedder. In the analysed monolingual settings, this approach proves to be less performant than the Siamese counterpart. In Table 4, we report only the results for training with Speech Commands data, but the tendency remains unchanged with different dataset setups.

Finally, we conclude with evaluations performed on Polish datasets. Unfortunately, the experiments confirm our initial concerns. Models trained solely on such limited datasets are entirely unusable in detecting keywords in continuous recordings.

### 3.3. Keyword Detection in a Cross-Lingual Setup

Following the expected failure of models trained exclusively on Polish recordings, we try to approach the problem of detecting Polish keywords with models trained on English datasets, i.e., in a cross-lingual mode. We hope that the general audio processing capabilities acquired by training on more extensive English datasets will allow for a successful transfer of knowledge to Polish recognition tasks, despite the inherent differences between acoustic features of the languages.

Table 5 summarises our findings in cross-lingual scenarios. When trained solely on the Speech Commands recordings, both Siamese and prototypical models are better in detecting Polish keywords than their counterparts trained on limited Polish data. Nevertheless, this improvement is still insufficient to achieve satisfactory performance. The Siamese model, which outperforms the prototypical approach, achieves a micro-AUPRC of only 20.3% on synthetic mixtures of Polish keywords ( $KW_{PL,mix}$ ). In practical terms, this result means that we can roughly achieve a precision of 70% at a 25% recall rate. However, the outcomes

for individual keywords vary widely. For instance, the best performing one, *umowa*, has a recall of 67% with 96% precision. Unfortunately, although we can find several other classes with potentially usable results, many keywords have a near-zero detection rate.

**Table 5.** Keyword detection performance for models trained and evaluated in cross-lingual scenarios. Metric values are reported in the same manner as in Table 4.

Model	Training <sup>1</sup> (# KW)	Patterns <sup>2</sup>	Evaluation <sup>3</sup> (# KW)	AUPRC		F-Score
				Micro	Macro	
<b>English → Polish</b>						
Siamese	SC 1 <sub>EN</sub> <sup>(30)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	20.3%	10.6%	0.39
Siamese	SC 2 <sub>EN</sub> , CV <sub>EN</sub> <sup>(37)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	10.3%	8.9%	0.31
Siamese	SC 1 <sub>EN</sub> , SWC <sub>EN</sub> <sup>(4K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	15.5%	11.6%	0.34
Siamese	ALL <sub>EN</sub> <sup>(4K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	7.3%	10.7%	0.26
Prototypical	SC 1 <sub>EN</sub> <sup>(30)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	13.4%	2.6%	0.27
Siamese	SC 1 <sub>EN</sub> <sup>(30)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.2%	0.2%	0.03
Siamese	SC 2 <sub>EN</sub> , CV <sub>EN</sub> <sup>(37)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.1%	0.2%	0.02
Siamese	SC 1 <sub>EN</sub> , SWC <sub>EN</sub> <sup>(4K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.1%	0.1%	0.02
Siamese	ALL <sub>EN</sub> <sup>(4K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.1%	0.1%	0.01
Prototypical	SC 1 <sub>EN</sub> <sup>(30)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.3%	0.2%	0.04
<b>Combined (English + Polish) → Polish</b>						
Siamese	ALL <sub>EN</sub> , ALL <sub>PL</sub> <sup>(9K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	6.5%	7.2%	0.23
Prototypical	ALL <sub>EN</sub> , ALL <sub>PL</sub> <sup>(9K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	7.6%	2.7%	0.22
Siamese	ALL <sub>EN</sub> , ALL <sub>PL</sub> <sup>(9K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.1%	0.1%	0.02
Prototypical	ALL <sub>EN</sub> , ALL <sub>PL</sub> <sup>(9K+)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.2%	0.0%	0.04

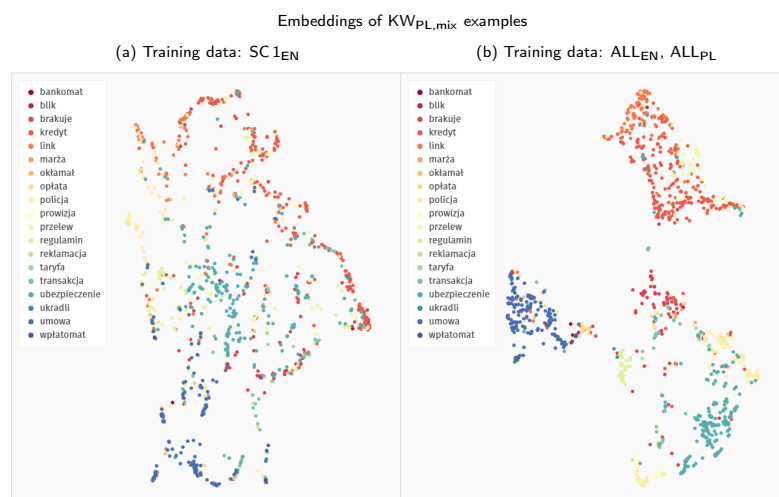
1: Training data are denoted in the same way as in Table 4. 2: All search patterns come from the Polish target keywords dataset (KW<sub>PL</sub>). 3: Evaluation is performed on Polish mixtures (KW<sub>PL,mix</sub>) and real recordings (KW<sub>PL,real</sub>) with 22 keywords.

When looking at possible training dataset extensions, we observe a similar situation as with monolingual models. Training solely on the Speech Commands dataset proves to be the most efficient way to achieve acoustically discriminative embedders. Additional recordings are similarly detrimental in creating an embedding space appropriate for cross-lingual transfer to Polish patterns.

This observation also holds for further extensions with Polish training recordings, though not without some caveats. Initially, we hoped to fill potential gaps in the generated embedding space by introducing additional training examples more closely resembling the phonetic structure of the target patterns. However, the outcome of this process was somewhat ambiguous.

On the one hand, if we compare the 2D representations of the embedding space presented in Figure 6, the second model, supplied with extended training data, groups Polish keywords into much tighter clusters. Although low-dimensionality mappings of complex embedding spaces might be misleading at times, this visual difference most probably indicates that the second model can more effectively discriminate between the keyword classes.

On the other hand, this capability does not translate to an advantage when comparing the detection performance of both models. While we do not have a definite explanation for this phenomenon, we hypothesise that a more dispersed representation might be actually beneficial in our scenario. In contrast to the evaluation on Speech Commands keywords, where the evaluation and search patterns come from the same distribution, Polish patterns used as keyword templates differ in the recording conditions from the YouTube evaluation fragments. Therefore, a broader, less regularised representation might expose more potential points of contact to find close neighbours that could match actual keyword occurrences in audio streams. Such an increased coverage could be significant since the data manifold of Polish search patterns is quite limited to begin with.



**Figure 6.** Comparison of the UMAP visualisations of Polish keywords processed through the Siamese embedder: (a) embeddings generated with a model trained only on the Speech Commands data; (b) embeddings generated with a model trained on all the available data (both English and Polish); Polish keywords are extracted directly from YouTube videos. Synthetic mixtures of these keywords are denoted as  $KW_{PL,mix}$  throughout the results section.

This representational problem is accentuated by evaluations performed on longer, authentic Polish audio streams ( $KW_{PL,real}$ ). In this scenario, all systems fail to provide any hint of usable results. We can devise a two-fold explanation for this behaviour.

First of all, the embedding space learnt by our models does not capture the acoustic differences at a detailed enough level. Therefore, these models cannot handle utterances outside of their limited vocabulary. Non-keyword audio content easily derails the detectors, which is confirmed by numerous false positives.

The second factor is associated with the difficulty of the problem itself. In contrast to our English setups, the evaluation performed on Polish recordings uses longer fragments of naturally sounding speech from diverse recording conditions, thus being the closest to an actual environment in which these kinds of systems might be deployed. Consequently, many keyword occurrences are less perceptible than in semi-synthetic mixtures.

Based on all these observations, we suspect that a self-supervised approach to training acoustic models could be promising in solving similar problems as described in this paper. Self-supervision should help create rich representations, more robustly capturing the differences between various words present in the recording—all without the need for extra hand-labelling. Unfortunately, devising a sensible self-supervised approach is not a trivial task, and each iteration of such an experiment requires a significant computational effort to train the actual model. Therefore, due to the constrained timeline of our project, we were unable to explore this option further.

### 3.4. Keyword Detection with Generic Speech Embeddings

Instead of investigating self-supervised techniques, in this last experimental section, we concentrate on models trained traditionally, in a fully supervised manner, but on much bigger datasets. Based on our assumption that the main factor limiting the performance of our models is the lack of a more generic and robust internal speech representation, we replace our previously analysed similarity models with a pre-trained speech embedding model provided by Google [6]. It has been trained on more than 100,000 h of English

audio clips, which we expect should cover a big part of possible recording conditions and variants of speech. We present the results of applying this model in Table 6, divided into four different approaches.

**Table 6.** Keyword detection performance for the pre-trained speech embedder. We use it either with default detection settings or with an additional post-processing procedure. We also compare the embeddings generated directly from the pre-trained model and from a model fine-tuned on selected datasets. Metric values are reported in the same manner as in Table 4.

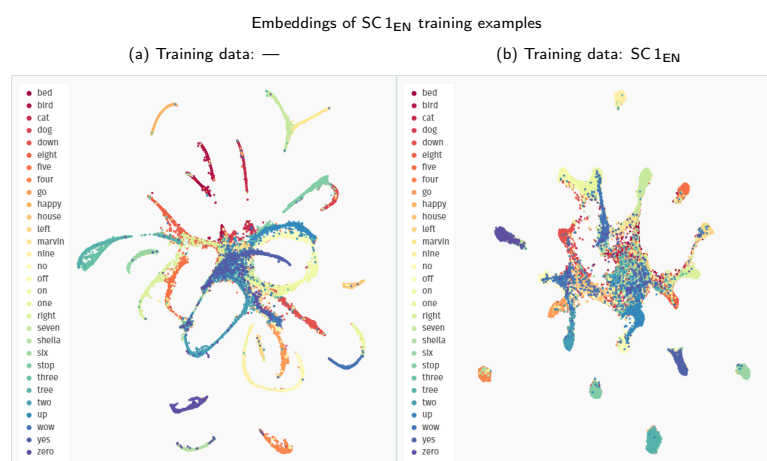
Model	Training <sup>1</sup> (# KW)	Patterns <sup>2</sup>	Evaluation <sup>3</sup> (# KW)	AUPRC		F-Score
				Micro	Macro	
<b>Speech embedder (English), pre-trained</b>						
Google	—	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> <sup>(30)</sup>	15.3%	71.5%	0.41
Google	—	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> <sup>(30)</sup>	3.7%	38.5%	0.07
Google	—	$\Delta$ SC <sub>EN</sub>	$\Delta$ SC <sub>EN,mix</sub> <sup>(5)</sup>	4.6%	4.3%	0.65
Google	—	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	27.2%	62.5%	0.41
Google	—	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	3.4%	21.0%	0.05
<b>Speech embedder (English), pre-trained, with post-processing</b>						
Google	—	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> <sup>(30)</sup>	84.8%	87.7%	0.84
Google	—	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> <sup>(30)</sup>	40.6%	49.8%	0.48
Google	—	$\Delta$ SC <sub>EN</sub>	$\Delta$ SC <sub>EN,mix</sub> <sup>(5)</sup>	86.5%	87.8%	0.85
Google	—	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	46.2%	53.9%	0.69
Google	—	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	6.6%	9.2%	0.17
<b>Speech embedder (English), fine-tuning</b>						
Google	SC 1 <sub>EN</sub> <sup>(30)</sup>	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> <sup>(30)</sup>	15.0%	30.1%	0.30
Google	SC 1 <sub>EN</sub> <sup>(30)</sup>	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> <sup>(30)</sup>	1.2%	1.8%	0.03
Google	WUT <sub>PL</sub> <sup>(36)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	18.4%	19.9%	0.36
Google	WUT <sub>PL</sub> <sup>(36)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	0.7%	2.8%	0.02
<b>Speech embedder (English), fine-tuning, with post-processing</b>						
Google	SC 1 <sub>EN</sub> <sup>(30)</sup>	SC 1 <sub>EN</sub>	SC 1 <sub>EN,mix</sub> <sup>(30)</sup>	24.4%	25.7%	0.42
Google	SC 1 <sub>EN</sub> <sup>(30)</sup>	SC 1 <sub>EN</sub>	SC 1 <sub>EN,Vox</sub> <sup>(30)</sup>	1.3%	1.8%	0.04
Google	WUT <sub>PL</sub> <sup>(36)</sup>	KW <sub>PL</sub>	KW <sub>PL,mix</sub> <sup>(22)</sup>	15.5%	16.0%	0.37
Google	WUT <sub>PL</sub> <sup>(36)</sup>	KW <sub>PL</sub>	KW <sub>PL,real</sub> <sup>(22)</sup>	1.3%	1.5%	0.05

1: The speech embedding model [6] is pre-trained on English YouTube audio clips. In most experiments, we use the generated embeddings directly, without any further training of the model. In fine-tuning experiments, we use utterances from the Speech Commands dataset (SC 1<sub>EN</sub>) or our own recordings (WUT<sub>PL</sub>). 2,3: Search patterns and evaluation recordings are denoted in the same way as in Table 4.

First, we use the pre-trained embedder directly, without any fine-tuning, with the standard detector. This setup means that the pre-trained model returns coordinates in the embedding space for each analysed fragment of the evaluation recording and the keyword patterns. We then compute the distances to each provided keyword example and use properly adjusted thresholds with our primary detection approach involving aggregation and filtering. The macro-results of this system are good on the Speech Commands dataset (SC 1<sub>EN,mix</sub>), although not as good as the specifically trained models. However, the system incorporating the pre-trained embedder is much better with Polish keywords (KW<sub>PL,mix</sub>). It is also the first system to obtain non-zero results on the most interesting, authentic Polish evaluation (KW<sub>PL,real</sub>).

Comparing Figures 5a and 7a, we can observe some significant changes in the embedding space generated for the Speech Commands examples. The Google speech embedder mapping of keyword classes is more complex than in the Siamese model's case. Instead of densely packed clusters, the pre-trained model distributes the encountered examples across more elongated shapes. This change indicates that the speech embedder captures more variability factors in the data, apart from the keyword class, creating a more nuanced representation. Concurrently, it still creates a distinct separation between the classes, although

with some overlap in the middle of the plot. The behaviour in this central region might explain why the raw performance of the pre-trained embedder on Speech Commands data might be slightly worse in direct comparison to a Siamese model.



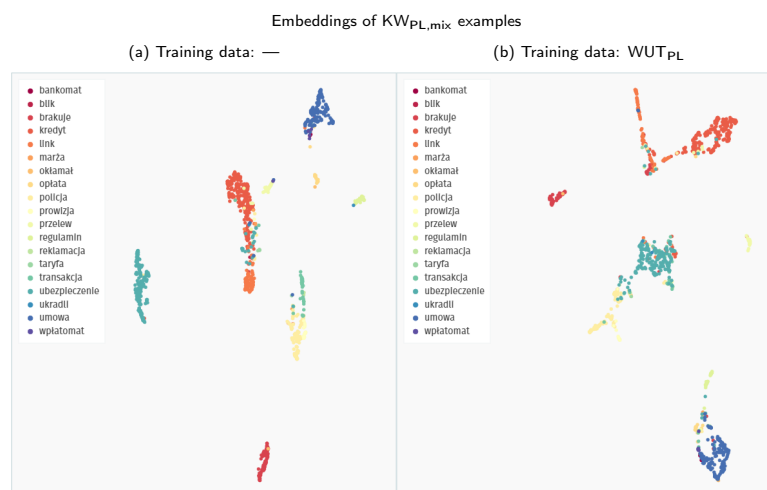
**Figure 7.** Comparison of the UMAP visualisations of the Speech Commands training examples (SC<sub>1EN</sub>) processed through the Google speech embedder: (a) embeddings generated with a pre-trained model without fine-tuning; (b) embeddings generated with a pre-trained model fine-tuned on the Speech Commands dataset.

Despite this robust representation of speech fragments generated by the Google model, a direct application of the returned similarity distances proves to be relatively ineffective in a detection setting. While most keywords produce pretty accurate results, a select few (*eight*, *off*, and *up*) create a tremendous number of false positives. These errors drive down the model's average performance, indicated by the significant discrepancy between the micro- and macro-values of the reported metrics. This problem is presented more explicitly in Figure A1, in Appendix A.

We mitigate this issue by introducing the post-processing approach described in Section 2.3.2. By adjusting the detector setting in this way, we can almost entirely filter out these massive false detections reported by the model, as shown in Figure A2. This modification comes at a small cost of slightly worse outcomes on some of the high-quality keywords, but on the whole, it creates a detection system with a much more practical behaviour.

The performance of the post-processing version of the system on Speech Commands data (SC<sub>1EN,mix</sub>) is comparable to our Siamese and prototypical models trained from scratch. The results for semi-synthetic evaluation with SC<sub>1EN,Vox</sub> are slightly worse, but the performance on the “delta” classes ( $\Delta$ SC<sub>EN,mix</sub>) is on par with the original Speech Commands keywords. This improvement shows that the Google speech embedder has been exposed to an extensive range of potential English keywords, making it easier to adjust the operating vocabulary on the fly.

The most important change from our perspective is the improvement in Polish keyword detection. A good representation of the Polish keywords, as depicted in Figure 8a, allows the model to approach much more sensible levels even with the baseline detector. However, the post-processing variant brings the best F-score on the KW<sub>PL,mix</sub> dataset up to 0.69, significantly outperforming all the other approaches analysed in this paper. The macro-AUPRC values on authentic Polish recordings (KW<sub>PL,real</sub>) are better with the standard detector, but post-processing improves both the micro-AUPRC and the F-score value. Unfortunately, it is still somewhat discouraging, achieving a level of 0.17.



**Figure 8.** Comparison of the UMAP visualisations of Polish keywords processed through the Google speech embedder: (a) embeddings generated with a pre-trained model without fine-tuning; (b) embeddings generated with a pre-trained model fine-tuned on  $WUT_{PL}$  training examples.

Finally, we also evaluate fine-tuned versions of the Google speech embedder, hoping to combine the advantages of both worlds—pre-trained generic representations and dataset-specific mapping. The fine-tuning procedure is described in Section 2.2.3. Unfortunately, our efforts very swiftly prove to be destructible to the intricate representational capabilities of the original model. As evidenced by Figure 7b, fine-tuning on the Speech Commands dataset creates embeddings with some tightly packed clusters, similar to the behaviour of the Siamese model. Still, most of the keywords become intermixed after this procedure. This degradation is also confirmed quantitatively, as the results for the fine-tuned models are comparatively worse across the board. This drop in performance also pertains to fine-tuning on Polish data, although the embedding space shown in Figure 8b seems to be less impacted.

### 3.5. Final Evaluation

As a final step in evaluating our keyword detection systems, we analyse their behaviour on the actual target recordings of call centre conversations ( $CC_{PL,real}$ ). Due to organisational impediments, we could perform this procedure only once, with a limited number of systems. Therefore, we employ the pre-trained embedder solution with our Polish keyword patterns, as such a combination presents the most promising results on continuous Polish recordings. We choose both variants of the system, with the standard detection pipeline and post-processing. The pre-trained embedder is used directly, without fine-tuning.

Table 7 summarises the findings of our final evaluation. The detection pipeline was executed only once with a predetermined detection threshold. Therefore, instead of full AUPRC numbers, we present actual metric values obtained at this sensitivity point.

**Table 7.** Keyword detection performance on the final evaluation dataset ( $CC_{PL,real}$ ).

Model (Detector)	Micro			Macro		
	Precision	Recall	F-Score	Precision	Recall	F-Score
Google	0.4%	18.5%	0.01	14.2%	25.6%	0.18
Google (post)	3.6%	17.0%	0.06	8.2%	22.2%	0.12

Both models perform poorly, especially when looking at the micro-aggregation scheme. As the results of our interim validations on Polish datasets were relatively poor, we did not expect a much better outcome, especially since we had to cope with a domain shift on entirely unseen data.

A more detailed analysis shows that the *Google* model returns a very high number of false positives for two short keywords, “*blik*” and “*link*”. As the utterances of these keywords contain only one syllable, it is understandable that matching based only on acoustic features might be unsuccessful. Although our post-processing method can correctly suppress these erroneous detections, it introduces its own biases, keeping the final performance still at a low level.

Concluding our evaluation, we must admit that the quality of predictions generated by the system with the pre-trained embedder proved to be disappointing. This approach was insufficient in creating a general robust keyword spotter for the Polish language. However, there can still be a small added value of such a system when employed as a support tool to highlight specific keywords. Although it will not recall all the occurrences, it can still help people performing the reviewing work if its precision is sufficiently high. We were able to find a couple of characteristic multi-syllable keywords that exhibited promising results in this regard. For instance, words such as “*reklamacja*” and “*transakcja*” had a precision of over 75% combined with a recall rate of 10–20%. While it is not exactly what we have hoped for when devising the system, our proof-of-concept solution has shown that the main focus when creating robust Polish keyword spotting systems should lie on the mundane task of data annotation.

#### 4. Discussion

##### 4.1. Summary of Findings

In this paper, we have explored the problem of cross-speaker keyword spotting for the low-resource setting of the Polish language.

Our options were limited by the lack of publicly available datasets suitable for training production-quality Polish speech-to-text systems. Therefore, we have focused on spotting keywords with detectors based on acoustic similarity. These approaches are generally less demanding on the data annotation side.

We evaluated two similarity ranking models, i.e., Siamese and prototypical networks. Our experiments with English datasets have shown that these methods can create acoustically discriminative representations of processed recordings when provided with sufficiently diverse training examples. Unfortunately, due to the data scarcity problem, we could not create robust keyword spotters solely on Polish data.

Although the perceptual principles of comparing two audio fragments remain the same on the fundamental level, our acoustic similarity models were unable to generalise from English to Polish. The acoustic differences between languages and recording conditions proved to be too big for such a cross-lingual transfer to succeed.

Therefore, we have evaluated a different approach by utilising a generic speech embedding model provided by Google, extensively trained on thousands of hours of English speech. The advantage provided by a very comprehensive training dataset could be seen in more complex representations of the speech samples and much better adaptability to cross-lingual transfer. Although the evaluation on Polish synthetic recordings was quite promising, even with this pre-trained embedder, we still could not create a system that

would be fully functioning in realistic scenarios and could effectively process naturally sounding continuous audio streams.

#### 4.2. Future Work

Based on our research findings, we reckon that acoustic similarity comparisons can be a viable approach in various audio matching problems. Nevertheless, the task of creating a robust generic embedding space for speech recordings is not easy, especially when no datasets of considerable size are available for the target domain, as shown by our negative results. This outcome hints at a number of approaches that could be evaluated in future works.

First of all, our evaluations show that simply extending the scope of training data with out-of-domain examples is not always profitable. However, it is possible that we were unsuccessful in finding more effective training methods, better suited for mixed datasets. Techniques, such as domain adaptation of embeddings that have proved successful in NLP tasks [47], could help bridge the gap between models trained on generic datasets and evaluation on target recordings with different characteristics or even across languages.

Looking at the visualisations of the generated embeddings, we see that similarity models can usually maintain correct separation between various classes. Therefore, we expect that, with some careful adjustments to the post-processing schemes, we could improve the quality of the final system. Better ways to discard background noises and erroneous detections in continuous recordings could help utilise the whole potential of the similarity classifiers, which exhibit a good performance in more isolated settings.

However, to achieve these goals, more robust validation procedures and datasets would be needed. This problem is particularly relevant since the disparity between the performance of keyword classifiers and keyword detectors is striking. In fact, during our work, we could not verify the performance of our systems on representative English audio streams. We think that establishing new, more realistic evaluation protocols for keyword spotters would be an interesting extension for future work, and it would be valuable for a broad research community. Our research highlights that the task of searching for individual words in an audio stream is much more challenging to solve than the classification of separated words, which most current methods are benchmarked against.

Additionally, recent developments in self-supervised training of audio representations create exciting opportunities for low-resource languages, such as Polish. We expect that solutions such as XLSR-53 [34] could prove helpful in bridging the generalisation gap that we encounter in low-resource scenarios.

In the end, if no practical improvements can be achieved other than by increasing the sheer amount of data, we think that the introduction of more resource-efficient annotation procedures based on active learning could make such efforts realisable with lower budgets. For instance, one approach that could be employed is the clustering of unlabelled data. The *K*-medoids technique was shown to reduce labelling budgets by half in sound classification tasks [48]. When combined with the feedback of a continuously retrained model, we expect that such solutions could greatly improve the annotation workflow.

#### 5. Conclusions

The goal of our work was to create a proof-of-concept solution that could effectively detect Polish keywords in low-quality call centre recordings. Based on our research hypothesis, we developed keyword detectors employing few-shot acoustic similarity models. The models have a satisfactory accuracy in English and for selected Polish keywords, but they fail for many shorter Polish utterances. Effectively, the created software system enables navigation in call centre recordings only for a limited subset of Polish keywords. However, such functionality can still reduce the processing time in the complaint processes.



**Author Contributions:** Conceptualisation, K.J.P. and R.N.; methodology, K.J.P., Ł.L. and K.R.; software, Ł.L., K.J.P. and K.R.; validation, K.J.P. and Ł.L.; formal analysis, K.J.P. and Ł.L.; investigation, K.J.P., Ł.L. and K.R.; resources, Ł.L., K.J.P., K.R. and R.N.; data curation, K.J.P., Ł.L. and K.R.; writing—original draft preparation, K.J.P., Ł.L. and K.R.; writing—review and editing, K.J.P., Ł.L., K.R. and R.N.; visualisation, K.J.P.; supervision, K.J.P. and R.N.; project administration, K.J.P. and R.N.; funding acquisition, R.N. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Warsaw University of Technology statutory research grant in 2021 and by the mBank SA research project in 2020.

**Institutional Review Board Statement:** Not applicable.

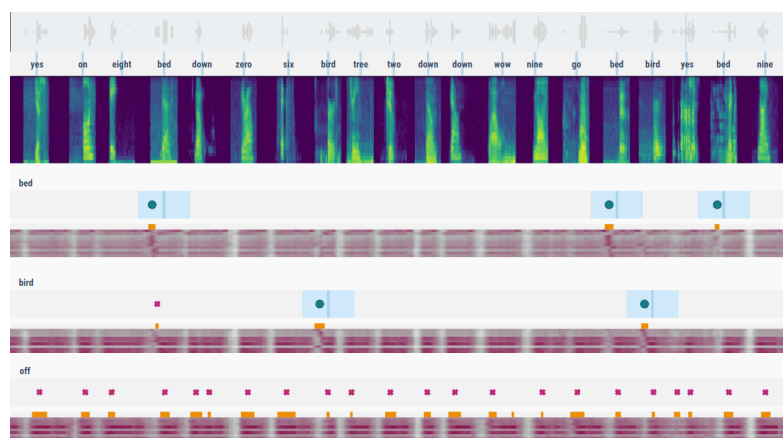
**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Publicly available datasets: Speech Commands v1: [http://download.tensorflow.org/data/speech\\_commands\\_v0.01.tar.gz](http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz) (accessed on 10 December 2021); Speech Commands v2: [http://download.tensorflow.org/data/speech\\_commands\\_v0.02.tar.gz](http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz) (accessed on 10 December 2021); Mozilla Common Voice: <https://commonvoice.mozilla.org/en/datasets> (accessed on 10 December 2021); Spoken Wikipedia Corpus: <https://nats.gitlab.io/swc> (accessed on 10 December 2021); The YouTube fragments with Polish keywords are available on request for research purposes.

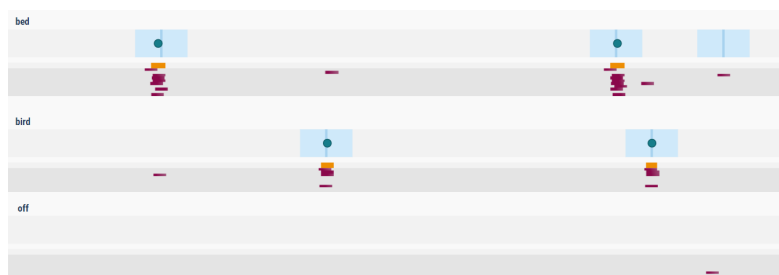
**Acknowledgments:** We would like to thank Karol Chęciński, Piotr Gawrysiak and Kamil Żbikowski for their support.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A



**Figure A1.** Example of a detection report for the Google speech embedding model with standard detection settings, without post-processing. The upper part shows a timeline of the Speech Commands mixture file with a corresponding spectrogram visualisation. The lower part contains an excerpt from the detection reports for three selected keywords. Blue regions indicate actual keyword occurrences with an acceptable detection collar. The main lanes below each keyword show similarity values for each of the provided keyword patterns (i.e., multiple rows per keyword). Darker values correspond to closer matches. Desaturated parts indicate discarded fragments with distances above the threshold. Orange markers denote possible detections. If a sufficiently long streak of detections is generated, actual occurrences are emitted—denoted either by a dot (correct detection) or a cross sign (incorrect).



**Figure A2.** Same detection report for the Google speech embedding model, with post-processing applied to the similarity values.

## References

1. Thomas, S.; Suzuki, M.; Huang, Y.; Kurata, G.; Tuske, Z.; Saon, G.; Kingsbury, B.; Picheny, M.; Dibert, T.; Kaiser-Schatzlein, A.; et al. English broadcast news speech recognition by humans and machines. In Proceedings of the ICASSP 2019—2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 6455–6459.
2. Amodei, D.; Anantharayanan, S.; Anubhai, R.; Bai, J.; Battenberg, E.; Case, C.; Casper, J.; Catanzaro, B.; Cheng, Q.; Chen, G.; et al. Deep Speech 2: End-to-end speech recognition in English and Mandarin. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 173–182.
3. Koch, G.; Zemel, R.; Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In Proceedings of the ICML Deep Learning Workshop, Lille, France, 10–11 July 2015; Volume 2.
4. Snell, J.; Swersky, K.; Zemel, R.S. Prototypical networks for few-shot learning. *arXiv* **2017**, arXiv:1703.05175.
5. Wang, Y.; Salamon, J.; Bryan, N.J.; Bello, J.P. Few-shot sound event detection. In Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 81–85.
6. Lin, J.; Kilgour, K.; Roblek, D.; Sharifi, M. Training keyword spotters with limited and synthesized speech data. In Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 7474–7478.
7. Sainath, T.N.; Parada, C. Convolutional neural networks for small-footprint keyword spotting. In Proceedings of the Sixteenth Annual Conference of the International Speech Communication Association, Dresden, Germany, 6–10 September 2015.
8. Lengerich, C.; Hannun, A. An end-to-end architecture for keyword spotting and voice activity detection. *arXiv* **2016**, arXiv:1611.09405.
9. Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv* **2018**, arXiv:1804.03209.
10. Tang, R.; Lin, J. Honk: A pytorch reimplementation of convolutional neural networks for keyword spotting. *arXiv* **2017**, arXiv:1710.06554.
11. Zhang, Y.; Suda, N.; Lai, L.; Chandra, V. Hello edge: Keyword spotting on microcontrollers. *arXiv* **2017**, arXiv:1711.07128.
12. de Andrade, D.C.; Leo, S.; Viana, M.L.D.S.; Bernkopf, C. A neural attention model for speech command recognition. *arXiv* **2018**, arXiv:1808.08929.
13. Zeng, M.; Xiao, N. Effective combination of DenseNet and BiLSTM for keyword spotting. *IEEE Access* **2019**, *7*, 10767–10775. [[CrossRef](#)]
14. Lin, Z.Q.; Chung, A.G.; Wong, A. Edgespeechnets: Highly efficient deep neural networks for speech recognition on the edge. *arXiv* **2018**, arXiv:1810.08559.
15. Choi, S.; Seo, S.; Shin, B.; Byun, H.; Kersner, M.; Kim, B.; Kim, D.; Ha, S. Temporal convolution for real-time keyword spotting on mobile devices. *arXiv* **2019**, arXiv:1904.03814.
16. Mittermaier, S.; Kürzinger, L.; Waschneck, B.; Rigoll, G. Small-footprint keyword spotting on raw audio data with sinc-convolutions. In Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 7454–7458.
17. Majumdar, S.; Ginsburg, B. MatchboxNet-1D Time-Channel Separable Convolutional Neural Network Architecture for Speech Commands Recognition. *arXiv* **2020**, arXiv:2004.08531.
18. Li, B.; Wu, F.; Lim, S.N.; Belongie, S.; Weinberger, K.Q. On feature normalization and data augmentation. *arXiv* **2020**, arXiv:2002.11102.
19. Coucke, A.; Chlieh, M.; Gisselbrecht, T.; Leroy, D.; Poumeyrol, M.; Lavril, T. Efficient keyword spotting using dilated convolutions and gating. In Proceedings of the ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 6351–6355.
20. Raziq, A.; Hyun-Jin, P. End-to-end streaming keyword spotting. *arXiv* **2018**, arXiv:1812.02802.
21. Mazzawi, H.; Gonzalvo, X.; Kracun, A.; Sridhar, P.; Subrahmanya, N.; Lopez-Moreno, L.; Park, H.J.; Violette, P. *Improving Keyword Spotting and Language Identification via Neural Architecture Search at Scale*; INTERSPEECH: Graz, Austria, 2019; pp. 1278–1282.

22. Guo, J.; Kumatani, K.; Sun, M.; Wu, M.; Raju, A.; Ström, N.; Mandal, A. Time-delayed bottleneck highway networks using a DFT feature for keyword spotting. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 5489–5493.
23. Hannun, A.; Case, C.; Casper, J.; Catanzaro, B.; Diamos, G.; Elsen, E.; Prenger, R.; Sathesh, S.; Sengupta, S.; Coates, A.; et al. Deep Speech: Scaling up End-To-End Speech Recognition. *arXiv* **2014**, arXiv:1412.5567.
24. Battenberg, E.; Chen, J.; Child, R.; Coates, A.; Li, Y.G.Y.; Liu, H.; Sathesh, S.; Sriram, A.; Zhu, Z. Exploring neural transducers for end-to-end speech recognition. In Proceedings of the 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Okinawa, Japan, 16–12 December 2017; pp. 206–213.
25. van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. *arXiv* **2016**, arXiv:1609.03499.
26. Xiong, W.; Wu, L.; Alleva, F.; Droppo, J.; Huang, X.; Stolcke, A. The Microsoft 2017 Conversational Speech Recognition System. *arXiv* **2017**, arXiv:1708.06073.
27. Li, J.; Lavrukhin, V.; Ginsburg, B.; Leary, R.; Kuchaiev, O.; Cohen, J.M.; Nguyen, H.; Gadde, R.T. Jasper: An end-to-end convolutional neural acoustic model. *arXiv* **2019**, arXiv:1904.03288.
28. Kriman, S.; Beliaev, S.; Ginsburg, B.; Huang, J.; Kuchaiev, O.; Lavrukhin, V.; Leary, R.; Li, J.; Zhang, Y. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 6124–6128.
29. Gulati, A.; Qin, J.; Chiu, C.C.; Parmar, N.; Zhang, Y.; Yu, J.; Han, W.; Wang, S.; Zhang, Z.; Wu, Y.; et al. Conformer: Convolution-augmented Transformer for Speech Recognition. *arXiv* **2020**, arXiv:2005.08100.
30. Kubanek, M. Method of speech recognition and speaker identification using audio-visual of polish speech and hidden markov models. In *Biometrics, Computer Security Systems and Artificial Intelligence Applications*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 45–55.
31. Ziółko, M.; Gałka, J.; Ziółko, B.; Jadczyk, T.; Skurzok, D.; Masiór, M. Automatic speech recognition system dedicated for Polish. In Proceedings of the Twelfth Annual Conference of the International Speech Communication Association, Florence, Italy, 28–31 August 2011.
32. Pohl, A.; Ziółko, B. Using part of speech n-grams for improving automatic speech recognition of Polish. In Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition, New York, NY, USA, 19–25 July 2013; pp. 492–504.
33. Baeviski, A.; Zhou, H.; Mohamed, A.; Auli, M. wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv* **2020**, arXiv:2006.11477.
34. Conneau, A.; Baeviski, A.; Collobert, R.; Mohamed, A.; Auli, M. Unsupervised cross-lingual representation learning for speech recognition. *arXiv* **2020**, arXiv:2006.13979.
35. Ardila, R.; Branson, M.; Davis, K.; Henretty, M.; Kohler, M.; Meyer, J.; Morais, R.; Saunders, L.; Tyers, F.M.; Weber, G. Common voice: A massively-multilingual speech corpus. *arXiv* **2019**, arXiv:1912.06670.
36. Panayotov, V.; Chen, G.; Povey, D.; Khudanpur, S. Librispeech: An asr corpus based on public domain audio books. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 19–24 April 2015; pp. 5206–5210.
37. Baumann, T.; Köhn, A.; Hennig, F. The Spoken Wikipedia Corpus collection: Harvesting, alignment and an application to hyperlistening. *Lang. Resour. Eval.* **2019**, *53*, 303–329. [[CrossRef](#)]
38. Żelasko, P.; Ziółko, B.; Jadczyk, T.; Skurzok, D. AGH corpus of Polish speech. *Lang. Resour. Eval.* **2016**, *50*, 585–601. [[CrossRef](#)]
39. Korżinek, D.; Marasek, K.; Brocki, L.; Wołk, K. Polish read speech corpus for speech tools and services. *arXiv* **2017**, arXiv:1706.00245.
40. Peźnik, P. Increasing the Accessibility of Time-Aligned Speech Corpora with Spokes Mix. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7–12 May 2018.
41. Peźnik, P. Spokes-a search and exploration service for conversational corpus data. In Proceedings of the Selected Papers from the CLARIN 2014 Conference, Soesterberg, The Netherlands, 23–25 October 2014; pp. 99–109.
42. Demenko, G.; Grochowski, S.; Klessa, K.; Ogrórkiewicz, J.; Wagner, A.; Lange, M.; Śledziński, D.; Cylwik, N. Jurisdic: Polish speech database for taking dictation of legal texts. In Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08), Marrakech, Morocco, 28–30 May 2008.
43. Szwelnik, T.; Kawalec, J.; Gutowska, D. *Polish Speech Database LDC2019S19*; Linguistic Data Consortium: Philadelphia, PA, USA, 2019. [[CrossRef](#)]
44. Polish & English Language Corpora for Research & Applications. Available online: <http://pelcra.pl/new/snuv> (accessed on 7 October 2021).
45. Nagrani, A.; Chung, J.S.; Xie, W.; Zisserman, A. Voxceleb: Large-scale speaker verification in the wild. *Comput. Sci. Lang.* **2020**, *60*, 101027. [[CrossRef](#)]
46. Mesaros, A.; Heittola, T.; Virtanen, T. Metrics for polyphonic sound event detection. *Appl. Sci.* **2016**, *6*, 162. [[CrossRef](#)]
47. Kruspe, A. A simple method for domain adaptation of sentence embeddings. *arXiv* **2020**, arXiv:2008.11228.
48. Shuyang, Z.; Heittola, T.; Virtanen, T. Active learning for sound event classification by clustering unlabeled data. In Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 March 2017; pp. 751–755.

### **B.3. Automatic hyperparameter tuning in on-line learning: Classic Momentum and ADAM**

Title	Automatic hyperparameter tuning in on-line learning: Classic Momentum and ADAM
Authors	Paweł Wawrzyński, Paweł Zawistowski, Łukasz Lepak
Conference	2020 International Joint Conference on Neural Networks (IJCNN 2020)
Year	2020
DOI	10.1109/IJCNN48605.2020.9207204
Ministerial score	140

# Automatic hyperparameter tuning in on-line learning: Classic Momentum and ADAM

Paweł Wawrzyński, Paweł Zawistowski, Łukasz Lepak

*Institute of Computer Science*

*Warsaw University of Technology*

Nowowiejska 15/19, 00-665 Warsaw, Poland

pawel.wawrzyński@pw.edu.pl, pawel.zawistowski@pw.edu.pl, lukasz.lepak.stud@pw.edu.pl

**Abstract**—We propose a method that adapts hyperparameters, namely step-sizes and momentum decay factors, in on-line learning with classic momentum and ADAM. The approach is based on the estimation of the short- and long-term influence of these hyperparameters on the loss value. In the experimental study, our approach is applied to on-line learning in small neural networks and deep autoencoders. Automatically tuned coefficients surpass or roughly match the best ones selected manually in terms of learning speed. As a result, on-line learning can be a fully automatic process, producing results from the first run, without preliminary experiments aimed at manual hyperparameter tuning.

**Index Terms**—online optimization, Stochastic Gradient Descent, hyperparameter tuning, neural networks, Classic Momentum, ADAM

## I. INTRODUCTION

In this paper, we consider the typical setting for on-line learning: we wish to optimize a parameter,  $\theta \in \mathbb{R}^d$ , of a learning system. For each time step, there exists a known (momentary) loss function  $J(\theta, \xi)$ , where  $\xi$  denotes a randomly generated data sample. The goal of learning is to find the point  $\theta^* \in \mathbb{R}^d$  for which the global loss function

$$\bar{J}(\theta) = EJ(\theta, \xi) \quad (1)$$

attains its minimum. We assume that only the gradient of the momentary loss function,  $\nabla_{\theta} J(\theta, \xi)$ , is available, which is an unbiased estimate of the (unavailable) global loss gradient  $\nabla \bar{J}(\theta)$ .

Most fundamental methods of on-line learning, like stochastic gradient descent [1] (SGD) and classic momentum [2] (CM) require hyperparameters called step-sizes and momentum decay factors that generally depend on the problem and process stage.

In practice, on-line learning is usually conducted by selecting the aforementioned parameters using trial-and-error, which is time consuming and not satisfying. There have been attempts, like AdaGrad [3] or ADAM [4], to design an algorithm that does not depend on any manually tuned hyperparameters, or at least works well with the defaults. However, these algorithms also require step-sizes and momentum decay factors, and their default values do not guarantee good performance for all learning problems.

In this paper, we design an algorithm that optimizes the step-size and the momentum decay factor in CM and ADAM while these methods are running. The algorithm presented here extends and refines the ASDM method presented by Wawrzyński [5]. The focus is put on a novel method of analyzing the long-term influence of the parameters on the momentary loss value. This method overcomes important ASDM deficiencies.

In addition, the analysis of the short-term influence of the step-size and the momentum decay factor on a momentary value of  $\theta$  introduced in the previous paper for CM is here extended to ADAM. The resulting learning algorithm has been analyzed experimentally and compared in simulations with CM, accelerated gradient, ADAM, AdaGrad, and AdaDelta. All algorithms are applied to 14 different learning problems and two parameter settings.

The main contributions of this paper are thus as follows:

- we propose to estimate the initial step-size in ASDM using the estimate of the largest eigenvalue of the Hessian,
- we propose to define a quality measure for the algorithm's hyperparameters' adjustments based on their influence on the long-term performance of the learning process,
- we introduce indicators of optimization instability and propose heuristics to ensure that the process remains stable,
- we conduct extensive experiments to test the approach against current state-of-the-art methods.

The paper is organized as follows. The next section presents related work. In Sec. III, the formal definition of the considered problem is given. In Sec. IV, the influence of hyperparameters on the learning process is derived, while Sec. V presents the resulting algorithm. Sec. VI reports the experimental study with on-line learning in feedforward neural networks. The last section concludes the paper.

## II. RELATED WORK

The earliest methods of on-line learning include SGD, which has its roots in the aforementioned works of Robbins and Monro [1] and Kiefer et al. [6]. Two other classic approaches, which build upon the concepts of SGD and are directly related to this paper, are classic momentum [2, 7] (CM) and accelerated gradient [8] (AG). Both CM and AG are based on the concept of extracting a velocity component

from the iterative update procedure utilized by SGD; they only differ in the way this component gets updated.

The CM approach, which is formally defined in Sec. III of this paper, has been thoroughly analyzed by Qian [9], where the convergence bounds for the learning rate and momentum parameters have been analyzed. Bhaya and Kaszkurewicz [10] present a view on CM from the control perspective on the conjugate gradient algorithm [11] and perform an analysis in terms of Lyapunov stability to choose the learning rate and momentum parameters.

There have been multiple attempts to create algorithms which adapt the momentum term to improve CM performance. These include using one dimensional minimisation to estimate this term [12].

Swanston et al. [13] propose to use the angle between the previous update steps and the current one to determine the magnitude of the momentum term.

Graepel and Schraudolph [14] utilize a curvature matrix connected with a forgetting factor to formulate the Stable Adaptive Momentum approach.

The back-propagation Gradient Descent Adaptive Momentum Algorithm [15] (GDAM) adapts the momentum while maintaining a fixed learning rate.

Hameed et al. [16] utilize eigenvalues of the autocorrelation matrix of the optimized model's inputs to adapt momentum in a back-propagation algorithm.

YellowFin [17] is an approach that adapts the learning rate and momentum term by approximating the curvature using single-dimensional noisy quadratic models.

In order to get rid of free parameters, the classic methods have also been combined with gradient normalization in multiple approaches. AdaGrad [3] scales the gradient descent step at time  $t$  with an inversed diagonal matrix constructed using a cumulative sum of gradient products from times  $1, \dots, t$ . However, it requires setting a global learning rate manually and results in a decay of learning rates as the optimization progresses. The AdaDelta method [18] addresses these issues by restricting the cumulative sum to a time window of fixed size and utilizing a diagonal Hessian approximation to obtain correct units for the parameter update vectors. A similar remedy to the decaying learning rates problem present in AdaGrad was suggested for RMSProp [19], which also suggests applying a running average of the gradient magnitude to scale the global learning rate.

ADAM and ADAMAX [4] are currently among the most widely used optimization algorithms that utilize first and second-order moment corrections in their parameter update rules. A version incorporating Nesterov momentum into ADAM is introduced by Dozat [20] and called NADAM.

Although ADAM is a robust and widely used approach in the field of deep learning models, there exist cases in which it is not able to converge [21]. This has been tackled by the AMSGRAD [21] approach. Furthermore, AdaBound and AmsBound methods [22] try to address the problem (signalled by Wilson et al. [23]) of extreme learning rates which occurs

in, respectively, ADAM and AMSGRAD and slows down the optimization process in the long run.

To the best of our knowledge, the only method that adjusts the momentum decay factor on-line in CM was introduced by Schraudolph and Graepel [24], where a stochastic version of the conjugate gradient algorithm is presented.

In recent years, tremendous progress has been made in the field of online convex optimization, which considers a simplified version of the problem analyzed here in which  $\bar{J}$  is convex. One interesting branch of relevant research is connected with meta-descent methods like the stochastic meta-descent algorithm [25] (SMD) which adapts the learning rates in successive iterations using an exponentiated gradient descent controlled by a global learning rate.

The hypergradient descent method [26] (HD) applies gradient descent directly on the learning rate parameters in an online fashion. The  $L^4$  adaptation scheme [27] is a meta-algorithm which calculates step size at every iteration using a linearization of the loss function (inspired by root-finding Newton's method).

The limitations of current meta-optimization approaches are explored by Wu et al. [28]. The authors argue that short time horizons typically used in meta-optimization settings lead to using too small learning rates — this poses a significant challenge in that area of research.

### III. NOTATION AND PROBLEM FORMULATION

In the below equations,  $t$  denotes discrete time,  $\theta_t \in \mathbb{R}^d$  is the optimized vector,  $m_t \in \mathbb{R}^d$  is an auxiliary vector called momentum,  $\beta_t > 0$  is the step-size,  $\lambda_t \in (0, 1)$  denotes the momentum decay factor, the symbol  $\circ$  denotes the Hadamard (elementwise) product, and  $s_t \in \mathbb{R}^d$  is a vector that normalizes different coordinates of  $\theta$ .

Additionally, we denote  $g(\theta, \xi) = \nabla_{\theta} J(\theta, \xi)$ . In the context of feedforward neural network training,  $\theta$  is a vector of neural weights,  $\xi_t$  is a data sample (an input-output pair or a minibatch of these), and  $g(\theta, \xi)$  is computed by means of gradient backpropagation.

We wish to find the minimum of  $\bar{J}$  (1) using the following procedure:

$$\begin{aligned} m_t &= \lambda_t m_{t-1} - \beta_t g(\theta_t, \xi_t) \circ s_t \\ \theta_{t+1} &= \theta_t + m_t, \quad t = 1, 2, \dots \end{aligned} \quad (2)$$

In original CM, the elements of  $s_t$  are uniformly equal to 1. The ADAM algorithm implements (2) with  $s_t$  containing the inverses of square roots of average squares of elements of  $g(\theta_{t-i}, \xi_{t-i}), i \geq 0$ .

The problem considered here is how to tune  $\beta_t$  and  $\lambda_t$  on the run of procedure (2) to make it most efficient.

### IV. QUALITY INDEX FOR OPTIMIZATION OF $\beta$ AND $\lambda$

The approach to  $\beta$  and  $\lambda$  optimization in the course of learning can be summarized as follows:

- the influence of previous values of  $\beta$  and  $\lambda$  on the current  $\theta_t$  and the exponentially smoothed  $\theta_t$ , denoted

by  $\bar{\theta}_t$ , is analyzed; the parameters of smoothing also undergo constant optimization,

- the values of  $\beta$  and  $\lambda$  are being incrementally adjusted in the direction, that if these parameters had been pushed before, then current positions of  $\theta_t$  and  $\bar{\theta}_t$  would be better.

The idea of optimization of the current loss,  $J(\theta_t, \xi_t)$ , by manipulating previous values of  $\beta$  and  $\lambda$  is not new. However, when applied directly, it yields small  $\beta$  and  $\lambda$ , since their small values are the best way to suppress random fluctuations of  $\theta_t$  thereby minimizing the current loss. But small  $\beta$  and  $\lambda$  lead to slow learning in the long term. Therefore, we analyze how  $\beta$  and  $\lambda$  influence  $\bar{\theta}_t$ , whose improvement rate roughly reflects the long-term speed of learning.

Technically the aggregation of the impact of  $\beta_i$  and  $\lambda_i$  on future values of  $\theta_t$  is done by means of an operator,  $\mathcal{S}_\gamma$ , which is defined for  $\gamma \in (0, 1]$  as follows

$$\mathcal{S}_\gamma \frac{dv}{d\alpha_k} = \sum_{i \leq k} \gamma^{k-i} \frac{dv}{d\alpha_i}. \quad (3)$$

It is a discounted sum of derivatives of the same value  $v$  with respect to parameters  $\alpha_i$ , in which the weight of a specific derivative decreases with growing  $k - i$ . Short-term influence of  $\beta_k, \lambda_k$  on  $m_t, \theta_{t+1}, t \geq k$  can be expressed in the following recursive equations:

$$\mathcal{S}_\gamma \frac{dm_t}{d\lambda_t} = m_{t-1} + \lambda_t \gamma \mathcal{S}_\gamma \frac{dm_{t-1}}{d\lambda_{t-1}} - \beta_t \gamma \frac{\partial g_t}{\partial \theta_t} \mathcal{S}_\gamma \frac{d\theta_t}{d\lambda_{t-1}} \circ s_t \quad (4)$$

$$\mathcal{S}_\gamma \frac{d\bar{\theta}_{t+1}}{d\lambda_t} = \gamma \mathcal{S}_\gamma \frac{d\theta_t}{d\lambda_{t-1}} + \mathcal{S}_\gamma \frac{dm_t}{d\lambda_t}, \quad (5)$$

$$\mathcal{S}_\gamma \frac{dm_t}{d\beta_t} = \lambda_t \gamma \mathcal{S}_\gamma \frac{dm_{t-1}}{d\beta_{t-1}} - g_t \circ s_t - \beta_t \gamma \frac{\partial g_t}{\partial \theta_t} \mathcal{S}_\gamma \frac{d\theta_t}{d\beta_{t-1}} \circ s_t \quad (6)$$

$$\mathcal{S}_\gamma \frac{d\bar{\theta}_{t+1}}{d\beta_t} = \gamma \mathcal{S}_\gamma \frac{d\theta_t}{d\beta_{t-1}} + \mathcal{S}_\gamma \frac{dm_t}{d\beta_t}. \quad (7)$$

For brevity,  $g_t$  is written in here in place of  $g(\theta_t, \xi_t)$ . The above equations can be derived following the original work of Wawrzynski [5].

#### A. Influence of $\beta$ and $\lambda$ on the trend in $\theta_t$

Here we analyze the influence of  $\beta$  and  $\lambda$  on the long-term movement of  $\theta_t$ . In this order, we define  $\bar{\theta}_t$  as a smoothed  $\theta_t$ , namely

$$\bar{\theta}_1 = \theta_1, \quad \bar{\theta}_{t+1} = \mu \bar{\theta}_t + (1 - \mu) \theta_t, \quad (8)$$

where  $\mu \in [0, 1)$  is optimized to minimize  $J(\bar{\theta}_t, \xi_t)$ .

Let us visualize the optimization process as a descent down a multidimensional valley between steep slopes. The goal is to descend in the direction of the bottom of this valley and  $\bar{\theta}_t$  plays the role of the projection of  $\theta_t$  onto its bottom. We will optimize  $\beta_t$  and  $\lambda_t$  to approximately minimize a linear combination of  $J(\theta_t, \xi_t)$  and  $J(\bar{\theta}_t, \xi_t)$ . For this purpose we analyze how fast  $\bar{\theta}_t$  is moving depending on  $\beta$  and  $\lambda$ .

Using the  $\mathcal{S}_\gamma$  operator, it is easy to quantify the influence of  $\beta$  and  $\lambda$  on  $\bar{\theta}_t$  (8), namely

$$\mathcal{S}_\gamma \frac{d\bar{\theta}_{t+1}}{d\beta_t} = \mu \gamma \mathcal{S}_\gamma \frac{d\bar{\theta}_t}{d\beta_{t-1}} + (1 - \mu) \gamma \mathcal{S}_\gamma \frac{d\theta_t}{d\beta_{t-1}}, \quad (9)$$

$$\mathcal{S}_\gamma \frac{d\bar{\theta}_{t+1}}{d\lambda_t} = \mu \gamma \mathcal{S}_\gamma \frac{d\bar{\theta}_t}{d\lambda_{t-1}} + (1 - \mu) \gamma \mathcal{S}_\gamma \frac{d\theta_t}{d\lambda_{t-1}}. \quad (10)$$

For  $\bar{\theta}_t$  to play its role in the projection of  $\theta_t$  on the valley's bottom,  $J(\bar{\theta}_t)$  needs to be minimized with respect to  $\mu$ . In order to adjust  $\mu$  in the course of learning with CM, we notice that from (8) we have

$$\frac{d\bar{\theta}_{t+1}}{d\mu} = \bar{\theta}_t - \theta_t + \mu \frac{d\bar{\theta}_t}{d\mu}, \quad (11)$$

and then

$$\frac{dJ(\bar{\theta}_t, \xi_t)}{d\mu} = \frac{dJ(\bar{\theta}_t, \xi_t)}{d\bar{\theta}_t} \frac{d\bar{\theta}_t}{d\mu}. \quad (12)$$

The above derivative will indicate the direction of incremental adjustments of  $\mu$ . In order to compute  $\theta_t - \bar{\theta}_t$  in (11) conveniently we define

$$\phi_1 = 0, \quad \phi_{t+1} = m_t + \mu \phi_t. \quad (13)$$

A simple induction combining (8) and (13) reveals that  $\bar{\theta}_t - \theta_t = -\phi_t$ .

#### B. Optimization of $\beta$ and $\lambda$

In order to maintain the desired trend in  $\theta_t$  both in the short and long-term we propose here to minimize (with respect to  $\beta$  and  $\lambda$ ) the following quality index

$$Q_t = J(\bar{\theta}_t, \xi_t) + r_t^T (\theta_t - \theta_{t-1}). \quad (14)$$

where  $r_t = g(\bar{\theta}_t, \xi_t)$  is treated as a constant.

The first term of the sum in (14) is crucial as it directly leads to decreasing the loss value observed for  $\bar{\theta}_t$ . However, a second term needs to be included in order to penalize the fluctuations of consecutive  $\theta_t$  values. This penalty is for the increment  $\theta_t$  along the gradient of the loss function, thereby increasing the loss.

Then, we propose recognizing how infinitesimal changes of  $\beta_k, \lambda_k$  for  $k < t$  would influence (14), which can be quantified as, respectively,

$$\mathcal{S}_\gamma \frac{dQ_t}{d\beta_{t-1}} = g(\bar{\theta}_t, \xi_t)^T \left( \mathcal{S}_\gamma \frac{d\bar{\theta}_t}{d\beta_{t-1}} + \mathcal{S}_\gamma \frac{dm_{t-1}}{d\beta_{t-1}} \right), \quad (15)$$

$$\mathcal{S}_\gamma \frac{dQ_t}{d\lambda_{t-1}} = g(\bar{\theta}_t, \xi_t)^T \left( \mathcal{S}_\gamma \frac{d\bar{\theta}_t}{d\lambda_{t-1}} + \mathcal{S}_\gamma \frac{dm_{t-1}}{d\lambda_{t-1}} \right). \quad (16)$$

Next,  $\beta$  and  $\lambda$  are adjusted in the directions opposite to the above derivatives.

#### V. ALGORITHM ASDM2

Algorithm 1 is based on pillars presented in the previous sections. When the symbol “ $\leftarrow$ ” is used, the full assignment is given on the right side of the symbol. The notation introduced above is generally preserved in the algorithm but simplified e.g., the subscript  $t$  is omitted. The algorithm has two versions:

```

1 Assign 0 to all estimators
2 Initialize  $\theta$ 
3 Assign  $t \leftarrow 1$ , set any  $\beta \neq 0$ 
4 (If needed) Update  $s$ 
5  $\frac{dQ}{d\beta} \leftarrow g(\bar{\theta}, \xi_t)^T (\mathcal{S}_\gamma \frac{d\theta}{d\beta} + \mathcal{S}_\gamma \frac{dm}{d\beta})$ 
6  $\frac{dQ}{d\lambda} \leftarrow g(\bar{\theta}, \xi_t)^T (\mathcal{S}_\gamma \frac{d\theta}{d\lambda} + \mathcal{S}_\gamma \frac{dm}{d\lambda})$ 
7  $\frac{dJ}{d\mu} \leftarrow g(\bar{\theta}, \xi_t)^T \frac{d\theta}{d\mu}$ 
8 if  $t \leq t_0$  then
9    $\beta \leftarrow \frac{1}{2} \left( \frac{t-1}{t} \frac{1}{2} \beta^{-1} + \frac{1}{t} \left\| \frac{\partial g(\theta, \xi_t)}{\partial \theta} g_t \right\| / \|g_t\| \right)^{-1}$ 
10   $\alpha \leftarrow \ln \beta$ 
11   $\lambda \leftarrow 0.5$ 
12   $\eta \leftarrow -\ln(1 - \lambda)$ 
13 else
14   $\alpha \leftarrow \alpha - \delta \frac{dQ}{d\beta} / \sqrt{\mathcal{A}(\frac{dQ}{d\beta})^2}$ 
15   $\beta \leftarrow \exp(\alpha)$ 
16   $\eta \leftarrow \eta - \delta \frac{dQ}{d\lambda} / \sqrt{\mathcal{A}(\frac{dQ}{d\lambda})^2}$ 
17   $\eta \leftarrow \max\{-\ln(1 - \lambda_{\min}), \eta\}$ 
18   $\lambda \leftarrow 1 - \exp(-\eta)$ 
19   $\nu \leftarrow \nu - \delta \frac{dJ}{d\mu} / \sqrt{\mathcal{A}(\frac{dJ}{d\mu})^2}$ 
20   $\mu \leftarrow 1 - \exp(-\nu)$ 
21  $\mathcal{S}_\gamma \frac{d\theta}{d\beta} \leftarrow \frac{d\theta(\theta, \xi_t)}{d\theta} \mathcal{S}_\gamma \frac{d\theta}{d\beta}$ 
22  $\mathcal{S}_\gamma \frac{d\theta}{d\lambda} \leftarrow \frac{d\theta(\theta, \xi_t)}{d\theta} \mathcal{S}_\gamma \frac{d\theta}{d\lambda}$ 
23 if  $t > 1$  then
24    $\alpha_0 \leftarrow \ln \left( \frac{1}{2} \min \left\{ \frac{\|\mathcal{S}_\gamma \frac{d\theta}{d\beta}\|}{\|\mathcal{S}_\gamma \frac{d\theta}{d\beta}\|}, \frac{\|\mathcal{S}_\gamma \frac{d\theta}{d\lambda}\|}{\|\mathcal{S}_\gamma \frac{d\theta}{d\lambda}\|} \right\} \right)$ 
25   if  $\alpha > \alpha_0$  then
26      $\alpha \leftarrow \alpha - 2\delta$ 
27      $\beta \leftarrow \exp(\alpha_0)$ 
28    $\gamma \leftarrow \min \left\{ 1, C \frac{\mathcal{A}\|g\|^2}{\|\mathcal{S}_\gamma \frac{d\theta}{d\beta}\|^2}, C \frac{\mathcal{A}\|m\|^2}{\|\mathcal{S}_\gamma \frac{d\theta}{d\lambda}\|^2} \right\}$ 
29   if  $\lambda > \gamma$  then
30      $\eta \leftarrow \eta - 2\delta$ 
31      $\lambda \leftarrow \gamma$ 
32 Update  $\bar{\theta}$  with (8)
33 Update  $\mathcal{S}_\gamma \frac{d\theta}{d\beta}$  with (9)
34 Update  $\mathcal{S}_\gamma \frac{d\theta}{d\lambda}$  with (10)
35 Update  $\frac{d\theta}{d\mu}$  with (11)
36 Update  $\mathcal{S}_\gamma \frac{dm}{d\beta}$  with (6)
37 Update  $\mathcal{S}_\gamma \frac{dm}{d\lambda}$  with (4)
38 Update  $m$  and  $\theta$  with (2)
39 Update  $\phi$  with (13)
40 Update  $\mathcal{S}_\gamma \frac{d\theta}{d\beta}$  with (7)
41 Update  $\mathcal{S}_\gamma \frac{d\theta}{d\lambda}$  with (5)
42  $\mathcal{A}(\frac{dQ}{d\beta})^2 \leftarrow w_t^p \mathcal{A}(\frac{dQ}{d\beta})^2 + (1 - w_t^p) (\frac{dQ}{d\beta})^2$ 
43  $\mathcal{A}(\frac{dQ}{d\lambda})^2 \leftarrow w_t^p \mathcal{A}(\frac{dQ}{d\lambda})^2 + (1 - w_t^p) (\frac{dQ}{d\lambda})^2$ 
44  $\mathcal{A}(\frac{dJ}{d\mu})^2 \leftarrow w_t^p \mathcal{A}(\frac{dJ}{d\mu})^2 + (1 - w_t^p) (\frac{dJ}{d\mu})^2$ 
45  $\mathcal{A}\|g\|^2 \leftarrow w_t^p \mathcal{A}\|g\|^2 + (1 - w_t^p) \|g_t\|^2$ 
46  $\mathcal{A}\|m\|^2 \leftarrow w_t^p \mathcal{A}\|m\|^2 + (1 - w_t^p) \|m\|^2$ 
47 Increment  $t$  and go to Line 4

```

**Algorithm 1:** Autonomous stochastic descent with momentum version 2: ASDM2.

- ASDM2/b is based on CM and applies no gradient normalization:  $s_t$  is the vector of 1s (“/b” stands for “bare”),
- ASDM2/n is based on ADAM i.e., CM with gradient normalization (hence “/n”).

The core of the algorithm is Line 38, where the basic adjustment of  $m$  and  $\theta$  is done. The algorithm utilizes several technicalities such as (i) initialization of  $\beta$  based on a rough estimate of the inverse of the largest eigenvalue of the Hessian  $\nabla^2 \bar{J}$  (Lines 3 and 9), (ii) re-parameterization of  $\beta$ ,  $\lambda$ , and  $\mu$  (Lines 10, 12, 15, 18, 20, 27) (iii) keeping  $\beta$ ,  $\lambda$ , and  $\gamma$  small enough to prevent abnormal operation of the algorithm (Lines 25–28), and (iv) exponential smoothing of some estimates (Lines 42–46). All the aforementioned technicalities are discussed below.

#### A. Initialization of $\beta$ and $\lambda$

According to [5], a good initial  $\beta$  would be the inverse of the largest eigenvalue of the global loss function Hessian i.e.,

$$\beta \approx \left( \max_{v \in \mathbb{R}^d} \|\nabla^2 \bar{J}(\theta)v\| / \|v\| \right)^{-1}. \quad (17)$$

As it is not possible to determine that value, we notice that

$$\nabla^2 \bar{J}(\theta) = E \nabla^2 J(\theta, \xi) = E \frac{\partial g(\theta, \xi)}{\partial \theta} \quad (18)$$

and set  $\beta_t$  equal to half the inverse of the average of values maximized in (17) with  $v = g_i$ , namely

$$\beta_t = \frac{1}{2} \left( \frac{1}{t} \sum_{i=1}^t \left\| \frac{\partial g(\theta_i, \xi_i)}{\partial \theta_i} g_i \right\| / \|g_i\| \right)^{-1} \quad (19)$$

for  $t \leq t_0$ , where  $t_0 > 0$  determines how long the initial stage of learning lasts. For  $t \leq t_0$ , we set  $\lambda_t \equiv 0.5$ . See Lines 8–12, where  $\beta_t$  is calculated recursively on the basis of  $\beta_{t-1}$  and (19).

#### B. Re-parameterization and adjusting $\beta$ , $\lambda$ , and $\mu$

We want to adjust  $\beta$ ,  $\lambda$ , and  $\mu$  with increments of controlled magnitude, regardless of the actual scale of  $\beta$ ,  $1 - \lambda$ , and  $1 - \mu$ . In this order, the parameters  $\alpha$ ,  $\eta$ , and  $\nu$  are introduced and the following equivalence is established

$$\beta = \exp(\alpha), \quad \alpha = \ln(\beta), \quad (20)$$

$$\lambda = 1 - \exp(-\eta), \quad \eta = -\ln(1 - \lambda), \quad (21)$$

$$\mu = 1 - \exp(-\nu), \quad \nu = -\ln(1 - \mu). \quad (22)$$

Technically, these are  $\alpha$ ,  $\eta$ , and  $\nu$  that are incrementally adjusted. Since the updates are normalized,  $\alpha$ ,  $\eta$ , and  $\nu$  are updated on average  $\pm \delta$ , thus  $\beta$ ,  $1 - \lambda$ , and  $1 - \mu$  are updated by factor  $(1 \pm \delta)$ , where  $\delta$  is a meta-stepsizes. See Lines 10, 12, 15, 18, 20, and 27.

Effectively, the step-size  $\beta$  and the momentum decay factor  $\lambda$  are adjusted according to the discussion in Sec. IV-B. The main steps of these operations are in Lines 5, 6, 14, and 16. The supplementary steps are in Lines 32–34, 35–37, 40–41.

Additionally, in order to locate  $\bar{\theta}_t$  properly, the  $\mu$  parameter needs to be adjusted. That happens in Lines 7, 19, and 35.



### C. Keeping $\beta$ , $\lambda$ , and $\gamma$ small enough

As discussed above, the learning becomes unstable when the step-size  $\beta$  becomes significantly larger than

$$\|v\|/\|\nabla^2 \bar{J}(\theta_t)v\| \quad (23)$$

for a certain  $v \in \mathbb{R}^d$ . We utilize the fact that  $\nabla^2 J(\theta_t, \xi_t)$  is multiplied by a vector in (6) and (4). Namely, early signs of instability are detected when  $\beta$  is larger than

$$\beta_0 = \frac{1}{2} \min \left\{ \frac{\left\| \mathcal{S}_\gamma \frac{d\theta_t}{d\beta_{t-1}} \right\|}{\left\| \frac{\partial g_t}{\partial \theta_t} \mathcal{S}_\gamma \frac{d\theta_t}{d\beta_{t-1}} \right\|}, \frac{\left\| \mathcal{S}_\gamma \frac{d\theta_t}{d\lambda_{t-1}} \right\|}{\left\| \frac{\partial g_t}{\partial \theta_t} \mathcal{S}_\gamma \frac{d\theta_t}{d\lambda_{t-1}} \right\|} \right\}. \quad (24)$$

When that happens (Line 25),  $\beta_0$  is used instead of  $\beta$  (Line 27), and  $\beta$  is decreased (Line 26).

Both  $\gamma$  and  $\lambda$  are exponential decay factors that determine the memory lengths of certain estimators. The term  $\mathcal{S}_\gamma(d\theta_{t+1}/d\lambda_t)$  captures the influence of the previous values of  $\lambda_t$  on  $\theta_{t+1}$ . The memory length defined by  $\gamma$  should be at least as large as the memory length defined by  $\lambda$ . Therefore, whenever  $\gamma < \lambda$ ,  $\lambda$  is being decreased (Lines 29, 30). Also,  $\lambda$  is kept equal to at least  $\lambda_{\min}$ , as very small values of this parameter hardly ever work well in practice.

It can be seen from (7) and (5) that  $\mathcal{S}_\gamma(d\theta_{t+1}/d\lambda_t)$  and  $\mathcal{S}_\gamma(d\theta_{t+1}/d\beta_t)$  are adjusted additively by  $g_t$  and  $m_t$ , respectively. Considering the limited precision of floating point numbers,  $\|\mathcal{S}_\gamma(d\theta_{t+1}/d\lambda_t)\|$  and  $\|\mathcal{S}_\gamma(d\theta_{t+1}/d\beta_t)\|$  can not be too many orders of magnitude larger than average  $\|g_t\|$  and  $\|m_t\|$ , respectively.

We apply  $\gamma$  small enough to assure

$$\begin{aligned} \|\mathcal{S}_\gamma(d\theta_{t+1}/d\lambda_t)\|^2 &\leq C \cdot \text{average}\|g_t\|^2 \\ \|\mathcal{S}_\gamma(d\theta_{t+1}/d\beta_t)\|^2 &\leq C \cdot \text{average}\|m_t\|^2, \end{aligned}$$

where  $C$  is a large constant depending on the representation of numbers. It expresses how many times one number can be larger than another with their addition still being effective. In our experiments, we set  $C = 10^8$ .

### D. Exponential smoothing

Let  $x_t, t = 1, 2, 3, \dots$  be a sequence. Let  $\mathcal{A}x_t$  be the exponentially moving average of  $(x_t)$ , namely

$$\mathcal{A}x_t = \frac{\sum_{i=0}^{t-1} \rho^i x_{t-i}}{\sum_{i=0}^{t-1} \rho^i} \quad (25)$$

for  $\rho \in (0, 1]$ . Elementary transformations reveal that

$$\mathcal{A}x_t = w_t^\rho \mathcal{A}x_{t-1} + (1 - w_t^\rho)x_t \quad (26)$$

for  $w_t^\rho = \rho(1 - \rho^{t-1})/(1 - \rho^t)$ . Exponential smoothing is applied in Lines 42–46 of ASDM2. The terms  $\mathcal{A}(\frac{dQ}{d\beta})^2$ ,  $\mathcal{A}(\frac{dQ}{d\lambda})^2$ , and  $\mathcal{A}(\frac{dJ}{d\mu})^2$  play the role of scalar variables in the algorithm.

### E. Gradient normalization

If the algorithm applies gradient normalization (Line 4), the normalization takes the following form

$$\begin{aligned} \mathcal{A}(g \circ g)_i &\leftarrow w_t^\rho \mathcal{A}(g \circ g)_i + (1 - w_t^\rho)g_i(\theta_t, \xi_t)^2, \quad (27) \\ s_i &\leftarrow 1/\sqrt{\mathcal{A}(g \circ g)_i + \epsilon}, \quad (28) \end{aligned}$$

where  $\epsilon > 0$  is a coefficient that prevents division by zero,  $\rho$  is a decay factor (like  $\rho = 0.999$ ), and subscript  $i$  denotes the  $i$ -th coordinate of the vector.

### F. Coefficients

The algorithm requires several coefficients to be provided. Their values applied in the experiments discussed below are based mainly on common-sense, and are as follows:  $\epsilon = 10^{-8}$ ,  $\delta = 0.0005$ ,  $\lambda_{\min} = 0.5$ ,  $\rho = 0.999$ ,  $t_0 = 10$ , and  $C = 10^8$ . On several occasions, a certain value is set below a certain threshold (Line 24) or above a certain value (Line 26, 30). In all such cases, “below” is twice smaller, and “above” is twice larger.

The initial  $\mu$  is set equal to 0.99.

While all these coefficients may undergo some optimization, their default values give good performance over diverse testbed learning tasks.

## VI. EXPERIMENTAL STUDY

This section reports experiments with the algorithms presented in the previous section. The algorithms are tested in three settings: firstly, training shallow neural classifiers for 10 arbitrary classification problems from the UCI Machine Learning Repository [29], secondly, creating three classic deep dense autoencoders [30], and thirdly a convolutional autoencoder for CIFAR-10 dataset, taken from [31].

The two new algorithms, namely ASDM2/b and ASDM2/n, are compared with the following known ones: CM, AG, ADAM, AdaGrad, and AdaDelta. The known algorithms are considered in two settings: with optimized parameters e.g., CM/o, and with default parameters e.g., ADAM/d. The optimized parameters are the momentum decay factor and the step-size selected from the Cartesian product  $\{0.9, 0.99\} \times \{\dots, 0.1, 0.05, 0.02, 0.01, \dots\}$  such that they give the smallest ultimate loss. The default parameters are ones applied by Tensorflow when their values are not provided when calling the algorithm.

### A. Optimization tasks' details

In case of the shallow neural classifiers, we take 10 arbitrary classification problems from the UCI Machine Learning Repository [29]. They are listed in the first part of Tab. I. For each, we build a neural classifier with a single hidden, logistic sigmoidal layer, and a linear output layer. The number of neurons in the output layer is equal to the number of classes, and the number of hidden neurons is roughly optimized in preliminary experiments aiming to minimize the test error. Initial weights of the hidden neurons are drawn from normal distribution  $N(0, \sigma^2)$ , where  $\sigma = 1/\sqrt{\dim(\text{input})}$ . Networks inputs are scaled with their means and standard deviations.

TABLE I  
BASIC PARAMETERS OF ANALYZED LEARNING PROBLEMS. ABR — PROBLEM NAME ABBREVIATION, SIZE — NUMBER OF SAMPLES, IDIM — INPUT DIMENSION, ODIM — OUTPUT DIMENSION, NCNT — HIDDEN LAYER SIZE, MBS — MINI-BATCH SIZE, LEN — NUMBER OF SAMPLES PROCESSED BEFORE TERMINATION (I.E., A RUN TAKES LEN/MBS STEPS).

Problem	Abr	Size	Idim	Odin	Ncnt	Mbs	Len
Credit card defaults	CCard	30000	32	2	21	200	$10^7$
Dota2 games results	Dota2	102944	116	2	55	200	$10^7$
HTRU2	Htru2	17898	8	2	34	200	$10^7$
Sensorless drive	Motor	58509	48	11	89	200	$10^7$
Poker hand	Poker	1025010	10	10	89	200	$10^7$
Robot navigation	Robot	5456	24	4	34	200	$10^7$
Statlog shuttle	Shuttle	58000	9	7	34	200	$10^7$
Skin segmentation	Skin	245057	3	2	21	200	$10^7$
Spambase	Spam	4601	57	2	34	200	$10^7$
Theorem proving	Theo	6118	51	6	89	200	$10^7$
Curves	Curves	20000	784	784	—	200	$10^8$
Handwritten Digits	MNIST	60000	784	784	—	200	$10^8$
Olivetti Faces	Faces	165600	625	625	—	200	$10^8$
CIFAR10 autoencoder	CF10ae	50000	3072	3072	—	200	$2 \cdot 10^7$

Required outputs are one-hot vectors. The loss reported is mean-square error.

Experiments connected with deep autoencoders are based on three datasets containing grayscale images. They are fed to a dense autoencoder neural network. The task for the network is to produce output equal to input. Hidden layers of the networks have the following sizes:

- Curves: 400, 200, 100, 50, 25, 6, 25, 50, 100, 200, 400,
- MNIST: 1000, 500, 250, 30, 250, 500, 1000,
- Faces: 2000, 1000, 500, 30, 500, 1000, 2000.

All layers in the networks are logistic sigmoidal, with the exception of bottleneck layers and the output layer in Faces, which are linear. Sparse weights initialization is applied. Details of the experimental setting are adopted from [32] and [33]. The loss reported is mean-square error.

Images from CIFAR10 dataset are fed to a convolutional autoencoder. All details of the network and training procedure are taken from [31].

Table II shows how optimal step-sizes for CM, AG, ADAM, AdaGrad, and AdaDelta may differ for various problems. The differences reach four orders of magnitude.

### B. Software and computational complexity

Our experimental software has been written in two versions: in C++ with CUDA, and in Python with Tensorflow. The Python version is freely available on <https://github.com/Bestest96/ASDM2-TF>. The most computationally expensive operations in Algorithm 1 are gradient back-propagation performed twice in each loop step and Hessian—vector multiplication, also performed twice. The latter operation is slightly more expensive than the first one. Consequently, ASDM2 needs 3-3.5 times more real time per loop step than CM applied to the same problem.

TABLE II  
STEP-SIZES AND MOMENTUM DECAY FACTORS OPTIMAL FOR EACH PROBLEM. SS — STEP-SIZE, MDF — MOMENTUM DECAY FACTOR.

Alg. Problem	CM		AG		ADAM		AGrad	ADelta	
	ss	mdf	ss	mdf	ss	mdf	ss	ss	mdf
CCard	0.5	0.9	0.02	0.99	0.05	0.9	0.2	1	0.99
Dota2	0.05	0.99	0.05	0.99	0.002	0.99	0.2	1	0.99
Htru2	0.02	0.99	0.05	0.99	0.05	0.9	0.2	1	0.99
Motor	0.02	0.99	0.05	0.99	0.002	0.99	0.05	1	0.9
Poker	0.01	0.99	0.05	0.99	0.005	0.9	0.2	1	0.99
Robot	0.2	0.99	0.1	0.99	0.05	0.9	0.2	1	0.99
Shuttle	0.2	0.99	0.1	0.99	0.02	0.9	0.2	1	0.99
Skin	0.05	0.99	0.1	0.99	0.05	0.9	0.5	1	0.99
Spam	0.2	0.99	0.1	0.99	0.02	0.9	0.2	1	0.99
Theo	0.01	0.99	0.02	0.99	0.01	0.9	0.2	1	0.99
Curves	0.01	0.9	0.002	0.9	0.002	0.9	0.02	0.5	0.99
MNIST	0.02	0.9	0.02	0.9	0.002	0.9	0.02	1	0.99
Faces	0.001	0.99	0.001	0.99	0.0002	0.99	0.01	0.5	0.99
CF10ae	$10^{-5}$	0.99	$10^{-5}$	0.99	0.001	0.9	0.005	0.5	0.99

### C. Results

The results are depicted in Tab. III in the form of average losses attained at the end of training. The table also presents the accuracy of errors in the form of standard deviations. Only training losses are reported, as here we focus on optimization rather than the quality of models that could be demonstrated by test losses. The smallest losses for each problem are indicated by bold face font. The observations made are summarized below.

- 1) In most cases (12 cases out of 14) the winner is either form of ASDM2.
- 2) In most cases (12/14), the manual optimization of the parameters of ADAM yield significant improvement of its behavior. However, its default parameters make that algorithm perform well in comparison to others, except all variants of ASDM2.
- 3) ASDM2/n may be understood as ADAM with  $\beta$  and  $\lambda$  tuned on-the-fly by the method introduced here. In (13/14) cases, ASDM2/n outperformed ADAM/d (with default parameters). In (12/14) cases, ASDM2/n also outperformed ADAM/o (with optimized parameters).
- 4) The performance of AdaGrad and AdaDelta is especially disappointing. Those algorithms were presented as a way to optimize the step-size on-the-fly in SGD and CM, respectively. That does not check out in our experiments.

Table IV presents parameters  $\beta$ ,  $\lambda$ ,  $\gamma$ , and  $\mu$  that ASDM2 was using in the middle of training. Note that these parameters were being constantly adjusted. The  $\beta$  and  $\lambda$  parameters determined by the algorithm may be compared to those optimized manually, which are depicted in Tab. II. Analogies are rather vague, but do exist. ASDM2 adjusts the parameters to the current stage of the learning process, and what is seen in Tabs. IV are points in certain trajectories. The  $\gamma$  and  $\mu$  parameters are close to 1, as was expected.

In general, the ASDM2 algorithm in its various forms yields encouraging results. Usually it reaches its goal, which is

TABLE III  
FINAL LOSS ESTIMATES OBTAINED BY AVERAGING 10 INDEPENDENT RUNS AND REPORTING THE MEAN LOSS VALUE. THE ACCURACY IS DEFINED AS THE STANDARD DEVIATION OF THE SAMPLE MEAN. /D — DEFAULT STEP-SIZE AND MOMENTUM DECAY FACTOR, /O — OPTIMIZED ONES. THE ACCURACY IS PRESENTED WITH THE SAME NUMBER OF DIGITS AFTER THE DECIMAL POINT BUT WITHOUT LEADING ZEROS E.G., 15.35 ±31 DENOTES 15.35 ±0.31, AND 0.0055 ±4 DENOTES 0.0055±0.0004.

Alg. Problem	CM	AG	ADAM		AdaGrad	AdaDelta		ASDM2	
			/d	/o		/d	/o	/b	/n
CCard	0.262 ±1	0.264 ±1	0.262 ±1	0.260 ±1	0.266 ±1	0.342 ±2	0.265 ±1	0.257 ±1	<b>0.256 ±1</b>
Dota2	0.422 ±1	0.418 ±1	0.412 ±1	0.394 ±1	0.405 ±1	0.500 ±1	0.423 ±1	0.414 ±1	<b>0.379 ±1</b>
Htru2	0.0326±7	0.0300±3	0.0313±5	0.0293±5	0.0294±4	0.1033±16	0.0321±5	0.0307±2	<b>0.0291±4</b>
Motor	0.0642±10	0.0480±8	0.0668±9	0.0415±5	0.0829±9	0.802 ±16	0.0956±13	0.0614±24	<b>0.0413±7</b>
Poker	0.451 ±10	0.315 ±13	0.448 ±15	0.327 ±3	0.477 ±7	0.569 ±1	0.524 ±3	0.499 ±6	<b>0.302 ±3</b>
Robot	<b>0.0642±21</b>	0.0908±23	0.1198±14	0.1064±12	0.1270±13	0.6178±19	0.1251±17	0.0843±32	0.0925±44
Shuttle	0.0064±7	0.0045±3	0.0084±3	<b>0.0038±2</b>	0.0093±3	0.2768±41	0.0135±4	0.0100±3	0.0059±2
Skin	0.0062±2	0.0055±1	0.0074±1	0.0043±2	0.0074±2	0.2573±75	0.0089±2	0.0053±1	<b>0.0037±2</b>
Spam	0.0245±10	0.0230±3	0.0279±3	0.0172±4	0.0257±3	0.4270±56	0.0390±5	0.0243±3	<b>0.0171±3</b>
Theo	0.442 ±2	0.409 ±1	0.430 ±2	0.393 ±1	0.440 ±1	0.720 ±1	0.454 ±1	0.408 ±1	<b>0.381 ±2</b>
Curves	0.178 ±2	0.312 ±16	0.158 ±4	0.157 ±3	0.444 ±8	15.930±1	0.303 ±5	0.129 ±1	<b>0.109 ±1</b>
MNIST	1.03 ±1	1.03 ±1	1.27 ±6	1.12 ±1	2.11 ±1	15.35 ±31	1.31 ±1	<b>0.90 ±1</b>	0.92 ±1
Faces	15.99 ±3	16.81 ±38	15.32 ±25	14.61 ±2	39.63 ±32	103.26±9	19.73 ±3	<b>14.45 ±5</b>	15.76 ±8
CF10ae	5.11 ±12	3.75 ±8	2.29 ±2	2.29 ±2	4.98 ±9	23.75 ±7	2.69 ±4	3.73 ±10	<b>2.22 ±2</b>

TABLE IV  
AVERAGE (OVER TRAINING RUNS) VALUES OF  $\beta$ ,  $\lambda$ ,  $\gamma$ , AND  $\mu$  IN THE MIDDLE OF TRAINING.

Alg. Problem	ASDM2/b				ASDM2/n			
	$\beta$	$\lambda$	$\gamma$	$\mu$	$\beta$	$\lambda$	$\gamma$	$\mu$
CCard	0.33	0.92	1	0.997	0.0024	0.81	1	0.997
Dota2	0.24	0.93	0.9999	0.998	0.0024	0.74	0.9994	0.9994
Htru2	0.19	0.96	0.9998	0.995	0.0025	0.89	0.9999	0.995
Motor	0.11	0.93	0.998	0.995	0.0011	0.76	0.995	0.996
Poker	0.076	0.93	1	0.997	0.00022	0.97	0.9997	0.996
Robot	0.26	0.97	0.9998	0.997	0.0033	0.84	1	0.996
Shuttle	0.20	0.86	0.9999	0.990	0.00048	0.89	0.998	0.985
Skin	0.31	0.98	0.9992	0.993	0.0028	0.96	0.9994	0.994
Spam	0.22	0.97	0.9996	0.997	0.0019	0.78	0.9999	0.996
Theo	0.11	0.94	1	0.997	0.0013	0.71	0.9992	0.996
Curves	0.001	0.996	0.9995	0.998	3.2e-5	0.97	0.9998	0.9991
MNIST	0.016	0.91	0.9999	0.9995	7.5e-5	0.87	1	0.9994
Faces	0.004	0.94	1	0.9996	3.4e-5	0.84	0.9998	0.9996
CF10ae	4.2e-5	0.94	0.997	0.97	2.5e-5	0.98	0.994	0.998

approximate optimization on-the-fly of the  $\beta$  and  $\lambda$  parameters for CM, AG, and ADAM. Consequently, ASDM2 yields good speed of learning from the first run. However, in some cases, the algorithm does not provide optimal  $\beta$  or  $\lambda$ , and those cases are especially interesting. Such cases are Robot and Shuttle.

What is conspicuous about those problematic cases in Tab. IV are the relatively small values of  $\gamma$ . While for the sake of numeric accuracy (see Sec. V-C), it does not make sense to set  $\gamma$  equal to 1, there is a systematic reason why the larger that parameter is, the better. Rough interpretation of the terms  $\mathcal{S}_\gamma dJ(\theta_t, \xi_t)/d\beta_{t-1}$  and  $\mathcal{S}_\gamma dJ(\theta_t, \xi_t)/d\lambda_{t-1}$  is as follows: ASDM2 at each  $t$  looks at what would happen with  $J(\theta_t, \xi_t)$  if  $\beta_t$  and  $\lambda_t$  had been larger in preceding  $(1-\gamma)^{-1}$  steps. Therefore, if the algorithm is forced to set small  $\gamma$ , it has only a myopic view on the influence of  $\beta$  and  $\lambda$  on the learning process. That, in turn, leads to suboptimal values of  $\beta$  and  $\lambda$ . How to enable large  $\gamma$  for all problems and learning

stages is a curious research topic.

## VII. CONCLUSIONS

The manual tuning of hyperparameters in on-line learning slows research down since it requires a whole process to be repeated many times. It also makes many potential applications of machine learning unavailable, since in most cases one can not tell the user to go pick the right hyperparameters by trial and error.

In this paper, a step has been made to get rid of “pesky” hyperparameters such as step-size and momentum decay factor, and to be able to get results of on-line learning after a single run. A method was introduced that adjusts those hyperparameters in CM and ADAM. The method is based on the recognition of the short- and long-term influence of the hyperparameters on the learning process. The hyperparameters are tuned to make the process fast yet stable. The method does not depend on any preliminary knowledge of the learning problem, thereby making on-line learning a process that needs to be run only once. In the experimental study, the method was applied to shallow neural networks, as well as deep auto-encoders, and in most cases it performed better than any manually selected hyperparameters.

## ACKNOWLEDGMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

## REFERENCES

- [1] H. Robbins and S. Monro, “A stochastic approximation method,” *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.
- [2] B. T. Polyak, “Some methods of speeding up the convergence of iteration methods,” *USSR Computational*

- Mathematics and Mathematical Physics*, vol. 4, pp. 1–17, 1964.
- [3] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- [4] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *CoRR*, vol. abs/1412.6980, 2014.
- [5] P. Wawrzynski, “ASD+M: Automatic parameter tuning in stochastic optimization and on-line learning,” *Neural Networks*, vol. 96, pp. 1–10, 2017.
- [6] J. Kiefer, J. Wolfowitz *et al.*, “Stochastic estimation of the maximum of a regression function,” *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.
- [7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, no. 323, pp. 533–536, 1986.
- [8] Y. Nesterov, “A method of solving a convex programming problem with convergence rate  $o(1/\sqrt{k})$ ,” *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.
- [9] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks*, vol. 12, 1999.
- [10] A. Bhaya and E. Kaszkurewicz, “Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method,” *Neural Networks*, vol. 17, pp. 65–71, 2004.
- [11] T. A. Straeter, “On the extension of the davidon-broyden class of rank one, quasi-newton minimization methods to an infinite dimensional hilbert space with applications to optimal control problems,” 1971.
- [12] G. Qiu, M. Varley, and T. Terrell, “Accelerated training of backpropagation networks by using adaptive momentum step,” *Electronics letters*, vol. 28, no. 4, pp. 377–379, 1992.
- [13] D. Swanson, J. Bishop, and R. J. Mitchell, “Simple adaptive momentum: new algorithm for training multi-layer perceptrons,” *Electronics Letters*, vol. 30, no. 18, pp. 1498–1500, 1994.
- [14] T. Graepel and N. N. Schraudolph, “Stable adaptive momentum for rapid online learning in nonlinear systems,” in *International Conference on Artificial Neural Networks*. Springer, 2002, pp. 450–455.
- [15] M. Z. Rehman and N. M. Nawi, “The effect of adaptive momentum in improving the accuracy of gradient descent back propagation algorithm on classification problems,” in *International Conference on Software Engineering and Computer Systems*. Springer, 2011, pp. 380–390.
- [16] A. A. Hameed, B. Karlik, and M. S. Salman, “Back-propagation algorithm with variable adaptive momentum,” *Knowledge-Based Systems*, vol. 114, pp. 79–87, 2016.
- [17] J. Zhang and I. Mitliagkas, “Yellowfin and the art of momentum tuning,” *arXiv preprint arXiv:1706.03471*, 2017.
- [18] M. D. Zeiler, “Adadelta: An adaptive learning rate method,” in *arXiv:1212.5701*, 2012.
- [19] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” 2012.
- [20] T. Dozat, “Incorporating Nesterov momentum into Adam,” in *ICLR*, 2016.
- [21] L. Luo, Y. Xiong, and Y. Liu, “Adaptive gradient methods with dynamic bound of learning rate,” in *International Conference on Learning Representations*, 2019. [Online]. Available: <https://openreview.net/forum?id=Bkg3g2R9FX>
- [22] L. Luo, Y. Xiong, Y. Liu, and X. Sun, “Adaptive gradient methods with dynamic bound of learning rate,” *arXiv preprint arXiv:1902.09843*, 2019.
- [23] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The marginal value of adaptive gradient methods in machine learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4148–4158.
- [24] N. Schraudolph and T. Graepel, “Towards stochastic conjugate gradient methods,” in *Proceedings of the 9th International Conference on Neural Information Processing*, 2002, pp. 853–856.
- [25] N. N. Schraudolph, “Local gain adaptation in stochastic gradient descent,” 1999.
- [26] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, “Online learning rate adaptation with hypergradient descent,” *arXiv preprint arXiv:1703.04782*, 2017.
- [27] M. Rolinek and G. Martius, “L4: Practical loss-based stepsize adaptation for deep learning,” in *Advances in Neural Information Processing Systems*, 2018, pp. 6433–6443.
- [28] Y. Wu, M. Ren, R. Liao, and R. Grosse., “Understanding short-horizon bias in stochastic meta-optimization,” in *International Conference on Learning Representations*, 2018. [Online]. Available: <https://openreview.net/forum?id=H1MczcgR->
- [29] A. Frank and A. Asuncion, “UCI machine learning repository,” 2010. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [30] A. P. George and W. B. Powell, “Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming,” *Machine Learning*, vol. 65, pp. 167–198, 2006.
- [31] P. Wawrzynski, “Efficient on-line learning with diagonal approximation of loss function hessian,” in *IJCNN*, 2019.
- [32] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, pp. 504–507, 2006.
- [33] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *ICML*, vol. 28, no. 3. JMLR Workshop and Conference Proceedings, 2013, pp. 1139–1147.

## **B.4. Reinforcement Learning for on-line Sequence Transformation**

Title	Reinforcement Learning for on-line Sequence Transformation
Authors	Grzegorz Rypeść, Łukasz Lepak, Paweł Wawrzyński
Conference	17th Conference on Computer Science and Intelligence Systems (FedCSIS 2022)
Year	2022
DOI	10.15439/2022F70
Ministerial score	70

# Reinforcement Learning for on-line Sequence Transformation

Grzegorz Rypeść, Łukasz Lepak, Paweł Wawrzyński

Warsaw University of Technology, Institute of Computer Science, Warsaw, Poland  
{grzegorz.rypec.stud, lukasz.lepak.dokt, pawel.wawrzyński}@pw.edu.pl

**Abstract**—In simultaneous machine translation (SMT), an output sequence should be produced as soon as possible, without reading the whole input sequence. This requirement creates a trade-off between translation delay and quality because less context may be known during translation. In most SMT methods, this trade-off is controlled with parameters whose values need to be tuned. In this paper, we introduce an SMT system that learns with reinforcement and is able to find the optimal delay in training. We conduct experiments on Tatoeba and IWSLT2014 datasets against state-of-the-art translation architectures. Our method achieves comparable results on the former dataset, with better results on long sentences and worse but comparable results on the latter dataset.

## I. INTRODUCTION

**S**IMULTANEOUS machine translation (SMT) can be defined as producing output sequence tokens while reading input sequence tokens in an on-line fashion. These tokens may represent words in given languages, chunks of audio streams, or any other sequential data. The main difference between SMT and more general neural machine translation (NMT) is how the input and output sequences are processed. Most NMT methods read all input tokens and then generate the output sequence. Because of this, even though efficient state-of-the-art NMT methods exist, they cannot be used in SMT applications. Also, SMT methods need to consider the trade-off between delay and quality, as faster translation implies less context from the input. In most cases, this trade-off has to be optimized by checking various parameter settings, which is resource- and time-consuming.

SMT can be decomposed into a sequence of readings of the input tokens and writings of the output tokens. Reinforcement learning (RL) [1] is often applied to train SMT systems that sequentially choose between these actions and/or choose the written token. In this paper, we present an RL-based method with self-learning delay. Unlike in other approaches, we apply bootstrapping in training, which means that the sequences translated can be in principle infinite. We conduct experiments on Tatoeba and IWSLT2014 datasets against state-of-the-art translation architectures. Our method achieves comparable results on the former dataset, with better results on long sentences and worse but comparable results on the latter dataset.

The paper is organized as follows. Section II overviews literature related to neural machine translation, reinforcement learning, and simultaneous machine translation. Section III formally defines the problem considered in this paper. Section IV presents our method. Section V describes simulations

evaluating the presented architecture. Section VI discusses the experimental results and limitations of our approach. Section VII concludes the paper.

## II. RELATED WORK

*a) Neural machine translation (NMT):* A basic architecture for neural machine translation includes an encoder that is fed with the input sequence; its final state becomes the initial state of a decoder that produces the output sequence [2]. In order to produce the right output, attention must be paid to significant input tokens. Attention was introduced to the encoder-decoder architecture in [3] and [4]. An architecture for NMT that is based solely on attention is Transformer [5]. Recurrent neural networks (RNN) were applied to capture short-term dependencies in input sequences and combined with multilayer attention in R-Transformer [6]. However, all these architectures only produce output when given the whole input sequence and hence are not applicable to on-line translation.

*b) Reinforcement learning (RL):* RL is a general framework for adaptation in the context of sequential decision making under uncertainty [1]. In this framework, an agent operates in discrete time, at each instant observing the state of its environment and taking action. Subsequently, the environment state changes, and the agent receives a numeric reward. Both the next state and the reward result from the previous state and action. By repeatedly facing the sequential decision problem in the same environment, the agent learns to designate actions in current environment states to be able to expect the highest future rewards.

In the context of this paper, especially interesting is the case where the agent cannot observe its state but only the value of a certain function of the state. This case is modeled as the Partially Observable Markov Decision Process (POMDP) [7]. In this model, the agent needs to collect subsequent observations to be able to recognize its current situation at any specific time. This can be done effectively with an RNN. Deep Recurrent Q-Learning [8] is an RL method for POMDP, which applies an RNN for that purpose.

RL has been applied to neural machine translation to optimize a policy, which, given an input sentence, assigned maximum probability to the corresponding output sentence. RL was applied this way to optimize the translation quality expressed in BLEU [9] which is not directly differentiable. RL has also been applied to train a random generator of sentences in

a generative adversarial architecture [10]. A similar architecture has been applied for the sequential generation of graphs [11].

c) *Simultaneous Machine Translation (SMT)*: A number of SMT methods use reinforcement learning. One of the first examples of using RL for SMT was presented in [12]. It uses imitation learning from the optimal sequence of actions to learn a policy for the system. In [13], a two-action framework was introduced, where the agent can read an input token, named READ, or write a new output token, named WRITE. This framework serves as a baseline for many new SMT methods, with authors extending and modifying it to achieve better results. The proposed reward function is based on the achieved BLEU score [14] and the translation delay metrics proposed by the authors, with the trade-off between delay and translation quality controlled by setting appropriate parameter values. In [15], a third action was added, named PREDICT, which works similarly to READ, but instead of reading an input token, it predicts this token. The reward function was also changed to include predictions' quality, with delay-quality trade-off still controlled by parameters. In [16], a commonly used NMT encoder-decoder structure was modified to work with SMT by making encoder, and attention dynamically change after every READ and adding an incremental decoder, which outputs a token from them after every WRITE. In [17], a method was proposed for extracting action sequences from NMT architectures, which were later used with sentence pairs in imitation learning to learn an optimal policy. Recently, reinforcement learning was used in multimodal translation [18], utilizing text and visual data to improve the quality of translations.

Not every SMT method uses reinforcement learning. In [19], the "wait- $k$ " strategy was proposed, which produces a new output token with a fixed delay equal to  $k$ . It can be easily implemented in commonly used NMT architectures, shown by modifying the original Transformer. In [20], the "wait- $k$ " strategy was used in speech-to-text task, showing it is efficient in applications other than machine translation.

### III. PROBLEM DEFINITION

We consider input sequences,  $x = (x_i)_{i=0}^{|x|-1}$ , that contain tokens,  $x_i \in \mathbb{R}^d$ ,  $d \in \mathbb{N}$ . The input sequences correspond to target sequences,  $y = (y_j)_{j=0}^{|y|-1}$ ,  $y_j \in \mathbb{R}^{d'}$ ,  $d' \in \mathbb{N}$ . The sequences are of variable lengths presented by the  $|\cdot|$  function. An *interpreter agent* is fed with subsequent tokens from  $x$  and produces tokens of an output sequence,  $(z_j)_{j=0}^{|z|-1}$ ,  $z_j \in \mathbb{R}^{d'}$  on the basis of  $x$ .

Three special tokens playing various roles exist in both the input and the output space. They are:

- NULL — a missing element,
- EOS — denotes the last element of each sequence,
- PAD — an element concatenated to sequences after EOS for technical reasons.

For brevity, we will assume  $x_i = \text{PAD}$ ,  $y_j = \text{PAD}$ ,  $z_j = \text{PAD}$  for, respectively,  $i \geq |x|$ ,  $j \geq |y|$ ,  $j \geq |z|$ .

Given  $x$ , the agent should produce  $z$  that minimizes the quality index in the form

$$J(y, z) = \sum_{j=0}^{K-1} L(y_j, z_j). \quad (1)$$

The loss  $L$  penalizes mistranslation;  $L(\text{PAD}, \text{PAD}) = 0$ ;  $K$  is a number larger than any  $|y|$ . The sequence  $z$  that minimizes (1) is of length  $|y|$ , contains tokens equal to those in  $y$ , and ends with EOS.

We also require the interpreter agent to be of limited capacity but handle sequences of arbitrarily large lengths. In other words, we require the agent to operate on-line, i.e., it is fed with subsequent tokens of the input sequence and simultaneously produces subsequent tokens of the output sequence.

## IV. METHOD

### A. Reinforcement learning to transform sequences

We formalize the transformation of one sequence into another as an iterative decision process. At each of its instants, an agent reads a subsequent token from the input sequence or writes a subsequent token of the output sequence, similarly to [13]. That is, at each instant, the agent executes one of two actions:

- READ — another input token is read. This action is useful when it is (still) unclear what output token should be produced.
- WRITE — a subsequent output token is produced. This action is useful when a certain comprehensive portion of input tokens have been read, and a subsequent part of its interpretation can be presented.

A *policy* is a method of selecting actions and producing output tokens based on tokens read and those produced so far.

After execution of some of the actions, the agent receives numerical *rewards*. Let the rewards received during the process be denoted by  $r = (r_k)_{k=0}^{|r|-1}$ . A reward,  $r_k$ , is emitted at the following times:

- An output token,  $z_j$ , has just been written. Then  $r_k$  is the negative cost of mistranslation, i.e.

$$r_k = -L(y_j, z_j). \quad (2)$$

- A whole input sequence has been read, and the READ action is taken. This action does not make sense at this time. Therefore, for a certain constant  $M > 0$ , we have

$$r_k = -M. \quad (3)$$

Let  $n(t)$  be the number of rewards emitted before the  $t$ -th action. The quality criterion for the policy is maximization of future discounted rewards. That is, at each time  $t$  the expected value of the *return*

$$R_t = \sum_{k=n(t)}^{|r|-1} \gamma^{k-n(t)} r_k \quad (4)$$

should be maximized, where  $\gamma \in (0, 1)$  is a discount factor. In one episode of its operation, the agent transforms a single

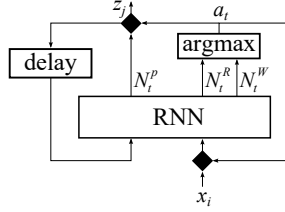


Fig. 1: Proposed architecture for on-line sequence transformation. The black squares represent passing/delaying  $x_i$  and outputting/skipping  $N_t^p$  depending on the action  $a_t$ .

sequence. It stops producing additional output tokens when it has outputted EOS.

In training, the agent, not having learned how to finish sequences, must be prevented from producing them infinitely long. Here we assume that an episode of training is terminated when the agent has produced as many tokens as in the target sequence  $y$ . The last target token is EOS, which is enough for the agent to learn to finish the output sequences.

Usually, in reinforcement learning [1] a reward comes after each action. However, here we want the agent to be rewarded only for the tokens it produces, bearing in mind that it does not produce them with READ actions. Rewards equal to zero for such actions do not make sense here because they could encourage the agent to maximize the sum of discounted rewards by postponing the production of output. Therefore, here we admit actions that are not immediately followed by rewards. Those emitted rewards have their own indices and are discounted according to them.

At each instant of its operation, a *state* of the agent's environment consists of the tokens the agent has read so far and the tokens it has written so far. However, before taking another action, it is only fed with the next input token and with the last written token. Therefore, the agent's environment is partially observable.

### B. Architecture

We propose an architecture that learns to make the actions discussed above. The policy has the form of a recurrent neural network. Its input size is  $d + d'$ . In an instant of its operation it is fed with a subsequent input token concatenated with a preceding output token. Specifically, the first input to the network is the pair  $(x_0, \text{NULL})$ . Let us assume that the agent has already read  $i$  input tokens and produced  $j$  output tokens. Thus, after the READ action, the network input is  $(x_i, \text{NULL})$ . After the WRITE action, the network input is  $(\text{NULL}, z_{j-1})$ .

In training *teacher forcing* can also be applied: The agent is fed not with the tokens it has already outputted but with target tokens.

Output of the network is of size  $d' + 2$ . The network produces a  $d'$ -dimensional *potential output token* and 2 scalar *return*

*estimates* that approximate returns (4) expected if actions WRITE and READ, respectively, were taken.

Let  $N_t$  be the  $d' + 2$ -dimensional output of the network at  $t$ -th instant. It is composed as

$$N_t = [N_t^p, N_t^W, N_t^R],$$

where  $N_t^p \in \mathbb{R}^{d'}$  is the potential output token, and  $N_t^W, N_t^R \in \mathbb{R}$  are the return estimates for the WRITE and READ actions, respectively. The architecture is depicted in Fig. 1.

The network output that estimates the return corresponding to the just taken action  $a_t$  is trained to approximate the conditional expected value

$$Q_t(a_t) = E(R_t | C_t), \quad (5)$$

where the condition  $C_t$  includes the following:

- 1) The action just taken is  $a_t$ .
- 2) Subsequently, those actions are selected, which correspond to the network return estimates with maximum values.
- 3) Input tokens read so far, and output tokens produced so far. At the time  $t$ , the rest of the tokens are unknown, thereby remaining random vectors.

The actions actually taken are usually selected as those maximizing the return estimates given by the network. However, with a small probability, the agent chooses the other action since it needs to explore different actions to learn their consequences. Therefore, we will not estimate  $Q_t(a_t)$  based on the actual return, but on a recursion instead. Specifically, let us denote by  $j$  the number of output tokens produced before the analyzed action is taken. A simple analysis reveals that  $Q_t(a_t)$  (5) satisfies the following recursive equation:

$$Q_t(a_t) = E \left\{ \begin{array}{l} -M + \gamma \max_b Q_{t+1}(b) \\ \quad \text{if } a_t = \text{READ}, \text{fin}(x) \\ \max_b Q_{t+1}(b) \\ \quad \text{if } a_t = \text{READ}, \neg \text{fin}(x) \\ -L(y_j, z_j) + \gamma \max_b Q_{t+1}(b) \\ \quad \text{if } a_t = \text{WRITE}, j < |y| - 1 \\ -L(y_j, z_j) \\ \quad \text{if } a_t = \text{WRITE}, j = |y| - 1 \end{array} \middle| a_t \right\} \quad (6)$$

where  $\text{fin}(x)$  means that all  $x$  tokens have been read. The condition for the above expectation is that the action actually taken is  $a_t$ .

Target values for the network will be based on the above recursive equation and the fact that  $Q_{t+1}(\text{READ})$  and  $Q_{t+1}(\text{WRITE})$  are estimated by  $N_{t+1}^R$  and  $N_{t+1}^W$ , respectively. Therefore, the network outputs at time  $t$  are trained as follows.<sup>1</sup> After the READ action, when  $x$  is not finished yet,  $N_t^R$  is adjusted:

$$N_t^R \leftarrow \max\{N_{t+1}^R, N_{t+1}^W\}. \quad (7)$$

<sup>1</sup>We apply the notation:

$$[\text{predicate}] = \begin{cases} 1 & \text{if predicate is true} \\ 0 & \text{otherwise.} \end{cases}$$



After the READ action, when  $x$  is already finished:

$$N_t^R \leftarrow -M + \gamma \max\{N_{t+1}^R, N_{t+1}^W\}. \quad (8)$$

After a WRITE action, the return estimate for the WRITE action is adjusted as

$$N_t^W \leftarrow -L(N^P, z_j) + [j < |y| - 1] \gamma \max\{N_{t+1}^R, N_{t+1}^W\}, \quad (9)$$

Also, the potential output is adjusted

$$N_t^P \leftarrow y_j. \quad (10)$$

### C. Weighting losses due to mistranslation and return estimation

The network produces outputs of two qualitatively different kinds: the potential output tokens and the return estimates. The network training requires minimization of an aggregated loss that combines a loss due to mistranslation and a loss due to return estimation. We propose to normalize these losses with their averages defined below.

Let  $n = 1, 2, \dots$  be a training minibatch index. We average original mistranslation losses,  $L_n^M$ , and original estimation losses,  $L_n^E$ , according to

$$\bar{L}_n^M = w_n \bar{L}_{n-1}^M + (1 - w_n) L_n^M, \quad (11)$$

$$\bar{L}_n^E = w_n \bar{L}_{n-1}^E + (1 - w_n) L_n^E \quad (12)$$

where  $\bar{L}_0^M = \bar{L}_0^E = 0$ , and

$$w_n = \rho(1 - \rho^{n-1}) / (1 - \rho^n), \quad (13)$$

where  $\rho \in (0, 1)$  is the decay factor, e.g.  $\rho = 0.99$ . The terms (11,12) approximate arithmetic means for small  $n$ , and exponential moving average for larger  $n$ .

Training the network aims at minimizing the aggregated loss in the form

$$L_n = L_n^M / \bar{L}_n^M + \eta(n, n_0) L_n^E / \bar{L}_n^E. \quad (14)$$

The term  $\eta(n, n_0)$  is a relative weight of the estimation loss for the current minibatch/epoch index  $n$  and the (expected) total number  $n_0$  of minibatches/epochs in the whole training. For small  $n$  this weight should be small:  $\eta(n, n_0) \approx \eta_{min}$ , since high accuracy of future rewards is pointless when quality of outputted tokens is poor.  $\eta(n, n_0)$  is gradually growing with  $n$  to a certain asymptote,  $\eta_{max}$ .  $\eta_{min}$  and  $\eta_{max}$  are hyperparameters of the training process, e.g. 1/50 and 1/5, respectively. The  $\eta$  function may have the form

$$\eta(n, n_0) = \eta_{max} - (\eta_{max} - \eta_{min}) \exp(-3n/n_0). \quad (15)$$

## V. EXPERIMENTAL STUDY

In this section, we demonstrate the effectiveness of our proposed architecture, henceforth called RLST (Reinforcement Learning for on-line Sequence Transformation). We perform experiments with seven machine translation tasks. They are based on datasets taken from Tatoeba [21] and dataset taken from IWSLT2014 [22].

In our machine translation tasks, the input sequence consists of tokens representing words of a sentence in a source language. The aim is to generate a sequence of tokens with

the same sentence meaning as the source sequence. We conduct experiments on datasets presented in Table I which contains basic statistics on the source and target languages datasets, sizes of source and target dictionaries, and numbers of sentences in each data split. For Tatoeba datasets, we also separate long test splits, where source sentences have more than 22 tokens. The long test split allows us to compare how models deal with longer input sentences. For all datasets, we compare our proposed RLST architecture with state-of-the-art machine translation architectures, namely encoder-decoder with attention [3] and Transformer [5]. For both encoder-decoder and Transformer, the minimized loss is cross-entropy. For RLST, we quantify  $L_n^M$  and  $L_n^E$  in (14) as cross-entropy and mean square error, respectively.

In our experiments, we employed the following procedure to optimize hyperparameters of the compared architectures. For Tatoeba datasets, we optimized the hyperparameters manually for all three architectures to obtain their best BLEU score [14] on the En-Es language pair and applied these values to all language pairs. For IWSLT2014 datasets, we optimized the hyperparameters of RLST manually and took the hyperparameters for the Transformer from [5] and for the encoder-decoder with attention from [23].

Our simulation experiments have been performed on a PC equipped with AMD Ryzen™ Threadripper™ 1920X, 64GB RAM, 4xNVIDIA™ GeForce™ RTX 2070 Super™.

### A. Tatoeba

Tatoeba datasets [21] contain various, mostly unrelated, sentences and their translations provided by the community. We preprocess them using spaCy tokenizer [24] and replace tokens that appear in training corpora less than three times with a unique token representing an unknown word. We also remove duplicated source sentences.

Experiments on Tatoeba for all architectures are run for 50 epochs, with a batch size of 128 and gradient clipping norm set to 10.0. Encoder-decoder and RLST have weight decay set to  $10^{-5}$ , while Transformer has weight decay set to  $10^{-4}$ . Source and target tokens are converted to trainable vectors of length 256 initialized with  $\mathcal{N}(0, 1)$ . There is a dropout applied to them with a probability of 0.2. We use Adam optimizer with default parameters and a constant learning rate equal to 0.0003. The reference encoder-decoder is presented in [3]. Its encoder is a bidirectional GRU recurrent layer with 256 hidden neurons followed by a linear attention layer with 64 neurons. The decoder is a GRU recurrent layer with 256 hidden neurons followed by a dropout with 0.5 probability and a linear output layer with a number of neurons equal to the target's vocabulary size. The teacher forcing ratio for encoder-decoder during training is set to 1.0. For the Transformer, we use the following parameters: the number of expected features in the encoder and decoder inputs is 256, the number of heads in multiattention is 8, the number of encoder and decoder layers is 6, the dimension of feedforward layers is 512, the dropout probability is 0.25 and the teacher forcing ratio to 1.0. For RLST, we use the following approximator. Input and

Dataset	Abbr	Src. dict.	Trg. dict.	Train set	Valid. set	Test set	Long test
Tatoeba Spanish-English	Tat Es-En	13 288	8 960	124 179	41 393	41 394	2 387
Tatoeba French-English	Tat Fr-En	13 792	10 056	161 283	53 761	53 762	2 613
Tatoeba English-Spanish	Tat En-Es	8 690	12 698	115 026	38 342	38 342	2 325
Tatoeba English-Russian	Tat En-Ru	10 009	21 820	241 785	80 595	80 595	1 756
Tatoeba English-German	Tat En-De	10 504	15 276	170 347	56 782	56 783	3 805
IWSLT2014-German-English	De-En	8 848	6 632	160 239	7 283	6 750	—
IWSLT2014-English-German	En-De	6 632	8 848	160 239	7 283	6 750	—

TABLE I: Basic statistics of machine translation datasets.

previous output embeddings with a dimension of 256 are passed to a dense layer with 512 neurons, Leaky ReLU activation with negative slope set to 0.01 and dropout probability of 0.2. Its output is processed by four GRU layers with the hidden dimension of 512 and residual connections between them. The output of the last recurrent layer is passed to a dense layer with 512 neurons, Leaky ReLU activation with a negative slope set to 0.01, and a dropout probability of 0.5. The output of the last dense layer is passed to the output linear layer with number of neurons equal to the target’s vocabulary size and additional 2 neurons representing Q-values of actions. We also set  $\gamma = 0.9$ ,  $\varepsilon = 0.3$ ,  $M = 3$ ,  $N = 50000$ ,  $\eta_{min} = 0.02$ ,  $\eta_{max} = 0.2$ ,  $\rho = 0.99$  and teacher forcing ratio to 1.0.

#### B. IWSLT2014

We conduct experiments on IWSLT2014 German-English and English-German datasets using the *fairseq* framework [22]. Data is preprocessed using the script provided by the benchmark, which utilizes byte-pair encoding (BPE) [25]. For every architecture, the training lasts for 100 epochs with varying batch sizes to ensure that the maximum number of tokens in a batch equals 4096 and the gradient clipping norm is set to 10.0. The encoder-decoder and RLST architectures are trained using Adam optimizer with default parameters and constant learning rate scheduling with weight decay of  $10^{-5}$ . The Transformer is also trained using Adam optimizer, with parameters and a learning rate scheduler described in [5]. For the encoder-decoder and the Transformer, we use the *lstm\_wiseman\_iwslt\_de\_en* architecture (based on [23]) and *transformer\_iwslt\_de\_en* (based on [5] with some changes), respectively. The encoder-decoder model has trainable source and target embeddings dimensions of 256 without dropout. Its encoder is an LSTM layer with 256 hidden neurons, and its decoder was also an LSTM layer with 256 hidden neurons followed by an output layer with the number of neurons equal to the target’s vocabulary size. The decoder uses the attention mechanism. The encoder and decoder layers have a dropout probability of 0.1. The Transformer has the following parameters: The trainable source and target embeddings dimensions are 512 without dropout, the number of neurons in feedforward layers is 1024, the number of multiattention heads is 4, the number of encoder and decoder layers is 6 and a dropout probability of 0.1. For RLST, we set trainable source and target embedding dimensions to 256 with a dropout of 0.2 probability. In the case of IWSLT-En-De, we

use the same approximator as in Tatoeba. For IWSLT-De-En, we changed the dimensions of dense and GRU layers from 512 to 768. We also set  $\gamma = 0.9$ ,  $\varepsilon = 0.30$ ,  $M = 7$ ,  $N = 100000$ ,  $\rho = 0.99$ ,  $\eta_{min} = 0.02$ ,  $\eta_{max} = 0.2$  and teacher forcing ratio to 1.0. For encoder-decoder and Transformer, we set beam search width to 1 and teacher forcing ratio to 1.0.

#### C. Results

The results are presented in Table II. For each dataset and architecture, we show BLEU values computed on a test split from checkpoints for which the BLEU value on a validation split was the highest. We also show the number of parameters for each model. The highest values of BLEU for each dataset are bolded. On the Tatoeba test confined to sentences of length up to 22 words, all three architectures achieved similar BLEU. However, in the test confined to longer sentences, RLST outperforms the other architectures in 4 language pairs out of 5, usually by a large margin. The architecture to achieve the best results on fairseq datasets is the Transformer. RLST and the encoder-decoder with attention achieve similar BLEU on this benchmark.

In order to gain an additional insight into the operation of the RLST interpreter agent we present in Figure 2 the timing of taking the READ and WRITE actions. As one may expect, initially, the agent is mostly reading, then reading and writing ratios are roughly equal, and finally, the agent is mostly writing. It appears that the agent has read about five more words than it has written for most of the time. That seems to correspond to a common intuition: A human interpreter also needs to be delayed a few words in producing an accurate translation of a speech.

## VI. DISCUSSION

Our proposed interpreter agent RLST is designed to transform arbitrarily long sequences on-line. In each cycle of its operation, it performs the same number of computations in which it reads an input token or writes an output token. The agent has only limited memory space to store information about recently read tokens, a context of these tokens defined by previous ones, and recently written tokens. Therefore, the agent is not a method of choice for translating sentences of moderate length without any context.

The RLST architecture outperformed others in the test on long sentences (longer than 22 words) taken from Tatoeba. The memory state of the interpreter agent preserved the context

Architecture → Dataset ↓	Encoder-decoder		Transformer		RLST	
	BLEU	Num. params	BLEU	Num. params	BLEU	Num. params
Tat Es-En	<b>50.33</b>	16 637 504	50.19	15 906 560	50.02	17 122 050
Tat Es-En (L)	16.52	16 637 504	13.42	15 906 560	<b>20.57</b>	17 122 050
Tat Fr-En	<b>53.95</b>	18 170 504	53.89	16 597 832	53.05	18 093 898
Tat Fr-En (L)	13.49	18 170 504	10.03	16 597 832	<b>16.42</b>	18 093 898
Tat En-Es	<b>45.14</b>	20 248 794	44.63	16 647 066	45.09	18 819 484
Tat En-Es (L)	16.1	20 248 794	12.39	16 647 066	<b>21.07</b>	18 819 484
Tat En-Ru	<b>47.71</b>	32 271 740	47.11	21 664 316	47.37	26 171 966
Tat En-Ru (L)	10.06	32 271 740	5.66	21 664 316	<b>11.28</b>	26 171 966
Tat En-De	<b>41.98</b>	24 015 596	41.63	18 433 964	40.62	21 266 350
Tat En-De (L)	<b>10.95</b>	24 015 596	9.5	18 433 964	10.2	21 266 350
IWSLT De-En	24.13	7 178 728	<b>32.17</b>	42 864 640	23.28	24 223 210
IWSLT En-De	19.01	7 748 240	<b>26.13</b>	43 999 232	18.32	15 331 986

TABLE II: BLEU scores on test splits and number of parameters for tested architectures. (L) on Tatoeba datasets denotes scores from long test split.

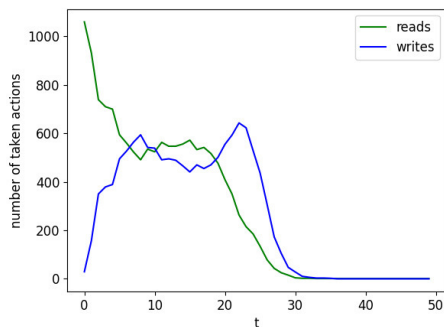


Fig. 2: Processing of 1089 source sentences of length 15 from the Tat En-Ru test dataset by the RLST interpreter agent. The graph shows when the READ and WRITE actions are taken.

of the outputted words better than the attention mechanism managed to do in the reference architectures. We hypothesize that the sequential nature of human language makes it possible to translate properly separate parts of a speech, but in order to do that, the end of each part must be identified. It appears that RLST manages to do it better in long sentences than the reference architectures.

The goal of a large fraction of algorithms developed in computer science is to transform input data into output data whose size is unknown in advance. For some data types, it is natural to process them sequentially. These types include natural language, sound, video, and bioinformatic data, e.g., genetic. The experiments in Section V confirm that our introduced RLST architecture is very well adapted to such data.

## VII. CONCLUSIONS

In this paper, we have presented the RLST architecture that transforms on-line sequences of arbitrary length without the need to define the trade-off between delay and quality. In the transformation process, it makes sequential decisions about

whether to read an input token or write an output token. The architecture learns to make these decisions with reinforcement. The experimental study compared the architecture with state-of-the-art machine translation methods, namely the Transformer and the encoder-decoder with attention. Benchmark datasets taken from Tatoeba and IWSLT with seven language pairs were employed in the experiments. The RLST architecture solved a more complex problem of on-line transformation than the reference methods, which produced output tokens knowing the entire source sequence. Even so, RLST produced translations of comparable quality. It also outperformed reference architectures in tests with long sentences (longer than 22 words) taken from Tatoeba. That confirms that it is particularly well suited to applications in which transformation of sequences of arbitrary lengths and/or on-line is required.

## ACKNOWLEDGMENTS

The project was funded by POB Research Centre for Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative Program – Research University (ID-UB).

## REFERENCES

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. doi: <https://dx.doi.org/10.1109/TNN.1998.712192>
- [2] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” 2014, arXiv:1409.3215. doi: <https://dx.doi.org/10.48550/arXiv.1409.3215>
- [3] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2015. doi: <https://dx.doi.org/10.48550/arXiv.1409.0473>
- [4] T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” in *Conference on Empirical Methods in Natural Language Processing*, 2015. doi: <https://dx.doi.org/10.18653/v1/D15-1166> pp. 1412–1421.
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *NIPS*, 2017.

- [6] Z. Wang, Y. Ma, Z. Liu, and J. Tang, “R-transformer: Recurrent neural network enhanced transformer,” 2019, arXiv:1907.05572. doi: <https://dx.doi.org/10.48550/arXiv.1907.05572>
- [7] S. Kapturowski, G. Ostrowski, J. Quan, R. Munos, and W. Dabney, “Recurrent experience replay in distributed reinforcement learning,” in *International Conference on Learning Representations*, 2019.
- [8] M. Hausknecht and P. Stone, “Deep recurrent q-learning for partially observable mdps,” 2015, arXiv:1507.06527. doi: <https://dx.doi.org/10.48550/arXiv.1507.06527>
- [9] L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu, “A study of reinforcement learning for neural machine translation,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018. doi: <https://dx.doi.org/10.18653/v1/D18-1397> pp. 3612–3621.
- [10] L. Yu, W. Zhang, J. Wang, and Y. Yu, “Seqgan: Sequence generative adversarial nets with policy gradient,” in *AAAI*, 2017.
- [11] G. L. Guimaraes, B. Sanchez-Lengeling, P. L. C. Farias, and A. Aspuru-Guzik, “Objective-reinforced generative adversarial networks (organ) for sequence generation models,” 2017, arXiv:1705.10843. doi: <https://dx.doi.org/10.48550/arXiv.1705.10843>
- [12] A. Grissom II, H. He, J. Boyd-Graber, J. Morgan, and H. Daumé III, “Don’t until the final verb wait: Reinforcement learning for simultaneous machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. doi: <https://dx.doi.org/10.3115/v1/D14-1140> pp. 1342–1352.
- [13] J. Gu, G. Neubig, K. Cho, and V. O. Li, “Learning to translate in real-time with neural machine translation,” in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, 2017. doi: <https://dx.doi.org/10.18653/v1/E17-1099> pp. 1053–1062.
- [14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *40th annual meeting on association for computational linguistics*, 2002. doi: <https://dx.doi.org/10.3115/1073083.1073135> pp. 311–318.
- [15] A. Alinejad, M. Siabani, and A. Sarkar, “Prediction improves simultaneous neural machine translation,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018. doi: <https://dx.doi.org/10.18653/v1/D18-1337> pp. 3022–3027.
- [16] F. Dalvi, N. Durrani, H. Sajjad, and S. Vogel, “Incremental decoding and training methods for simultaneous translation in neural machine translation,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018. doi: <https://dx.doi.org/10.18653/v1/N18-2079> pp. 493–499.
- [17] B. Zheng, R. Zheng, M. Ma, and L. Huang, “Simpler and faster learning of adaptive policies for simultaneous translation,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. doi: <https://dx.doi.org/10.18653/v1/D19-1137> pp. 1349–1354.
- [18] J. Ive, A. M. Li, Y. Miao, O. Caglayan, P. Madhyastha, and L. Specia, “Exploiting multimodal reinforcement learning for simultaneous machine translation,” in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, 2021. doi: <https://dx.doi.org/10.18653/v1/2021.eacl-main.281> pp. 3222–3233.
- [19] M. Ma, L. Huang, H. Xiong, R. Zheng, K. Liu, B. Zheng, C. Zhang, Z. He, H. Liu, X. Li, H. Wu, and H. Wang, “STACL: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. doi: <https://dx.doi.org/10.18653/v1/P19-1289> pp. 3025–3036.
- [20] Y. Ren, J. Liu, X. Tan, C. Zhang, T. Qin, Z. Zhao, and T.-Y. Liu, “SimulSpeech: End-to-end simultaneous speech to text translation,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. doi: <https://dx.doi.org/10.18653/v1/2020.acl-main.350> pp. 3787–3796.
- [21] Tatoeba, “<https://tatoeba.org/>,” 2020, retrieved 2020-05-05.
- [22] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, “fairseq: A fast, extensible toolkit for sequence modeling,” in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019. doi: <https://dx.doi.org/10.18653/v1/N19-4009>
- [23] S. Wiseman and A. M. Rush, “Sequence-to-sequence learning as beam-search optimization,” *arXiv preprint arXiv:1606.02960*, 2016. doi: <https://dx.doi.org/10.48550/arXiv.1606.02960>
- [24] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, “spaCy: Industrial-strength Natural Language Processing in Python,” 2020. doi: <https://dx.doi.org/10.5281/zenodo.1212303>
- [25] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *arXiv preprint arXiv:1508.07909*, 2015. doi: <https://dx.doi.org/10.48550/arXiv.1508.07909>

## **B.5. Least Redundant Gated Recurrent Neural Network**

Title	Least Redundant Gated Recurrent Neural Network
Authors	Łukasz Neumann, Łukasz Lepak, Paweł Wawrzyński
Conference	2023 International Joint Conference on Neural Networks (IJCNN 2023)
Year	2023
DOI	10.1109/IJCNN54540.2023.10191895
Ministerial score	140

# Least Redundant Gated Recurrent Neural Network

1<sup>st</sup> Łukasz Neumann  
Institute of Computer Science  
Warsaw University of Technology  
Warsaw, Poland  
lukasz.neumann@pw.edu.pl

2<sup>nd</sup> Łukasz Lepak  
Institute of Computer Science  
Warsaw University of Technology  
Warsaw, Poland  
lukasz.lepak.dokt@pw.edu.pl

3<sup>rd</sup> Paweł Wawrzyński  
Ideas NCBR  
Warsaw, Poland  
pawel.wawrzynski@ideas-ncbr.pl

**Abstract**—Recurrent neural networks are important tools for sequential data processing. However, they are notorious for problems regarding their training. Challenges include capturing complex relations between consecutive states and stability and efficiency of training. In this paper, we introduce a recurrent neural architecture called Deep Memory Update (DMU). It is based on updating the previous memory state with a deep transformation of the lagged state and the network input. The architecture is able to learn to transform its internal state using any nonlinear function. Its training is stable and fast due to relating its learning rate to the size of the module. Even though DMU is based on standard components, experimental results presented here confirm that it can compete with and often outperform state-of-the-art architectures such as Long Short-Term Memory, Gated Recurrent Units, and Recurrent Highway Networks.

**Index Terms**—recurrent neural networks, universal approximation

## I. INTRODUCTION

Recurrent Neural Networks (Recurrent NNs, RNNs) are designed to process sequential data and are vital components of systems that perform speech recognition [1], machine translation [2], handwritten text recognition [3], and other tasks [4].

An intuitively designed RNN is prone to gradient explosions or vanishing [5] due to its recurrent nature. The impact of a given input on future outputs of the RNN may vanish or explode with time. Specialized architectures with gates, namely Long Short-Term Memory (LSTM) networks [6] and Gated Recurrent Unit (GRU) networks [7], are designed to overcome this problem at the level of a single neuron. While these networks are widely successful, they come with a cost — their memory state undergoes only single-layer transformation from one time instant to another.

Several recurrent architectures apply deep processing of their internal states [8]–[10]. However, they are complex or challenging to train.

This paper addresses the above shortcomings by introducing a neural module designed to prevent the previously mentioned gradient problems while allowing the state transformation to be modelled by an arbitrary feedforward neural network. We call this module Deep Memory Update (DMU).<sup>1</sup> As a result, state transformation can easily be shaped in DMU. Additionally, the architecture is resistant to problems of gradient

exploding/vanishing. Experimental results presented in the paper confirm that DMU performs well in comparison to its state-of-the-art counterparts.

RNNs are often outperformed by feedforward networks with attention, especially by the transformer [11]. However, the computational complexity of these techniques excludes them from some applications [12], [13]. It is also likely that some combination of attention and RNNs, such as R-Transformer [14], ASRNN [15] and others [16], will outperform both. Therefore, in this paper, we focus solely on RNNs.

## II. RELATED WORK

Early RNNs [17]–[20] suffered from the problem of gradient vanishing/exploding, defined by [5]: A small change in the RNN's weights causes its future output's change that is vanishing or exploding in time. As a result, the impact of RNN's weights on its performance is either close to zero or infinity. In either case, it is impossible to train such a network. A gradient norm clipping strategy proposed in [8] may mitigate this problem to some extent. [21] used orthogonal matrices of weights in shallow RNNs to stabilize the gradient successfully.

The gradient vanishing/exploding problem was alleviated at a cell level with Long Short-Term Memory (LSTM) networks [6]. A neuron in such a network is a state machine with several so-called gates. The neuron generally preserves its state from one time to another but may also change it. The change depends on the dot product of the neuron inputs and its weights computed in its gates. LSTMs have been enhanced with batch normalization of a recurrent signal [22].

[7] proposed an architecture based on neurons simpler than those in LSTMs, called Gated Recurrent Units (GRUs). Despite its simplicity, it generally preserved the favourable properties of LSTM. [23] proposed a unit whose state was only computed based on its previous state and the outputs of the preceding neural layer. Networks based on such units, Independently Recurrent Neural Networks (IndRNNs), tend to outperform LSTMs and GRUs.

Capturing long-term dependencies in input sequences is a crucial challenge that RNNs face. [24] proposed to increase the lag of recurrent connections in higher network layers geometrically. [25] introduced SkipRNN that learns to skip state updates and shorten the effective size of the computational graph. [26] prove that RNNs operate via transformations

<sup>1</sup>We make the code available at <https://github.com/fuine/dmu>

of time, and the gates in LSTM and GRU networks are a straightforward way to perform these transformations.

LSTMs and GRUs are usually organized in several layers stacked on top of one another [27]. Input to each neuron within a layer includes the previous states of all the neurons in the layer. This way, at each time instant, the network input undergoes a deep transformation. However, the internal state of the network undergoes only a shallow, single-layer transformation.

Being able to apply an arbitrary nonlinear, deep transformation to its internal state is a valuable feature of a recurrent neural network. [28] proposed to increase the recurrence depth by adding multiple nonlinear layers to the recurrent transition, resulting in Deep Transition RNNs (DT-RNNs) and Deep Transition RNNs with Skip connections (DT(S)-RNNs). Gradient propagation issues are exacerbated in these architectures due to long credit assignment paths. [9] added extra connections between all states across consecutive time steps in a stacked RNN, which also increases recurrence depth. However, their model requires additional connections with increasing depth, gives only a fraction of state cells access to the deepest layers, and faces gradient propagation issues along the longest paths.

[10] introduced Recurrent Highway Networks (RHNs), which can be understood as LSTMs with specialized multilayer gates. These networks apply deep processing to their internal state while successfully coping with gradient vanishing/exploding. However, our proposed architecture requires only two state-processing gates as opposed to LSTM's three. Additionally, DMU allows for an arbitrary feedforward network to process the state.

A number of concepts may facilitate the performance of RNNs. [29] proposed a scheme of initialization of weights in these networks. RNNs are usually trained with Stochastic Gradient Descent with gradient estimates computed with backpropagation through time. However, recent work of [30] on forward propagation through time calls this practice into question. An interesting alternative to gated recurrent neural networks is network simulators of continuous dynamical systems [31]–[34].

### III. METHOD

In this section, we introduce the Deep Memory Update (DMU) module. It is a neural module with memory designed to have the following properties:

- 1) Its memory state can undergo an arbitrary nonlinear transformation from one moment to another.
- 2) The module can easily preserve its memory state from one moment of time to another.
- 3) Its learning is relatively fast and stable.

#### A. General structure

We present the structure of the Deep Memory Update (DMU) module in Fig. 1. The module operates in discrete time  $t = 1, 2, \dots$ . At each time, the module is fed with the input  $x_t \in \mathbb{R}^d$

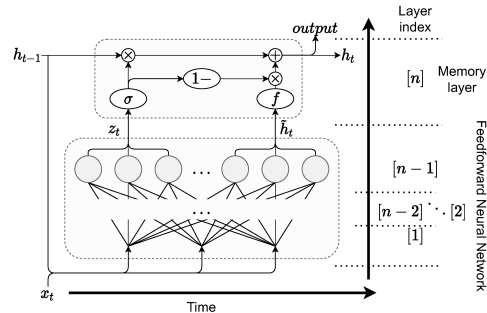


Fig. 1. Structure of Deep Memory Update module. The module comprises the feedforward neural network, which can arbitrarily process the state and a memory layer. The output of the module is also its hidden state.

$\mathbb{R}^m$  and produces the vector  $h_t \in \mathbb{R}^d$ , which is both its memory state and its output.

A lagged memory state,  $h_{t-1}$ , together with an input of the block,  $x_t$ , are fed to a feedforward neural network, FNN. The network's output layer is linear with  $2d$  neurons. It produces two vectors:  $z_t \in \mathbb{R}^d$  determines to what extent the memory state should be preserved, and  $\hat{h}_t \in \mathbb{R}^d$  determines the direction in which the state should change.

A pair of  $i$ -th elements of  $z_t$  and  $\hat{h}_t$  are fed to a  $i$ -th memory cell. The new cell state is a weighted, with  $z_t$ , average of the old state,  $h_{t-1}$ , and  $\hat{h}_t$ . The memory state update takes the form

$$\langle z_t, \hat{h}_t \rangle = \text{FNN}(h_{t-1}, x_t) \quad (1)$$

$$h_t = h_{t-1} \circ \sigma(z_t) + f(\hat{h}_t) \circ (\mathbf{1} - \sigma(z_t)), \quad (2)$$

where " $\circ$ " denotes the elementwise product,  $\mathbf{1}$  is a vector of ones,  $\sigma$  is a unipolar soft step function, e.g. the logistic sigmoid,

$$\sigma_i(z) = \frac{e^{z_i}}{1 + e^{z_i}} \text{ for } z_i \in \mathbb{R}, \quad (3)$$

and  $f$  is an activation function, e.g.

$$f_i(z) = \tanh(z_i) \text{ for } z_i \in \mathbb{R}. \quad (4)$$

Our proposed recurrent architecture is compared with GRU [7] in the supplementary material.

Let us consider how the required properties of DMU are achieved.

- 1) Since a feedforward neural network with at least two dense layers is a universal function approximator, the network state can undergo the arbitrary nonlinear transformation from one time moment to another.
- 2) The block preserves its memory state for large values of  $z_t$ . In particular, for  $z_t = +\infty$  we have  $h_t = h_{t-1}$ .
- 3) For efficient and stable training of the network, it is enough that the learning rate of the module is sufficiently lower than that of the rest of the network, as discussed in Section III-C.

### B. Initialization

The FNN block should be a universal approximator. It can be a multilayer perceptron with at least two layers, including a linear output layer. This layer needs to be linear because its output should not be limited. It should be possible that  $z_t \gg 1$  which causes the memory state to be preserved,  $h_t \cong h_{t-1}$ .

We recommend using the standard ways of initializing neural weight matrices in the FNN block, with one exception. Namely, upon weights' initialization, we recommend adding a positive scalar to the biases of the neurons that produce  $z_t$  values, e.g., 3. With positive elements of  $z_t$ , the memory state of the DMU module will be, by default, largely preserved from one moment  $t$  to another. This addition is optional in most of the tasks, however if the network initialized in the standard way fails to converge, the positive bias usually helps.

We use Xavier initialization [35] in all of the experiments. Additionally, in synthetic tasks, we use the positive bias with a value of 3.

### C. Training

Training of DMU may be based on gradient backpropagation through time and using the gradient with a method of stochastic optimization such as Stochastic Gradient Descent or ADAM [36]. These methods apply a learning rate to each trained weight. In turn, the learning rate defines a speed of optimization along derivatives with respect to this weight. Typically, the learning rates are equal for all weights.

Let us consider DMU as a module in a feedforward architecture. Its learning speed and stability can be noticeably improved by distinguishing a module's learning rate and setting its value smaller than that of the rest of the architecture. The learning of recurrent modules is exposed to instability, which naturally limits its learning speed. Nevertheless, it does not need to limit the learning speed of the surrounding feedforward modules, which are less exposed to instability, and thus may learn faster.

In our experiments in Sec. IV, we combine  $n$ -layer DMU modules with  $n'$ -layer feedforward output subnetworks. For  $\beta > 0$  being a learning rate for the output subnetwork we use a learning rate of the DMU module,  $\beta_{\text{DMU}}$ , equal to

$$\beta_{\text{DMU}} = \frac{\beta}{2n}. \quad (5)$$

The deeper the DMU module, the lower its learning rate. Additionally, when weight decay is used in the network training, its strength in the DMU module is reduced  $2n$  times.

### D. Gradient propagation in DMU

In order to analyze gradient propagation in DMU, we adopt the following further assumptions and notation:

- The detailed structure of the FNN inside DMU is presented in Fig. 2, with  $A_i, B, C, D$  denoting weight matrices,  $a_i, b, c$  denoting bias vectors and  $f_i$  denoting activation functions.
- Activation functions in hidden layers of FNN are bipolar sigmoids with derivatives and absolute values covering the intervals  $(0, 1]$  and  $(-1, 1)$ , respectively.

- $\sigma$  takes the form (3). Therefore,  $\sigma'(z) = \sigma(z)(1-\sigma(z)) < 1 - \sigma(z)$ .
- Vectors considered are in row form.

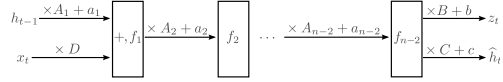


Fig. 2. Structure of the feedforward module inside DMU.  $A_i, B, C$  and  $D$  denote weight matrices,  $a_i, b$  and  $c$  denote vectors of biases and  $f_i$  denotes activation functions.

Note that  $h_t$  are in fact weighted averages, over  $i \geq 0$ , of  $f(\hat{h}_{t-i})$  never exceeding  $(-1, 1)$ . Therefore, the elements of  $h_t$  also never exceed  $(-1, 1)$ .

Let us analyze how the loss  $L_{t'}$  resulting from the network output at time  $t'$  propagates back to time  $t-1 < t'$ . We have the following recursion:

$$\begin{aligned} \frac{dL_{t'}}{dh_{t-1}} &= \frac{dL_{t'}}{dh_t} \frac{dh_t}{dh_{t-1}} \\ &= \frac{dL_{t'}}{dh_t} \frac{d}{dh_{t-1}} \left( h_{t-1} \circ \sigma(z_t) + f(\hat{h}_t) \circ (\mathbf{1} - \sigma(z_t)) \right) \\ &= \frac{dL_{t'}}{dh_t} \left( \text{diag}(\sigma(z_t)) + [h_{t-1}] \circ \frac{dz_t}{dh_{t-1}} \circ [\sigma'(z_t)] + \frac{d\hat{h}_t}{dh_{t-1}} \circ \right. \\ &\quad \left. \circ [f'(\hat{h}_t)] \circ [\mathbf{1} - \sigma(z_t)] - [f(\hat{h}_t)] \circ \frac{dz_t}{dh_{t-1}} \circ [\sigma'(z_t)] \right) \\ &= \frac{dL_{t'}}{dh_t} \left( \text{diag}(\sigma(z_t)) + [h_{t-1} - f(\hat{h}_t)] \circ \frac{dz_t}{dh_{t-1}} \circ [\sigma'(z_t)] \right. \\ &\quad \left. + \frac{d\hat{h}_t}{dh_{t-1}} \circ [f'(\hat{h}_t)] \circ [\mathbf{1} - \sigma(z_t)] \right), \end{aligned} \quad (6)$$

where  $\text{diag}(v)$  denotes the diagonal matrix with the vector  $v$  on its diagonal and  $[v]$  denotes the matrix with the same vector  $v^T$  in each column.

By neglecting activation functions inside the FNN block, we reduce it to a cascade of linear transformations and obtain the following approximations of the Jacobi matrices in (6):

$$\frac{dz_t}{dh_{t-1}} \cong B^T \left( \prod_{i=1}^{n-2} A_i \right)^T, \quad \frac{d\hat{h}_t}{dh_{t-1}} \cong C^T \left( \prod_{i=1}^{n-2} A_i \right)^T. \quad (7)$$

Considering that  $\sigma(z_t) \in (0, 1)$ ,  $h_{t-1} - f(\hat{h}_t) \in (-2, 2)$ ,  $\sigma'(z_t) \in (0, 1 - \sigma(z_t))$ ,  $f'(\hat{h}_t) \in (0, 1)$ , we obtain the following condition on non-increasing gradient:

- \* Eigenvalues of the matrices  $B^T \left( \prod_{i=1}^{n-2} A_i \right)^T$  and  $C^T \left( \prod_{i=1}^{n-2} A_i \right)^T$  remain in the intervals  $(-1/2, 1/2)$  and  $(-1, 1)$ , respectively.

Essentially, that means that the components of the weight matrices in the FNN block should not be too large.

When the above condition (\*) is satisfied, the gradient decreases when propagated back according to the first component of (6), that is, by a factor of  $\sigma(z_t)$ . Intuitively, when the memory



state  $h_{t-1}$  is preserved to another time-step proportionally to  $\sigma(z_t)$ , the impact of this memory state on future performance is preserved likewise.

#### IV. EXPERIMENTAL STUDY

To evaluate the DMU architecture, we test it on three synthetic problems and three modern problems based on real-life data. The synthetic problems are taken from [6], and are noisy sequences, adding, and temporal order. The modern data-based problems are polyphonic music modelling [37], natural language modelling [38], and Spanish/German/Portuguese to English machine translation tasks [39], [40].

We compare our DMU module using shallow architectures with ordinary recurrent neural networks (RNNs), GRU, LSTM, and RHN in the synthetic problems. We also compare DMU in its deep version with RHN in the data-based problems. To make the comparison fair, we embed a recursive subnetwork within the same neural architecture. That subnetwork is a layer or a few layers of recurrent units or a DMU module or RHN. Moreover, for each depth of RNNs, we compare different architectures of similar sizes measured by the number of weights.

A reader may find details of our experimental setting, hyperparameters of architectures and their training in the supplementary material.

##### A. Adding problem

The first task will be called “Adding”. It is taken from [6, sec. 5.4].

*a) Results.:* We present the results for the adding problem in Fig. 3. We conclude that DMU significantly outperforms all other modules, and GRU scores better than LSTM. RNN and RHN are not able to reach any threshold within 100 training epochs for any hyperparameters.

##### B. Temporal order

The next task, referred to as “TempOrd”, is taken from [6, sec. 5.6, Task 6b].

*a) Results.:* The results for the TempOrd task are depicted in Fig. 4. We note that DMU has faster convergence than GRU and maintains similar results for high thresholds (up to  $10^{-4}$ ). For lower thresholds, DMU outperforms GRU. LSTM reaches partial success on higher thresholds but fails for lower ones. RNN and RHN fail for all thresholds without a single successful 100 epoch run.

##### C. Noise-free and noisy sequences

We call this task “NoiseSeq”. It is taken from [6, sec. 5.2].

*a) Results.:* Figure 5 contains the results for the NoiseSeq task. We observe that GRU and DMU obtain similar results, in most cases reaching all the loss thresholds, with GRU training faster. RHN in about half of the cases does not reach any threshold, and in the other half, it reaches all of them. RNN performs worse than RHN, and LSTM performs worse than RNN.

##### D. Polyphonic music modelling

In this subsection, we evaluate modules on the polyphonic music modelling task, referred to as “PolyMusic”, based on the Nottingham music dataset [37].

*a) Results.:* The results of the polyphonic music modelling can be found in Table I. In this problem, DMU outperforms RHN at 3 out of 4 depths with regard to test mean loss.

TABLE I  
POLYMUSIC: RESULTS — LOSS.  $N$  DENOTES THE NUMBER OF HIDDEN LAYERS.

$N$	model	train			test		
		best	$\mu$	$\sigma$	best	$\mu$	$\sigma$
1	RHN	3.34	3.38	0.08	<b>3.552</b>	<b>3.598</b>	<b>0.037</b>
	DMU	<b>2.94</b>	<b>2.96</b>	<b>0.04</b>	3.57	3.63	0.05
2	RHN	3.39	3.41	0.10	3.55	3.61	0.06
	DMU	<b>3.02</b>	<b>3.09</b>	<b>0.07</b>	<b>3.49</b>	<b>3.55</b>	<b>0.04</b>
5	RHN	3.44	3.68	0.16	3.73	3.85	0.11
	DMU	<b>3.22</b>	<b>3.21</b>	<b>0.06</b>	<b>3.63</b>	<b>3.69</b>	<b>0.03</b>
10	RHN	3.70	3.93	0.15	<b>3.90</b>	4.08	0.12
	DMU	<b>3.52</b>	<b>3.54</b>	0.15	3.95	<b>4.04</b>	<b>0.09</b>

##### E. Natural language modelling

The task called “NatLang” is based on the Penn Treebank corpus of English [41].

*a) Results.:* Table II shows the results. DMU achieves consistently better results than RHN, often by a large margin. Only for a depth of 2 RHN performs slightly better than DMU with respect to the mean test perplexity.

TABLE II  
NATLANG: RESULTS — PERPLEXITY (LOWER = BETTER).  $N$  IS THE DEPTH OF THE NETWORK.

$N$	model	train			test		
		best	$\mu$	$\sigma$	best	$\mu$	$\sigma$
1	RHN	61.10	66.53	6.07	106.11	110.39	4.36
	DMU	<b>56.63</b>	<b>59.42</b>	<b>3.14</b>	<b>105.35</b>	<b>106.13</b>	<b>0.48</b>
2	RHN	<b>62.90</b>	66.32	4.23	<b>104.89</b>	<b>109.83</b>	4.04
	DMU	64.42	<b>64.86</b>	<b>1.43</b>	109.60	110.13	<b>0.47</b>
5	RHN	<b>82.33</b>	<b>86.10</b>	3.83	123.16	124.97	1.92
	DMU	92.40	94.06	<b>1.11</b>	<b>117.92</b>	<b>120.37</b>	<b>1.46</b>
10	RHN	<b>85.97</b>	149.43	86.87	<b>124.46</b>	171.60	60.38
	DMU	118.20	<b>119.48</b>	<b>1.53</b>	130.05	<b>131.83</b>	<b>1.44</b>

##### F. Machine translation

Next, we test the modules in the context of machine translation using recurrent architectures. The task is based on datasets of pairs of corresponding Spanish/Portuguese/German and English sentences [39], [40]. We will call experiments based on subsequent pairs “Spa2Eng”, “Por2Eng”, and “Ger2Eng”.

*a) Results.:* Table III contains the results. DMU achieves a better perplexity score than RHN for all three language pairs at each depth of both networks except for Portuguese at depth 1. Additionally, both networks achieve the best results for a depth of 1 or 2. Performance generally deteriorates with growing depth, significantly faster for RHN than for DMU.

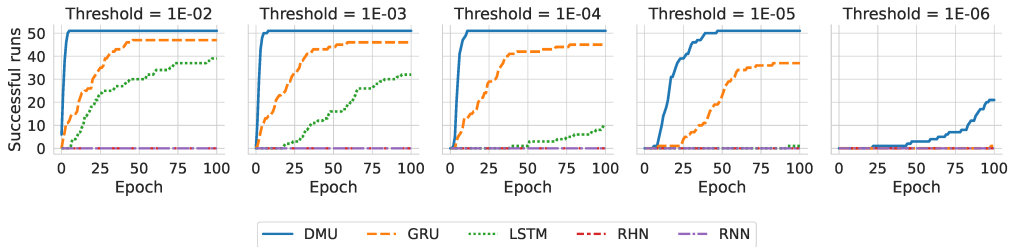


Fig. 3. Adding: Results of 51 runs, five graphs for different loss thresholds, a curve presents how many runs reach a given loss threshold at a given training epoch.

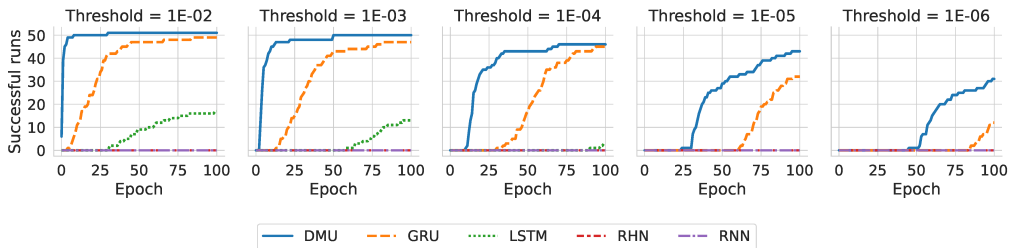


Fig. 4. TempOrd: Results of 51 runs, five graphs for different loss thresholds.

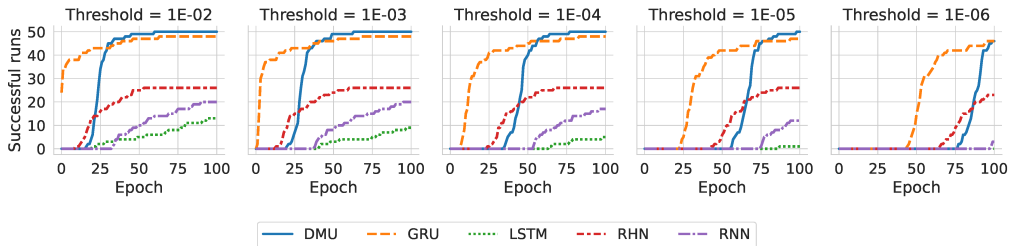


Fig. 5. NoiseSeq: Results of 51 runs, five graphs for different loss thresholds.

### G. Ordered and permuted MNIST

Finally, we compare DMU to selected state-of-the-art modules on the pixel-by-pixel MNIST image classification problem [29]. Each image is represented as a flattened array of pixels, and the module processes it one after another. Such setup allows us to evaluate the internal state drift on long inputs, as each image contains 784 pixels. The task comes in two flavors - sequential, in which each image is flattened in a row-wise manner and permuted, in which we apply the same random permutation to each image after flattening.

a) Results.: Table IV contains the results.

### H. Learning rate ablation

We verify how a reduction of a DMU learning rate according to (5) impacts the performance of the neural architecture with

this module. In this order, we register the performance of each architecture with approximately optimized, with a grid search, learning rate. In one variant, the learning rate is constant for the whole architecture. In the other, the learning rate of the DMU module and the learning rate for the rest are bound with (5). Numerical results of this ablation are presented in Tables. V–VII. Note that this ablation does not make sense for the analyzed synthetic problems because the recurrent module is the entire architecture in these cases. The results confirm that efficiency benefits from reducing the learning rate of the DMU module.

## V. DISCUSSION

Since the seminal paper of [6] the development of recurrent neural networks has been stimulated by the need to avoid

TABLE III  
TRANSLATION: RESULTS — PERPLEXITY (LOWER = BETTER). L. — LANGUAGE PAIR.

l.	N	model	train			test		
			best	$\mu$	$\sigma$	best	$\mu$	$\sigma$
Spa2Eng	1	RHN	8.26	7.72	0.29	<b>5.80</b>	6.31	0.36
		DMU	<b>7.13</b>	<b>7.24</b>	<b>0.17</b>	5.89	<b>6.05</b>	<b>0.14</b>
	2	RHN	<b>8.82</b>	<b>8.50</b>	<b>0.22</b>	<b>6.53</b>	<b>7.10</b>	0.36
		DMU	9.07	8.52	0.38	6.91	7.18	<b>0.32</b>
	5	RHN	12.38	24.68	12.48	12.34	47.60	41.68
		DMU	<b>7.74</b>	<b>7.91</b>	<b>0.35</b>	<b>7.50</b>	<b>8.02</b>	<b>0.35</b>
10	RHN	58.74	58.73	0.92	110.54	141.77	56.01	
	DMU	<b>8.78</b>	<b>8.84</b>	<b>0.26</b>	<b>7.99</b>	<b>8.40</b>	<b>0.23</b>	
Por2Eng	1	RHN	3.95	3.98	<b>0.09</b>	3.65	3.80	0.16
		DMU	<b>3.91</b>	<b>3.93</b>	0.11	<b>3.54</b>	<b>3.68</b>	<b>0.10</b>
	2	RHN	<b>4.29</b>	<b>4.33</b>	<b>0.06</b>	<b>3.67</b>	<b>3.93</b>	0.18
		DMU	4.63	4.45	0.14	3.74	3.97	<b>0.14</b>
	5	RHN	6.25	7.50	0.83	6.55	7.62	0.85
		DMU	<b>4.65</b>	<b>4.74</b>	<b>0.09</b>	<b>4.56</b>	<b>4.69</b>	<b>0.10</b>
10	RHN	48.58	48.35	0.37	79.10	99.86	23.80	
	DMU	<b>5.12</b>	<b>5.29</b>	<b>0.18</b>	<b>4.92</b>	<b>5.06</b>	<b>0.12</b>	
Ger2Eng	1	RHN	4.66	4.63	0.16	4.27	4.35	<b>0.06</b>
		DMU	<b>4.28</b>	<b>4.50</b>	<b>0.12</b>	<b>4.10</b>	<b>4.21</b>	0.08
	2	RHN	<b>5.26</b>	<b>5.10</b>	0.11	<b>4.41</b>	<b>4.59</b>	<b>0.10</b>
		DMU	5.38	5.28	<b>0.07</b>	4.56	4.79	0.19
	5	RHN	8.13	9.24	1.24	7.93	9.18	1.28
		DMU	<b>5.33</b>	<b>5.42</b>	<b>0.12</b>	<b>5.21</b>	<b>5.33</b>	<b>0.12</b>
10	RHN	47.99	48.41	0.29	83.47	134.79	92.37	
	DMU	<b>5.91</b>	<b>5.82</b>	<b>0.23</b>	<b>5.63</b>	<b>5.74</b>	<b>0.07</b>	

gradient exploding or vanishing in backpropagation through time. Indeed, these phenomena are likely to occur in neural networks with feedback loops. In LSTM and GRU architectures, they were eliminated at the cell level.

The DMU neural module introduced in this paper is based on memory cells whose state is updated with the weighted average of their previous content and new values proposed for them. Both the weights and the new proposed values come from a feedforward subnetwork whose inputs include the previous

TABLE IV  
TEST ACCURACY ON ORDERED AND PERMUTED PIXEL-BY-PIXEL MNIST.

Name	ordered	permuted	N	# params
LSTM baseline by [21]	97.3%	92.7%	128	≈68K
MomentumLSTM [42]	99.1%	94.7%	256	≈270K
Unitary RNN [21]	95.1%	91.4%	512	≈9K
Full Capacity Unitary RNN [43]	96.9%	94.1%	512	≈270K
Soft orth. RNN [44]	94.1%	91.4%	128	≈18K
Kronecker RNN [45]	96.4%	94.5%	512	≈11K
Antisymmetric RNN [33]	98.0%	95.8%	128	≈10K
Incremental RNN [31]	98.1%	95.6%	128	≈4K/8K
Exponential RNN [46]	98.4%	96.2%	360	≈69K
Sequential NAIS-Net [34]	94.3%	90.8%	128	≈18K
Lipschitz RNN [32]	<b>99.4%</b>	<b>96.3%</b>	128	≈34K
DMU (ours)	98.5%	93.4%	96	≈20K
DMU (ours)	98.7%	93.4%	128	≈34K

TABLE V  
POLYMUSIC: VARIED LEARNING RATE ABLATION RESULTS — LOSS. DMU-C — DMU WITH AN EQUAL LEARNING RATE FOR ALL MODULES.

N	model	train			test		
		best	$\mu$	$\sigma$	best	$\mu$	$\sigma$
1	DMU-C	<b>2.72</b>	<b>2.81</b>	0.08	<b>3.34</b>	<b>3.38</b>	<b>0.04</b>
	DMU	2.94	2.96	<b>0.04</b>	3.57	3.63	0.05
2	DMU-C	<b>2.99</b>	<b>3.03</b>	0.20	<b>3.43</b>	<b>3.49</b>	0.04
	DMU	3.02	3.09	<b>0.07</b>	3.49	3.55	<b>0.04</b>
5	DMU-C	3.25	3.31	0.24	3.90	3.99	0.11
	DMU	<b>3.22</b>	<b>3.21</b>	<b>0.06</b>	<b>3.63</b>	<b>3.69</b>	<b>0.03</b>
10	DMU-C	3.75	nan	nan	4.26	5.03	0.63
	DMU	<b>3.52</b>	<b>3.54</b>	<b>0.15</b>	<b>3.95</b>	<b>4.04</b>	<b>0.09</b>

TABLE VI  
NATLANG: VARIED LEARNING RATE ABLATION RESULTS — PERPLEXITY. DMU-C — DMU WITH AN EQUAL LEARNING RATE FOR ALL MODULES.

N	model	train			test		
		best	$\mu$	$\sigma$	best	$\mu$	$\sigma$
1	DMU-C	68.99	71.12	<b>1.78</b>	109.23	111.20	1.42
	DMU	<b>56.63</b>	<b>59.42</b>	3.14	<b>105.35</b>	<b>106.13</b>	<b>0.48</b>
2	DMU-C	81.31	81.01	1.69	117.56	118.17	0.65
	DMU	<b>64.42</b>	<b>64.86</b>	<b>1.43</b>	<b>109.60</b>	<b>110.13</b>	<b>0.47</b>
5	DMU-C	108.90	579.68	235.39	138.13	542.20	202.04
	DMU	<b>92.40</b>	<b>94.06</b>	<b>1.11</b>	<b>117.92</b>	<b>120.37</b>	<b>1.46</b>
10	DMU-C	696.30	697.57	<b>0.69</b>	642.18	642.91	<b>0.44</b>
	DMU	<b>118.20</b>	<b>119.48</b>	1.53	<b>130.05</b>	<b>131.83</b>	1.44

TABLE VII  
TRANSLATION: VARIED LEARNING RATE ABLATION RESULTS — ACCURACY. DMU-C — DMU WITH AN EQUAL LEARNING RATE FOR ALL MODULES.

l.	N	model	train			test		
			best	$\mu$	$\sigma$	best	$\mu$	$\sigma$
Spa2Eng	1	DMU-C	0.91	0.85	0.14	<b>0.70</b>	0.67	0.04
		DMU	<b>0.93</b>	<b>0.93</b>	<b>0.00</b>	0.70	<b>0.69</b>	<b>0.01</b>
	2	DMU-C	0.84	0.76	0.20	0.66	0.60	0.11
		DMU	<b>0.94</b>	<b>0.94</b>	<b>0.01</b>	<b>0.69</b>	<b>0.69</b>	<b>0.01</b>
	5	DMU-C	0.71	0.66	0.06	0.62	0.59	0.03
		DMU	<b>0.84</b>	<b>0.83</b>	<b>0.01</b>	<b>0.66</b>	<b>0.65</b>	<b>0.01</b>
10	DMU-C	0.34	0.30	0.02	0.35	0.31	0.02	
	DMU	<b>0.76</b>	<b>0.75</b>	<b>0.01</b>	<b>0.63</b>	<b>0.63</b>	<b>0.00</b>	
Por2Eng	1	DMU-C	<b>0.94</b>	0.93	0.02	<b>0.78</b>	0.77	0.01
		DMU	0.93	<b>0.94</b>	<b>0.01</b>	0.78	<b>0.77</b>	<b>0.00</b>
	2	DMU-C	0.94	0.93	0.02	0.75	0.75	0.01
		DMU	<b>0.96</b>	<b>0.95</b>	<b>0.01</b>	<b>0.78</b>	<b>0.77</b>	<b>0.00</b>
	5	DMU-C	0.77	0.63	0.20	0.70	0.59	0.16
		DMU	<b>0.86</b>	<b>0.86</b>	<b>0.00</b>	<b>0.73</b>	<b>0.73</b>	<b>0.01</b>
10	DMU-C	0.32	0.32	<b>0.00</b>	0.32	0.32	<b>0.00</b>	
	DMU	<b>0.81</b>	<b>0.80</b>	0.01	<b>0.71</b>	<b>0.71</b>	0.00	
Ger2Eng	1	DMU-C	0.92	0.92	0.02	0.75	0.75	<b>0.00</b>
		DMU	<b>0.92</b>	<b>0.93</b>	<b>0.01</b>	<b>0.75</b>	<b>0.75</b>	0.00
	2	DMU-C	0.88	0.90	0.01	0.73	0.72	0.01
		DMU	<b>0.94</b>	<b>0.94</b>	<b>0.01</b>	<b>0.75</b>	<b>0.74</b>	<b>0.00</b>
	5	DMU-C	0.72	0.70	0.02	0.66	0.65	0.01
		DMU	<b>0.85</b>	<b>0.84</b>	<b>0.01</b>	<b>0.71</b>	<b>0.70</b>	<b>0.01</b>
10	DMU-C	0.44	0.34	0.06	0.45	0.34	0.05	
	DMU	<b>0.80</b>	<b>0.78</b>	<b>0.01</b>	<b>0.68</b>	<b>0.68</b>	<b>0.00</b>	

state of the memory cells. Architectures based on the DMU

module compete with and often outperform those based on LSTM or GRU. The gradient vanishing/exploding problem is solved in DMU at the module level.

In some applications, deep transformation of the network state is necessary. However, then the effective length of the gradient path increases, which may destabilize training. RHN successfully coped with this problem at the expense of the complexity of its architecture. DMU applies a typical feedforward block of any depth for state transformation. Training stability is ensured by appropriately reducing the learning rate of the DMU module. As a result, DMU performed better than RHN of the same depth in all three analyzed data-based problems with a handful of exceptions.

Interestingly, contrary to [10] we note that depth-scaling of the model did not yield better results. We speculate that it can be explained by the lack of regularization other than weight decay. This was a deliberate choice to compare RHN and DMU modules without any unnecessary architectural additions.

In the future, we want to further investigate DMU's fast convergence rate on synthetic tasks. A greater understanding of the model's behaviour could help us improve the architecture and provide additional insight into the state drift problem of RNNs in general.

#### VI. CONCLUSIONS

In this paper, we propose DMU — a recurrent neural module that can perform an arbitrary nonlinear transformation of its memory state. Three experiments with synthetic data (Adding, Temporal order, Noisy sequence) presented here compare neural architectures based on DMU with those based on RNN, LSTM, and GRU. DMU yields the best results in two of them while having results comparable to the best module in the third one. Three experiments with real-life data (Polyphonic music, Natural language modelling, Machine translation) compare neural architectures based on DMU with those based on Recurrent Highway Networks of the same depth. The architecture based on DMU outperformed RHN in 15 out of 20 analyzed mean test score cases while staying competitive in the other five cases.

#### ACKNOWLEDGMENTS

The project was funded by POB Research Centre for Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative Program – Research University (ID-UB). We gratefully acknowledge the contribution of Aleksander Zamojski, Lidia Wojciechowska and Monika Berlińska to the code of DMU.

#### REFERENCES

- [1] A. Graves, A. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, arXiv:1303.5778, 2013.
- [2] Y. Wu, M. Schuster, *et al.*, *Google's neural machine translation system: Bridging the gap between human and machine translation*, arXiv:1609.08144, 2016.
- [3] T. Capes, P. Coles, *et al.*, "Siri on-device deep learning-guided unit selection text-to-speech system," in *Inter-speech*, 2017, pp. 4011–4015.
- [4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
- [5] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [7] K. Cho, B. V. Merriënboer, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.
- [8] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML*, 2013, pp. 1310–1318.
- [9] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *ICML*, 2015, pp. 2067–2075.
- [10] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," in *ICML*, 2017.
- [11] A. Vaswani, N. Shazeer, *et al.*, "Attention is all you need," in *NIPS*, 2017.
- [12] Z. Jia, Y. Lin, *et al.*, "Hetemotionnet: Two-stream heterogeneous graph recurrent neural network for multimodal emotion recognition," in *ACM Int. Conf. on Multimedia*, 2021, pp. 1047–1056.
- [13] C. Hansen, C. Hansen, *et al.*, "Contextual and sequential user embeddings for large-scale music recommendation," in *ACM Conf. on Recommender Systems*, 2020, pp. 53–62.
- [14] Z. Wang, Y. Ma, Z. Liu, and J. Tang, *R-transformer: Recurrent neural network enhanced transformer*, arXiv:1907.05572, 2019.
- [15] J. C.-W. Lin, Y. Shao, Y. Djenouri, and U. Yun, "Asrnn: A recurrent neural network with an attention model for sequence labeling," *Knowledge-Based Systems*, vol. 212, p. 106548, 2021.
- [16] Z. Liu, C. Lu, H. Huang, S. Lyu, and Z. Tao, "Hierarchical multi-granularity attention-based hybrid neural network for text classification," *IEEE Access*, vol. 8, pp. 149362–149371, 2020.
- [17] M. I. Jordan, "Serial order: A parallel, distributed processing approach," *Advances in Connectionist Theory Speech*, vol. 121(ICS-8604), pp. 471–495, 1986.
- [18] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
- [19] A. J. Robinson and F. Fallside, "The utility driven dynamic error propagation network," Cambridge University, Engineering Department, Tech. Rep. CUED/F-INFENG/TR.1, 1987.
- [20] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988.

- [21] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *ICML*, 2016, pp. 1120–1128.
- [22] T. Cooijmans, N. Ballas, C. Laurent, Çağlar Gülçehre, and A. Courville, "Recurrent batch normalization," in *ICLR*, 2017.
- [23] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (indrn): Building a longer and deeper rnn," in *CVPR*, 2018.
- [24] S. Chang, Y. Zhang, *et al.*, "Dilated recurrent neural networks," in *NIPS*, 2017.
- [25] V. Campos, B. Jou, X. G. i Nieto, J. Torres, and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," in *ICLR*, 2018.
- [26] C. Tallec and Y. Ollivier, "Can recurrent neural networks warp time?" In *ICLR*, 2018.
- [27] A. Graves, *Generating sequences with recurrent neural networks*, arXiv:1308.0850, 2013.
- [28] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *ICLR*, 2014.
- [29] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.
- [30] A. Kag and V. Saligrama, "Training recurrent neural networks via forward propagation through time," in *ICML*, 2021, pp. 5189–5200.
- [31] A. Kag, Z. Zhang, and V. Saligrama, "RNNs incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients?" In *ICLR*, 2020.
- [32] N. B. Erichson, O. Azencot, A. Queiruga, L. Hodgkinson, and M. W. Mahoney, "Lipschitz recurrent neural networks," *arXiv preprint arXiv:2006.12070*, 2020.
- [33] B. Chang, M. Chen, E. Haber, and E. Chi, "AntisymmetricRNN: A dynamical system view on recurrent neural networks," in *ICLR*, 2019.
- [34] M. Ciccone, M. Gallieri, J. Masci, C. Osendorfer, and F. Gomez, "Nais-net: Stable deep networks from non-autonomous differential equations," in *NIPS*, 2018, pp. 3025–3035.
- [35] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
- [36] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.
- [37] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *ICML*, 2012.
- [38] W. Zaremba, I. Sutskever, and O. Vinyals, *Recurrent neural network regularization*, arXiv:1409.2329, 2014.
- [39] Tatoeba, *Https://tatoeba.org*, Retrieved 2020-05-05, 2020.
- [40] ManyThings, *Http://www.manythings.org/anki/*, Retrieved 2020-05-05, 2020.
- [41] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.
- [42] T. M. Nguyen, R. G. Baraniuk, A. L. Bertozzi, S. J. Osher, and B. Wang, "Momentumrnn: Integrating momentum into recurrent neural networks," *arXiv preprint arXiv:2006.06919*, 2020.
- [43] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," in *NIPS*, 2016, pp. 4880–4888.
- [44] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal, "On orthogonality and learning recurrent networks with long term dependencies," in *ICML*, 2017, pp. 3570–3578.
- [45] C. Jose, M. Cisse, and F. Fleuret, "Kronecker recurrent units," in *ICML*, 2018, pp. 2380–2389.
- [46] M. Lezcano-Casado and D. Martinez-Rubio, "Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group," in *ICML*, 2019, pp. 3794–3803.
- [47] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, *On the properties of neural machine translation: Encoder-decoder approaches*, arXiv:1409.1259, 2014.
- [48] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, arXiv:1409.3215, 2014.

#### APPENDIX A COMPARISON OF DMU AND GRU

In the notation applied in this paper operation of a GRU [7] layer can be expressed as

$$\begin{aligned} r_t &= W_r x_t + U_r h_{t-1} + b_r \\ \hat{h}_t &= W_h x_t + U_h (\sigma(r_t) \circ h_{t-1}) + b_h \\ z_t &= W_z x_t + U_z h_{t-1} + b_z \\ h_t &= h_{t-1} \circ \sigma(z_t) + f(\hat{h}_t) \circ (\mathbf{1} - \sigma(z_t)) \end{aligned}$$

where  $W_r, U_r, W_h, U_h, W_z, U_z$  and  $b_r, b_h, b_z$  are matrices and vectors of weights. The operation of DMU is presented in eqs. (1) and (2). In the most straightforward configuration, this network is a layer of linear units. Then

$$\begin{aligned} \hat{h}_t &= W_h x_t + U_h h_{t-1} + b_h \\ z_t &= W_x x_t + U_x h_{t-1} + b_x \\ h_t &= h_{t-1} \circ \sigma(z_t) + f(\hat{h}_t) \circ (\mathbf{1} - \sigma(z_t)) \end{aligned} \tag{A.1}$$

Therefore, DMU is simpler in this basic configuration, thus having fewer weights per memory cell than a layer of GRUs, as it does not have the reset gate. In the general configuration, DMU can apply an arbitrary nonlinear transformation to its state, which GRU is unable to do. In practice, GRU layers are often stacked on one another which improves its performance on tasks that require complex nonlinear transformation of state. However, the state of the stacked GRU layers still can not be

TABLE VIII  
ARCHITECTURES USED FOR THE COMPARISON OF DIFFERENT NEURAL MODULES IN SYNTHETIC EXPERIMENTS.<sup>1</sup> RECURRENT BLOCK.

experiment		RNN	LSTM	GRU	RHN	DMU
NoiseSeq	rc. blk <sup>1</sup>	(5, 5)	(2, 2)	(2, 3)	((3, 3))	((5, 4))
	weights no.	595	880	687	672	573
Adding	rc. blk <sup>1</sup>	(5, 5)	(2, 2)	(3, 2)	((4, 3))	((5, 5))
	weights no.	111	99	108	136	106
TempOrd	rc. blk <sup>1</sup>	(6, 6)	(2, 3)	(2, 4)	((4, 3))	((5, 6))
	weights no.	236	212	208	224	203

TABLE IX  
ARCHITECTURES USED FOR THE COMPARISON OF RHN AND DMU. WE REPORT THE NUMBER OF NEURONS IN FEEDFORWARD LAYERS. THE LAST LAYER OF THE DMU'S FNN ON THE TRANSLATION TASK ALWAYS HAS 200 NEURONS. FOR THE TRANSLATION TASK, WEIGHTS' NUMBERS ARE PROVIDED FOR SPA2ENG, GER2ENG, AND POR2ENG, RESPECTIVELY.

experiment	depth	RHN	DMU	weights no.
PolyMusic	1	100	100	46.7K
	2	100	122	66.9K
	5	100	131	127K
	10	100	136	228K
NatLang	1	100	100	1.7M
	2	100	122	1.7M
	5	100	131	1.8M
	10	100	136	1.9M
Translation	1	200	200	27.8M/36.6M/24.5M
	2	200	340	28.0M/36.8M/24.7M
	5	200	300	28.4M/37.3M/25.2M
	10	200	300	29.2M/38.1M/26.0M

arbitrarily transformed in a single time instant since parts of this state are transformed within single layers.

LSTM [6] and RHN [10] are based on different, much more complex equations with even more weights. LSTM has twice more weights per memory cell than DMU has in the basic configuration.

## APPENDIX B EXPERIMENTS

### A. Architectures

We present architectures for each problem in Table VIII and Table IX. Corresponding hyperparameters can be found in Table VIII. The recurrent subnetwork is characterized by the number of units in subsequent layers. For example, a GRU subnetwork with two layers of 10 and 20 neurons will be briefly denoted by (10, 20). A DMU block with two FNN layers of 10 and 20 neurons will be denoted by (10, 20, 10) to account for the layer of memory cells within the block. In the data-based problems, we evaluate each module at varying depths. In all cases, the compared architectures have matching numbers of trained parameters. Hyperparameters for the models were selected based on the random and grid searches and then fine-tuned manually. The metric used to evaluate the hyperparameters was calculated on the validation subset in each case.

### B. Training

The data is split into training, validation, and testing set. On synthetic problems, training continues until the loss reaches a

specified threshold ( $10^{-6}$ ) on the validation set or the training budget is depleted. The error is then registered on the testing set and presented here. We follow a similar procedure for real-life problems, except the training process is stopped once the optimizer reaches the final epoch. All metrics are calculated using the model from the epoch with the best metric score on the validation set.

We run the experiment five times for each modern task/model/depth combination and aggregate the results. Standard result aggregation, such as averaging loss over time, would not be interpretable in the synthetic tasks since training is often unstable in these experiments. Therefore, the results for each synthetic problem are presented for multiple thresholds of the loss value. We plot the number of experiment runs that have reached the threshold in or before the specific epoch for each threshold. These thresholds allow us to assess how fast and how likely the module converges to a specific loss value. Thus, we can gain an insight into the quality of the module. Faster attainment of a specific threshold and convergence to lower thresholds are both desirable for the algorithm.

Hyperparameters used for each experiment/neural module are presented in Table X and Table XI. We use ADAM optimizer to train all architectures.

TABLE X  
HYPERPARAMETERS USED FOR SYNTHETIC TASKS.

experiment	hyperparameter	RNN	LSTM	GRU	RHN	DMU
NoiseSeq	learning rate	0.01	0.002	0.05	0.05	0.02
	seq. per epoch	200	200	200	200	200
	min seq. length	100	100	100	100	100
	max epochs	100	100	100	100	100
Adding	learning rate	0.01	0.001	0.05	0.02	0.02
	seq. per epoch	200	200	200	200	200
	min seq. length	100	100	100	100	100
	max epochs	100	100	100	100	100
TempOrd	learning rate	0.01	0.005	0.02	0.02	0.05
	seq. per epoch	200	200	200	200	200
	min seq. length	100	100	100	100	100
	max epochs	100	100	100	100	100

### C. Hardware

Our experiments have been performed on a PC equipped with AMD™Ryzen 1920X, 64GB RAM, 4xNVIDIA™RTX 2070 Super.

### D. Testing strategy

To evaluate synthetic tasks, we run an experiment for each module 51 times and aggregate the results. On real-life data tasks, we aggregate results over five runs for each recurrent module. We report metrics obtained in the best runs. These runs are selected based solely on their performance on the test set. Therefore, in some cases, metrics reported in the best column for the training dataset are worse than those in the mean column.

TABLE XI  
HYPERPARAMETERS USED FOR EACH EXPERIMENT AND EACH NEURAL MODULE.

experiment	depth	hyperparameter	RHN	DMU
PolyMusic	all	max epochs	500	500
	1	learning rate	0.005	0.005
		weight decay	0.001	0.0001
		scheduler gamma	1.0	1.0
	2	learning rate	0.005	0.005
		weight decay	0.001	0.0001
		scheduler gamma	1.0	1.0
	5	learning rate	0.005	0.005
		weight decay	0.001	0.0001
		scheduler gamma	1.0	1.0
	10	learning rate	0.005	0.002
		weight decay	0.001	0.0001
scheduler gamma		1.0	1.0	
NatLang	all	max epochs	40	40
	1	learning rate	0.02	0.02
		weight decay	0.0001	0.0001
		scheduler gamma	0.9	0.9
	2	learning rate	0.02	0.02
		weight decay	0.0001	0.0001
		scheduler gamma	0.9	0.9
	5	learning rate	0.02	0.01
		weight decay	0.0001	0.0001
		scheduler gamma	0.9	0.98
	10	learning rate	0.02	0.02
		weight decay	0.0001	0.0001
scheduler gamma		0.9	0.98	
Spa2Eng/ Por2Eng/ Deu2Eng	all	teacher forcing ratio	1.0	1.0
	1	max epochs	50	50
		learning rate	0.01	0.005
		weight decay	0.0001	0.0001
	2	scheduler gamma	0.9	0.9
		learning rate	0.01	0.01
		weight decay	0.0001	0.0001
	5	scheduler gamma	0.9	0.9
		learning rate	0.01	0.003
		weight decay	0.0001	0.0001
	10	scheduler gamma	0.9	1.0
		learning rate	0.01	0.003
weight decay		0.0001	0.0001	
	scheduler gamma	0.9	1.0	

#### E. Adding problem

In this problem, the network is fed with two-dimensional vectors  $[a, b]$ , where  $a$  is randomly chosen from the interval  $[-1, 1]$ , and  $b \in \{-1, 0, 1\}$  is a marker:  $-1$  denotes the first and last element of the sequence, there are two pairs marked by 1, the rest are marked by 0. The task of the network is to output the sum of  $a$ -s accompanied by  $b$ -s equal to 1 at the end of the sequence. Each network analyzed is composed of a recurrent block and a layer with softmax activation.

#### F. Temporal order

This task evaluates network's ability to model temporal ordering of data. The input and the output are both 8-

dimensional. They represent one of 8 symbols by one-hot encoding. The input symbols are:  $E$  (start),  $B$  (end),  $X$  or  $Y$ .  $X$  or  $Y$  occur at time  $t_1, t_2, t_3$ . In all three of these occurrences the choice of  $X$  or  $Y$  is random, the rest of a sequence is filled with symbols  $a, b, c, d$  also selected at random. Sequence length is chosen randomly between 100 and 110.  $t_1, t_2, t_3$  are selected randomly for each sequence, respectively between 10-20, 33-43 and 66-76. The output desired at the end of a sequence is either  $Q, R, S, U, V, A, B, C$ , depending on the combination of symbols that has occurred at times  $t_1, t_2$  and  $t_3$ . Each network analyzed is composed of a recurrent block and a layer with softmax activation.

#### G. NoiseSeq

We use noisy sequences to test the modules on the long time lag problems. The network is fed with symbols one-hot encoded in  $n$ -dimensional vectors. An input sequence is, with equal probability 0.5, either  $(x, a_1, \dots, a_{n-2})$  or  $(y, a_1, \dots, a_{n-2})$ , where  $x, y, a_1, \dots, a_{n-1}$  are selected on random prior to an experiment. The task of the network is to output the first symbol in the input sequence when at  $n - 1$ -st step. Each analyzed neural network is composed of a recurrent block and a layer with softmax activation.

#### H. PolyMusic

Inputs and outputs are 88-dimensional. They represent the binary encoding of possible piano-rolls at a current timestep (in MIDI note numbers, between 21 and 108 inclusive). Sequences vary in length. The task of the model is to predict the next time step in the sequence (i.e., output at time  $t$  is equal to input at time  $t + 1$ ). The loss function is a negative log-likelihood averaged over all time steps in the dataset/batch. The neural network is composed of a recurrent block and a layer with the sigmoid activation.

#### I. NatLang

Inputs and outputs are single number representations of the most frequent words in English and special tokens such as "unknown" or "end of sequence". Sequences include 100 words. The goal of the network is to predict another word within the current sequence. The loss function is perplexity (categorical cross-entropy exponent). See [38] for details. The whole neural network comprises a recurrent block, followed by a 100-neurons dense layer and an output layer with the softmax activation. For this experiment, the input word embedding is set to a small size (64) on purpose to limit overfitting.

#### J. Machine Translation

We use tokens representing words, punctuation marks, *sentence start*, and *sentence end* in all languages. Each token is encoded as a single, unique number. The goal is to translate Spanish/Portuguese/German sentences into English ones using a system with encoder-decoder architecture [7], [47], [48]. A whole translator has encoder-decoder architecture. An encoder is a recurrent block. A decoder is composed of a recurrent block and a layer with the softmax activation. Additionally, we use input and output embeddings of size 650.