Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Doctoral Thesis

Training instabilities in neural network-based sequential data modeling

Łukasz Neumann, M.Sc.

supervisor
Robert Nowak, PhD DSc

WARSAW 2023

Training instabilities in neural network-based sequential data modeling

Abstract.

This thesis presents a series of publications regarding instabilities during neural networks' optimization process (also called training) in sequential data modeling. In our study, we investigate two approaches to solving such problems: specialized neural architectures and second-order optimizers.

In the first part of our work, we focus on custom architectures reducing problems with training instabilities. First, we show that a combination of fixed state processing and custom convolutional neural network architecture is sufficient to successfully process a series of medical images. Based on this approach, we introduce the first fully-automatic allergic skin reaction recognition system. We show that our method yields accurate results comparable between patients, as opposed to current, manual state-of-the-art. Next, we propose a minimal gated recurrent neural architecture with an arbitrary, nonlinear state transformation. We theoretically show that this architecture alleviates gradient propagation issues while using fewer parameters than its competitors. Further, we present experimental results which indicate that our method often outperforms state-of-the-art solutions, especially on problems with high state drift. Interestingly, we note that the recurrent neural network's performance degrades with the depth of the state transformation. This observation holds on several tasks for two different architectures.

In the second part, we look closer at second-order optimization algorithms as an alternative to currently often-used gradient ones. To this end, we propose a modification of an evolution strategy algorithm, tailoring it for neural network training. We show that such an approach allows for optimizing both convolutional and recurrent neural networks. Notably, the proposed method can effectively train Jordan recurrent neural networks on tasks with significant gradient propagation problems, in contrast to the state-of-the-art first-order method. Finally, we conclude that combining second-order optimization with gradient information results in the best-performing models.

Overall, our research contributes to the mitigation strategies against training instabilities of neural architectures on sequential data. We introduce two new architectural and one optimization solution to the instability problems. One of the architectural solutions serves as a basis for the novel, real-life medical data analysis system.

Keywords: Sequential Data, Recurrent Neural Networks, Neural Network Training, Optimization

Niestabilności w procesie uczenia sieci neuronowych przy modelowaniu danych sekwencyjnych

Streszczenie.

W niniejszej pracy przedstawiamy serię publikacji poświęconej zagadnieniom niestabilności w procesie uczenia (optymalizacji) sieci neuronowych przy modelowaniu danych sekwencyjnych. W ramach przedstawionych prac badamy dwa podejścia do rozwiązania problemów związanych z niestabilnościami: dedykowane architektury sieci neuronowych oraz optymalizatory drugiego rzędu.

W pierwszej części pracy skupiamy się na architekturach sieci neuronowych zaprojektowanych na potrzeby niwelacji niestabilności w procesie uczenia. Pokazujemy, że połączenie z góry narzuconego, nieelastycznego sposobu przetwarzania stanu oraz dedykowanej konwolucyjnej sieci neuronowej wystarczy aby poprawnie przetwarzać serię obrazów medycznych. Na podstawie tego podejścia wprowadzamy pierwszy w pełni automatyczny system do rozpoznawania reakcji alergicznych skóry. Zaproponowana przez nas metoda otrzymuje dobre wyniki, które mogą być porównywalne pomiędzy pacjentami, w przeciwieństwie do obecnego, manualnego podejścia. Następnie prezentujemy minimalną rekurencyjną sieć neuronową z mechanizmem bramkującym oraz arbitralnym, nieliniowym przekształceniem stanu. Pokazujemy teoretycznie, że zaproponowana architektura rozwiązuje problemy z propagacją gradientu, przy użyciu mniejszej liczby parametrów niż konkurencyjne architektury. Ponadto demonstrujemy wyniki eksperymentalne, które wskazują, że nasz model często uzyskuje lepsze wyniki w porównaniu do obecnie używanych modeli, w szczególności na problemach z dużym dryfem stanu. W wyniku eksperymentów dla dwóch różnych architektur na kilku różnych zadaniach dochodzimy do ciekawego wniosku - jakość wyników rekurencyjnej sieci neuronowej spada wraz z głębokością transformacji stanu.

W drugiej części pracy skupiamy się na optymalizatorach drugiego rzędu jako alternatywie dla obecnie często używanych algorytmach bazujących na gradiencie. W ramach tych prac prezentujemy modyfikację algorytmu strategii ewolucyjnej, którą przystosowujemy do treningu sieci neuronowych. Pokazujemy, że nasz algorytm pozwala na optymalizację konwolucyjnych i rekurencyjnych sieci neuronowych. W szczególności zaproponowana metoda potrafi nauczyć rekurencyjną sieć neuronową Jordana na zadaniach z poważnymi problememami propagacji gradientu, które nie są rozwiązywane przez metody pierwszego rzędu. Pokazujemy również, że łącząc optymalizację drugiego rzędu z informacją o gradiencie otrzymujemy najlepiej działające modele.

Podsumowując, nasza praca przyczynia się do rozszerzenia strategii mitygacji niestabilności treningu sieci neuronowych przy modelowaniu danych sekwencyjnych. Wprowadzamy dwa architekturalne i jedno optymalizacyjne rozwiązanie problemów ze stabilnością procesu optymalizacji sieci neuronowych. Jedno z architekturalnych rozwiązań stanowi podstawę nowatorskiego systemu analizy danych medycznych.

Słowa kluczowe: Dane Sekwencyjne, Rekurencyjne Sieci Neuronowe, Trening Sieci Neuronowych, Optymalizacja

Contents

1. Introduction

A collection of at least two ordered data points with repetitions allowed creates a sequence and so we call such data sequential. Typically, a sequential dataset comprises multiple sequences, each treated as a single sample. The length of a specific sequence can vary between samples within a single dataset. The sequence length usually determines the amount of information that such a sequence carries.

Sequential data is found abundantly in the world around us. Time series are sequences of arbitrary values in time, such as temperature, stock value, the electrical activity of the heart or brain, or the position of celestial bodies in the sky. Predicting future data points in such sequences has been an essential task for humanity dating back at least to Ancient Greece, as indicated by the first analog computer, the Antikythera mechanism, which has been used to predict astronomical positions (Freeth et al., 2006). Nowadays, time series are analyzed in multiple important research fields – weather forecasting (Fathi et al., 2022). Apart from sequential data that are not ordered according to time, numerous types of sequences are used throughout science. Genetic sequences of RNA or DNA are the basis of the current genomics and virology research (Lesk, 2017; Chiara et al., 2021). Another example of important sequential data is natural language, where characters, syllables, words, and sentences can be treated as different sequences. Breakthroughs in natural language processing allow us to automatically process large volumes of spoken and written data (Chowdhary, 2020). Some researchers claim that these advances in language data processing allow for artificial intelligence models which show signs of General Intelligence (Bubeck et al., 2023).

Due to the varying size of samples, the amount of information contained within a single sample, and mutual information shared between single data points within a single sequence modeling sequential data can be challenging. There are several ways to process sequential data. Predicting the next data point based on the sequence is one of them, with examples such as weather forecasting or natural language modeling. Another approach is to calculate an arbitrary value based on the sequence, such as classification, e.g., ECG signal classification or genetic illness detection based on the genome sequence. A sequence can also be transformed to form a different sequence, such as text translation from one natural language to another.

Classical approaches to sequential data modeling vary depending on the task. Typically, for time series forecasting models based on autoregression and moving average are used, such as autoregressive moving average (ARMA) (Whittle, 1951), autoregressive integrated moving average (ARIMA) (Box and Jenkins, 1970) or autoregressive fractionally integrated moving average (ARFIMA) (Granger and Joyeux, 1980; Hosking, 1981). (Baum and Petrie, 1966) introduced the Hidden Markov model (HMM), which allows for modeling the system, which is a Markov process. HMMs have been successfully applied to numerous sequential data processing tasks (Mor et al., 2021). Dagum et al. (1992, 1995) proposed a generalization of methods such as ARMA, Kalman filters, and

HMM called Dynamic Bayesian Networks (DBN). DBN can model an arbitrary nonlinear time-based sequence.

Processing sequential data plays an important role in the machine learning landscape (Dietterich, 2002). Notable classic approaches are conditional random fields (Lafferty et al., 2001), structured support vector machines (Tsochantaridis et al., 2005) or recurrent neural networks (RNNs) (Elman, 1990; Robinson and Fallside, 1987; Werbos, 1988; Jordan, 1986). With the increasing computational power, the application of artificial neural networks to sequential data has been gaining notoriety in multiple domains, such as natural language processing (Manning, 2022; Wei et al., 2022; OpenAI, 2023; Touvron et al., 2023), time series forecasting (Hewamalage et al., 2021), music information retrieval (Choi et al., 2016; Sturm et al., 2016; Oore et al., 2020) or genetic data analysis (Levy et al., 2020; Koumakis, 2020; Routhier and Mozziconacci, 2022).

Despite these success stories, modeling sequential data using neural networks poses several challenges. Training instability is a known phenomenon when using gradient-based optimizers in recurrent neural architectures (Bengio et al., 1994). Traditionally, this problem has been solved by changing the architectures of the recurrent modules (Hochreiter and Schmidhuber, 1997; Cho et al., 2014; Zilly et al., 2017). Unfortunately, this solution usually results in networks with more parameters and more complicated state processing. Notably, these architectures do not allow for an arbitrary state transformation, despite some approaches to provide a deeper state processing in state-of-the-art architectures (Pascanu et al., 2014; Chung et al., 2015; Zilly et al., 2017).

Another approach to these problems is the usage of the second-order optimization technique. Among others, evolutionary algorithms (EA) can be used to train neural networks (Stanley et al., 2019). One particularly interesting class of these algorithms is evolution strategies (ES), which are known to perform well in a multitude of situations. However, most state-of-the-art evolution strategies rely on costly matrix operations to achieve good results. As they are usually applied to search spaces with up to several hundred dimensions, they are not a feasible basis for a neural network optimizer.

1.1. Thesis contributions

This thesis, presented in the form of a series of publications, is devoted to solving the problem of neural network training's instability in the context of sequential data. The main contributions can be summarized as follows:

- We propose a novel least redundant gated recurrent neural network architecture that solves the gradient explosion/vanishing problem while reducing the number of parameters compared to state-of-the-art models.
- We introduce a metaheuristics-based training method for neural networks. Using this approach, we subsequently show that the second-order optimization method allows for training vanilla recurrent neural networks, particularly on tasks with significant state drift problems.
- We show how differential images between series can improve the model's performance on medical data classification task. Combining this strategy with custom neural architecture, we propose the first fully-automated approach to skin allergic reaction recognition.

In Chapter 2 we describe the current state of the art in neural network optimizers, architectural solutions to the problem of training instabilities of recurrent neural networks, and draw a picture of skin allergy recognition. As presented in Fig. 1, training instabilities can be addressed in two distinct phases of solving a recurrent task.

The first phase takes place during the architecture design of the network. In Chapter 3 we propose two distinct architectural solutions to the instability problems. One solution uses a convolutional neural network (CNN) and fixed-state processing, while the other is a new, minimal gated recurrent module. We show the effectiveness of both approaches, while also introducing the first fully-automated skin allergic reaction recognition system based on the former solution.

On the other hand, instead of an architectural solution, a different kind of optimizer can be used to train neural networks. In Chapter 4 we introduce a new second-order neural network optimizer and show its effectiveness on several different tasks for both convolutional and recurrent neural networks.

1.2. List of publications

The thesis is based on three original works (2 published and one accepted for publication):

- **[P1] Neumann Łukasz**, Nowak Robert, Stępień Jacek, Chmielewska Ewelina, Pankiewicz Patryk, Solan Radosław, and Jahnz-Różyk Karina. "Thermography based skin allergic reaction recognition by convolutional neural networks.", Scientific Reports 12.1 (2022).
  Contribution:
  As the first author in this publication, the PhD Candidate designed an image alignment algorithm, neural classifier, and U-Net segmentation presented in the publication. In collaboration with co-authors, PhD Candidate implemented and tested the solution, as well as prepared the manuscript for publication.
  Ministerial score: **140**
  Impact factor: **4.996**
  Percentage of contribution: **70%**
- **[P2] Neumann Łukasz**, Lepak Łukasz, Wawrzyński Paweł. "Least redundant gated recurrent neural network.", 2023 International Joint Conference on Neural Networks (IJCNN), IEEE, 2023, *(accepted for publication)*
  Contribution:
  As the first author, the PhD Candidate was responsible for developing and analyzing the proposed method. Together With co-authors, PhD Candidate designed the algorithm, prepared the manuscript for publication, and presented it during the conference.
  Ministerial score: **140**
  Percentage of contribution: **70%**
- **[P3]** Jagodziński Dariusz, **Łukasz Neumann**, and Paweł Zawistowski[1]. "Deep Neuroevolution: Training Neural Networks Using a Matrix-Free Evolution Strat-

[1] All authors contributed equally.

egy." Neural Information Processing: 28th International Conference, ICONIP 2021.

Contribution:

As the equally contributing author, the PhD Candidate implemented, tested, and analyzed the proposed algorithm in this publication. With co-authors, the PhD candidate designed the method and prepared the manuscript for publication.

Ministerial score: **140**

Percentage of contribution: **33.3**%

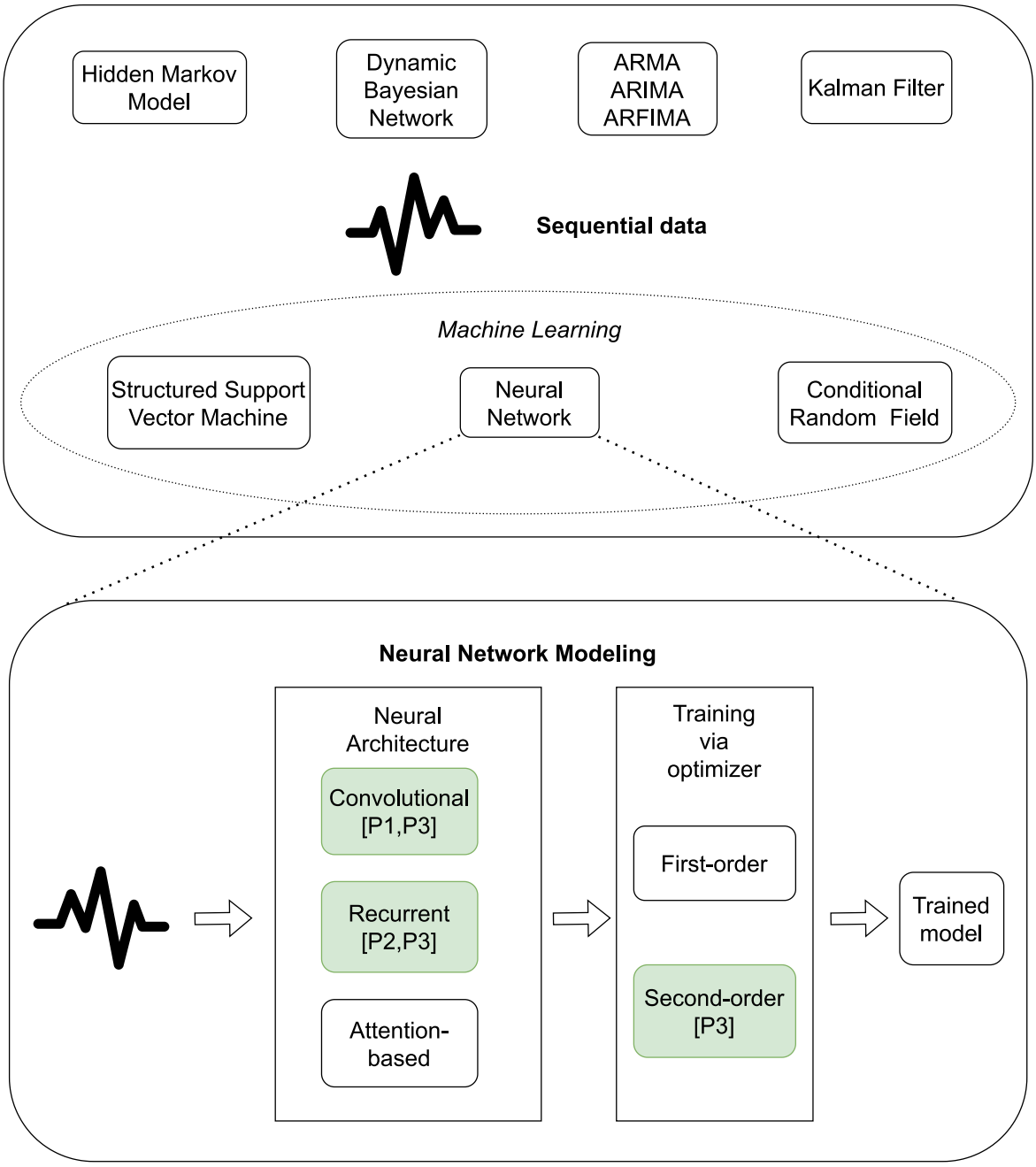


Figure 1. Training neural architectures in the sequential data processing context. To mitigate optimization instabilities, different approaches are used, either architectural or optimizer-based. Boxes with green backgrounds indicate solutions investigated in this thesis.

2. Background

2.1. Training neural networks

In this work, we focus on the problems encountered during training neural networks, particularly in the context of sequential data modeling. Mathematically, training is an optimization task defined as follows:

$$\min_{w} \quad Q(w) = \frac{1}{n} \sum_{i=1}^{n} Q_i(w),$$

where w are the networks parameters (also called weights), $Q(w)$ is the loss function, and Q_i is the value of the loss function on the i-th data sample. There are two main classes of methods that solve such task: gradient-based (first-order) and Hessian-based (second-order) approaches. Most methods follow the iterative approach, wherein at each step of the algorithm the weights of the network are corrected until the (hopeful) convergence of the process.

2.1.1. First-order methods

Gradient descent The most widely used training methods nowadays are based on the gradient descent mechanism, first introduced in (Cauchy et al., 1847). This approach requires that the loss function be differentiable and works by adjusting the weights in opposition to the gradient. It can be formalized as:

$$w := w - \eta \nabla Q(w) = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(w),$$

where η is a scaling factor influencing the magnitude of the parameters' update, conventionally known as the *learning rate*. First used in the context of non-linear problems by (Curry, 1944), this method needs to calculate the gradient in each step. This is impractical, as it effectively forces the processing of the entire dataset in each step.

Stochastic Gradient Descent (SGD) is an adaptation of the gradient descent method, which allows for calculating the estimate of the gradient on the (typically small) subset of data, called *batch*. As the name suggests, the algorithm calculates the stochastic estimation of the gradient in each step and uses it to adjust the weights of the model:

$$w := w - \eta \nabla Q_i(w)$$

where $Q_i(w)$ is the loss calculated on the batch. SGD can converge to local minima and usually establishes a pattern of "zig-zagging" between steps, even for trivial quadratic optimization problems.

Momentum mechanism was proposed in (Rumelhart et al., 1986) in order to help SGD converge quicker and solve the problem of zig-zagging. This extension to the original algorithm linearly combines a gradient update in a specific step with a gradient update from the previous step:

$$w := w - \eta \nabla Q_i(w) + \alpha \Delta w$$

where Δw is the gradient update from $i-1$th step and α is the exponential decay factor which weights the contribution of the new gradient update in relation to the previous update. Intuitively, this algorithm recreates a movement of a heavy ball (parameters) on the surface (search space). Nesterov (2003) proposed a popular modification of the SGD with momentum, wherein the order of weight and momentum updates is switched; that is, the jump in the accumulated direction is done first, followed by the correction to the velocity vector.

A further variation of the SGD focused on the per-parameter learning rate. AdaGrad (Duchi et al., 2011) achieved it by tracking the gradient squares per parameter and scaling the base learning rate accordingly. RMSProp (Tieleman et al., 2012) does so by accumulating the vector of the running averages of the magnitudes of gradients and dividing the base learning rate by such vector.

Kingma and Ba (2014) introduced an RMSProp-based algorithm called Adaptive Moment Estimation (Adam). This method combines the per-parameter learning rate technique from RMSProp with the momentum mechanism. Adam proved to be a widely used optimizer (Llugsi et al., 2021) and has been extended or modified in numerous fashions. AdaMax is a variation proposed in the original Adam paper (Kingma and Ba, 2014), which is based on the infinity norm. Loshchilov and Hutter (2017) proposed decoupling weight decay regularization in Adam.

Gradient-based methods typically employ the backpropagation technique(Werbos, 1974) to calculate the update for each parameter. This technique applies the Leibniz chain rule to each layer of the network, propagating the gradient from the last layer to the first one. While this method is widely used, it has some limitations. In one of them, the vanishing gradient problem (Hochreiter et al., 2001), as the depth of the network increases, the gradient tends to decrease exponentially. This can cause the weight updates in the first layer to become extremely small, or in certain situations, the training process may come to a halt altogether. Another issue regarding the backpropagation algorithm is the imposed limitations on the neurons' activation functions (Rojas, 1996, Chapter 7). Daskalakis and Panageas (2018) theoretically showed that gradient descent methods do not to converge to local min-max solutions (also called saddle points).

2.1.2. Second-order methods

An alternative to gradient-based optimizers is second-order methods. These algorithms use the Hessian matrix as a basis for the optimization process. They can thus converge faster and to a global minimum by leveraging the curvature of the objective

function. However, the operations on the Hessian matrix are computationally costly both in terms of memory ($\mathcal{O}(n^2)$) and time ($\mathcal{O}(n^3)$). With current neural architectures, the size of the search space and thus the size of the Hessian, the application of second-order methods which use this matrix is unfeasible. There are several approaches to approximate the Hessian matrix and reduce the computational complexity such that they become applicable to training neural networks.

Levenberg–Marquardt algorithm (LM) (Levenberg, 1944; Marquardt, 1963) is a modification of the Gauss-Newton method (Gauss, 1809) which uses gradient information. Both were designed to solve nonlinear least squares problems in an iterative fashion. LM uses an additional regularization term that controls the balance between first-order and second-order contributions to the optimization process. Classic LM algorithm works for small architectures, but as the size of the network grows, the computational complexity renders it unusable (Bilski et al., 2018). Several modifications of the LM exist, which allow it to operate for bigger networks. Ampazis and Perantonis (2002) proposed the addition of momentum to the algorithm helping the convergence on flat spots. Bilski et al. (2020) proposed local modification of the algorithm, which splits the Jacobian matrix allowing for a better computational complexity.

Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) (Liu and Nocedal, 1989) is a method that approximates the BFGS algorithm using constrained memory resources. Instead of storing the inverse of the entire Hessian matrix, the algorithm estimates it based on selected vectors resulting in linear space scaling. L-BFGS uses gradient information to guide its optimization and estimate the Hessian. One of the significant constraints of L-BFGS is its inability to work with small batches of data. There have recently been attempts at mitigating this problem; however, the results are mixed depending on the task and optimized architecture (Bollapragada et al., 2018; Berahas and Takáč, 2020).

Truncated-Newton methods are a family of second-order algorithms that solve optimization problems with high dimensionality. The approach consists of two loops. The inner one iteratively tries to solve the Newton equations and is truncated, hence the name. The outer loop focuses on the nonlinear optimization task, iteratively updating the solution using the results of the inner solver. Martens et al. (2010) proposed its usage to train neural networks, calling the approach Hessian-free.

Kronecker-factored Approximate Curvature (K-FAC) introduced by Martens and Grosse (2015) is an efficient second-order optimizer. The low computational complexity is achieved by approximating the inverse of the Fisher information matrix in a specific format that lends itself to the Kronecker factorization. While relatively fast, the original method only works for fully-connected networks. Ba et al. (2017) extended K-FAC to support convolutional neural networks, and Martens et al. (2018) introduced the modification of the algorithm for the recurrent architectures. Notably, these architectural constraints prevent the algorithm from optimizing arbitrary neural networks. Tang et al. (2021) proposed modifying the method to bridge the computational gap between it and first-order methods. However, this modification only works for fully-connected and convolutional architectures.

Neuroevolution denotes the idea of using evolutionary algorithms to design and train neural networks (Risi and Togelius, 2015; Stanley et al., 2019). These approaches are not necessarily second-order optimizers; for example, Genetic Algorithm was used to successfully train neural networks in the reinforcement learning setting (Such et al., 2017). In this setting Salimans et al. (2017) showed that Evolution Strategies (ESs) (Schwefel and Schwefel, 1977; Rechenberg, 1973) are also viable network optimizers. ESs are a class of algorithms inspired by natural evolution and iteratively modify (mutate) the population (set of potential solutions called "individuals") and select the best-performing individuals.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001) is the best known algorithm in this class. It has been successfully applied to multiple optimization problems. CMA-ES uses an estimated covariance matrix to draw new populations from the multivariate Gaussian distribution. This results in basic computational complexity of $\mathcal{O}(n^3)$. As such, CMA-ES can be feasibly applied to problems with up to several hundred dimensions and does not lend itself to neural network optimization. Notably, there is a hypothesis that the covariance matrix in the CMA-ES approximates the inverted Hessian of the search space. Shir and Yehudayoff (2020) proved it for a static model relying on a quadratic approximation.

Differential Evolution Strategy (DES) was introduced by Jagodziński and Arabas (2017); Arabas and Jagodziński (2020) to overcome the computational complexity of CMA-ES. This method is a crossover between CMA-ES and Differential Evolution (DE) (Storn and Price, 1995). Specifically, DES does not use the covariance matrix to generate a new population. Instead, a history of population midpoints is a basis for univariate Gaussian random vectors, which are combined with difference vectors between individuals from past populations. The authors of DES proved that populations generated using their approach are effectively drawn from the multivariate Gaussian distribution, such as in CMA-ES, but without estimating or calculating the covariance matrix. While the computational complexity problems are mitigated in DES, it is still a black-box second-order optimizer not adapted to neural network training. Similarly to L-BFGS, DES does not work with batched data. Instead, the individual needs to be evaluated on the entire dataset. Moreover, the suggested initial population for the DES should be drawn from the uniform distribution. This goes against standard initialization techniques for neural networks, which take into account the architecture of the network (Glorot and Bengio, 2010; He et al., 2015). Finally, DES does not use explicit gradient information. While this is desirable for some tasks and neural architectures, there are cases in which gradient information is available and does not cause stability issues during training.

2.2. Neural architectures for sequential data modeling

Recurrent neural networks (RNNs) Traditionally, to model sequential data in neural networks, a recurrent architecture was used (Elman, 1990; Robinson and Fallside, 1987; Werbos, 1988; Jordan, 1986). The one introduced by Jordan (1986) is usually considered the "vanilla" RNN architecture. In this setting, a single cell processes data series one step at a time in the following manner:

$$h_t = f(wh_{t-1} + ux_t + b)$$

where x_t is the input at t-th time step, u, w, b are cell's parameters, f is the activation function of the network and h_t is the *hidden state* of the network. This architecture allows for the accumulation of information between different time steps of the input.

To train such networks using a gradient-based method, a Backpropagation Through Time (BPTT) algorithm (Robinson and Fallside, 1987; Werbos, 1988; Mozer, 1995) is used. In essence, this technique unfolds the network in time, creating a graph of networks, each with input at a specific time step, connected with hidden states between time steps. These networks share the weights, as they are essentially the same network in each time step. Then, using the chain rule, the gradient is computed and propagated backward through time, hence the name. Finally, the weights are updated with respect to the gradient accumulated across all time steps. The size of the graph is equal to the length of the input series. What follows is that for longer input sequences, problems with gradient are especially highlighted. Even a slight change in the RNN's weights can lead to gradient vanishing or exploding, as described by Bengio et al. (1994). As a result, the weights' updates tend to zero or infinity. In either case, the training process does not succeed. Pascanu et al. (2013) alleviated the problem to some extent by introducing a gradient norm clipping strategy. Arjovsky et al. (2016) showed that gradient stabilization is possible in shallow RNNs when orthogonal weights matrices are used. Most popular solutions to the gradient problems are based on special recurrent architectures, which employ the usage of a gating mechanism. While the BPTT approach is the most popular one, there recently has been research proposing the usage of forward propagation through time to train RNNs (Kag and Saligrama, 2021).

Long Short-Term Memory (LSTM) , introduced by Hochreiter and Schmidhuber (1997), solve the problem of vanishing/exploding gradient. The idea behind this architecture is to include an additional module that decides when to store some information in the hidden state and when to forget it. This is achieved by introducing three *gates*, which regulate the information flow into the cell. LSTM has three gates, namely input, output, and forget gate, which work in the following manner:

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

where f_t, i_t, o_t are forget, input, and output gates' activations respectively, $\tilde{c}_t$ is cell input activation, c_t is cell state, tanh is hyperbolic tangent activation function, σ is a sigmoid activation function and $\odot$ is a Hadamard product. Forget gate regulates the

extent to which the previously accumulated information should be upheld. In contrast, the input gate determines which information from the current input should be incorporated into the cell's state. Each gate does so by assigning a real number between 0 and 1 to each value in the state/input vector, where 0 would discard the information while 1 would keep it. The output gate decides which information from the current state should be output, again by assigning values between 0 and 1.

LSTM architecture successfully mitigates the gradient vanishing problem by allowing gradients to flow unchanged through time. The architecture has been widely applied to solve problems in fields such as speech recognition (Graves, 2013), machine translation (Wu et al., 2016), drug design (Gupta et al., 2018), and many others (Schmidhuber, 2015). However, this success comes at a price. LSTM has a substantially more complex architecture, which incurs both computational costs, as well as memory costs (more parameters due to three gates). There have been several LSTM extensions, but most of them still suffer from the problems mentioned above. Gers and Schmidhuber (2001); Gers et al. (2002) introduced peephole connections, which allow the gates to use the cell state to calculate their activations. Cooijmans et al. (2016) proposed batch normalisation of the hidden-to-hidden transition. Nguyen et al. (2020) showed that the addition of momentum into the architecture allows for faster convergence of the model.

Gated Recurrent Unit (GRU) is a simplification of the LSTM architecture proposed by Cho et al. (2014). GRU contains only two gates, as opposed to LSTM's three, namely the input and forget gate, respectively called the update and the reset gate. Their interactions work in the following way:

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

where z_t, r_t are update and reset gate activations respectively and $\hat{h}_t$ is the candidate for the new hidden state. There are alternative formulations of the GRU cell by Dey and Salem (2017). GRU reduces the number of gates (and thus the number of parameters in the network) while maintaining results comparable to LSTM.

Deep state transformations in RNNs Typically architectures based on LSTM or GRU cells organize them in multiple consecutive layers (Graves, 2013). Each neuron within a layer receives input that comprises the previous states of all the neurons in that layer. This allows the whole network to perform a deep transformation of the input. Nevertheless, the internal state of the network undergoes a shallow transformation limited to a single layer.

Pascanu et al. (2014) introduced Deep Transition RNNs (DT-RNNs) and Deep Transition RNNs with Skip connections (DT(S)-RNNs) in which multiple nonlinear layers are used for the recurrent transition. This deepens the state transformation. However, such an approach suffers from gradient vanishing due to long credit assignment paths.

Chung et al. (2015) increased the recurrence depth of a stacked RNN by adding the global gating unit, which mediates the signal flow between different layers of the network. The resulting architecture, called Gated-Feedback RNN (GF-RNN), results in greater recurrence depth. On the other hand, GF-RNN necessitates more connections as the depth increases, allows only a subset of state cells to reach the deepest layers, and encounters gradient propagation issues over longer paths.

Zilly et al. (2017) proposed Recurrent Highway Networks (RHN), which builds on the LSTM architecture by adjusting the gates in a multilayer fashion. The resulting cell is able to apply a deep transformation of its state while successfully coping with the gradient vanishing problem. Nevertheless, as it is still an architecture based on LSTM, it still requires three gates (and thus more parameters) compared to architectures with fewer gates.

Continuous-time recurrent neural networks (CTRNN) are one particularly interesting approach to recurrent neural networks which has been recently gaining traction Ciccone et al. (2018); Chang et al. (2019); Kag et al. (2020); Erichson et al. (2020); Rusch et al. (2022). These architectures are based on a system of ordinary differential equations (ODEs) to model the state of the network. CTRNNs are known to outperform gated recurrent neural networks and, in some cases, attention-based architectures.

Attention-based approach , such as transformer (Vaswani et al., 2017), often tend to outperform recurrent neural networks. Notably, these architectures do not suffer from the vanishing gradient due to the skip connections. On the other hand, there are some areas/tasks in which usage of attention-based models is not feasible (Hansen et al., 2020; Jia et al., 2021). Additionally, some architectures, e.g., R-Transformer (Wang et al., 2019), ASRNN (Lin et al., 2021), MahNN (Liu et al., 2020) combine attention and RNNs to achieve better results.

2.3. Allergic skin reaction recognition

In our work, we show how fixed state processing can be used to classify medical sequential data - allergic skin reaction images. In this section we provide the background information about state of the art in the diagnosis of the allergic reactions.

Apart from medical history, there are currently two popular approaches to diagnosing allergies to specific reactions. In the first one, the level of particular Immunoglobin E antibodies in human serum or plasma is measured in vitro (Lambert et al., 2015). This test can only be used to diagnose type I (immediate) allergic reactions. The second approach is an in vivo method employing the usage of skin tests (Heinzerling et al., 2013). Type I allergy can be diagnosed with the skin prick test, while patch tests are used to diagnose type IV allergies. Using this technique, a small amount of allergen is placed on the skin, which is then punctured by the lancet. Additionally, histamine and negative control are applied, serving as reference points for the diagnosis. Next, external signs of the reaction are observed, and both wheal and erythema areas are measured. Finally, the diagnosis is made by a trained professional on an arbitrary scale from zero to five. Due to the ability of skin tests to map the specific reaction of a patient's immune system to an allergen, it is generally believed that these tests serve as a dependable indicator

of an individual's hypersensitivity to the substances being tested (Larenas-Linnemann et al., 2017).

What follows are clear limitations of the current approach. First, the grading is subjective, so meta-analysis of patients diagnosed by different professionals is problematic. Diagnosis is based on the visual assessment and measurement of the wheal. However, Nelson et al. (1998) showed that the technique used to puncture a patient's skin influences the wheal size. Moreover, the wheal is created by the fluid released by the skin's microvessels under the influence of histamine. Unfortunately, the wheal may also be caused by toxic dermatitis or even by negative control. Therefore, the wheal is a non-specific symptom of the allergic reaction.

Recently, non-invasive low-wavelength infrared (LWIR) imaging has been successfully used to diagnose skin burns, breast cancer, and melanoma skin cancer (Gurjarpadhye et al., 2015). Thermography can be used to assess acute inflammatory changes (Ramirez-GarciaLuna et al., 2022). Notably, it can serve as the basis of the wheal recognition (Baillie et al., 1990; Justo et al., 2016). Rok et al. (2017) demonstrated that thermovision-based tests yield a high level of sensitivity, specificity, and accuracy, exhibiting a solid agreement between the reference method employed in the clinic and the in vitro serum Immunoglobin E results.

There have been few attempts at automatic skin allergic reaction recognition, however, the ones based on thermography were purely exploratory. Moreno et al. (2020) showed that β-Lactam allergy classification is possible using a neural network trained on patients' questionnaire data. Convolutional networks can be used to diagnose skin diseases (Shanthi et al., 2020; Vanitha and Geetha, 2021). The link between thermal and allergic reactions has been observed (Rok et al., 2016, 2017; Stanev et al., 2020). However, these works do not introduce a fully automatic end-to-end classification approach. Anzengruber et al. (2019) attempted to classify allergic skin reactions using patch tests and the FLIR ONE application. All of the mentioned thermographic-based works suffer from small dataset sizes.

3. Mitigating stability problems using custom neural architectures

In this chapter, we present the architectural approach to solving network instability problems in sequential data modeling. All of the following observations are described in detail in publications P1 and P2.

In publication P1 we show an example of classical state processing approach to sequential data modeling based on delta images and convolutional neural networks. The proposed system is the first in the world fully automatic skin allergy recognition pipeline. It is able to automatically detect allergic skin reactions based on thermographic images and patient data. The proposed novel pipeline works in the following way:

1. Injection areas on the patient's forearms are marked with circular shapes.
2. Visible-spectrum and thermographic images of the patient's forearms are taken using a specialized device with correlated cameras and restraints, which help immobilize the forearm.
3. Medical personnel performs allergen injections in the marked areas.
4. Fifteen minutes after the injection, visible-spectrum and thermographic images of the forearms are taken once again.
5. Visible-spectrum images (both pre- and post-injection ones) are segmented based on the marks on the skin using the U-Net network (Ronneberger et al., 2015). This process creates reference points on the forearms.
6. Custom heuristic algorithm is used to fill in the gaps in the marks grid
7. Pre- and post-injection images are correlated by finding a transformation matrix between images based on the reference points. The intuition behind this step is to estimate the change in the position of the patient's forearm between two sets of images. We propose two different approaches to this step, one using a homography matrix and the other one based on linear regression, translations, and rotations.
8. Delta thermal images are created by subtracting aligned pre-injection images from post-injection ones. This step can be regarded as a fixed, simple state transformation between two time steps in the series.
9. As each image contains eight injection areas, the images (both post-injection visible-spectrum one and delta thermal one) are segmented using reference points into small regions of interest.
10. Convolutional neural network is used to classify each injection area. We test several approaches regarding the network's input — single image (either visible-spectrum one or delta thermal one) or both images. Regardless of the case, essential patient attributes are also fed to the model, containing information regarding sex, age, weight, and body temperature.

To test our approach, a clinical study has been conducted. One hundred patients underwent skin prick allergy tests with 12 allergens, totaling 404 thermal and 404

visible-spectrum images. After the segmentation phase and filtering out technically-invalid ones, a total of 1584 allergic reaction samples were obtained. During our studies, we tested both the segmentation method, as well as the allergic reaction classifier. We arrived at the following conclusions:

- Thermographic images can effectively be used for the recognition of allergies in humans.
- sequential data can be modeled by the CNN architecture using naive step-to-step processing.
- Proposed custom neural architecture outperforms standard state-of-the-art convolutional neural networks due to the stability of the training.
- The proposed system is the first automatic and objective approach to skin allergic reaction classification.

The above work uses naive, fixed-state modeling between time steps. In most cases, however, such a simple approach is not feasible. In the article P2 we present a novel RNN architecture, which allows for a significantly more elaborate, adaptive way of state modeling. The recurrent cell, called Deep Memory Update (DMU), can perform the arbitrary nonlinear transformation of its memory state. Additionally, it is able to alleviate gradient propagation issues while using only a single gate. The proposed unit consists of the feedforward neural network (FNN) and the memory cell. FNN processes the state and the input, and its output is fed into the memory cell in the following manner:

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

where "$\odot$" denotes the Hadamard product, $x_t, z_t, \widehat{h}_t$ are the input, memory preservation vector, and direction of the state change at time step t respectively, h_t is both the memory state and output at time step t, $\mathbf{1}$ is a vector of ones, σ is a unipolar soft step function, e.g., the logistic sigmoid, and f is an activation function, e.g., hyperbolic tangent. The usage of FNN with at least two dense layers allows the state to be arbitrarily processed, with the depth of the network dictating the depth of the state transformation. DMU has fewer weights per memory cell in its simplest form compared to other state-of-the-art gated RNN architectures such as GRU, LSTM, or RHN. This is achieved by using a single gate within the cell.

To further improve the network's stability and reduce convergence time, we proposed an initialization strategy for the unit, as well as learning rate scaling for the entire neural model incorporating DMU cells. During initialization, we recommended setting the bias for the z_t vector to a high value, which allows the network to mostly preserve the memory state between the time steps. In our experimental setting, we combined DMU cells with a dense layered subnetwork for the output. We proposed the following learning rate scaling:

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

where β is the learning rate for the output subnetwork, and n is the depth of DMU's FNN.

We theoretically showed that DMU is safe from gradient propagation issues. We tested the proposed architecture and compared it to state-of-the-art competitors on three synthetic tasks and four problems based on real-world data. The former were proposed by Hochreiter and Schmidhuber (1997) and the latter are polyphonic music modelling (Boulanger-Lewandowski et al., 2012), natural language modelling (Zaremba et al., 2014), machine translation (Tatoeba, 2020; ManyThings, 2020), and pixel-by-pixel ordered/permuted MNIST classification (Le et al., 2015). Additionally, we performed a scaled learning rate ablation to show the impact of the proposed scaling on the network results.

The results allowed us to conclude our study in the following ways:

- DMU is able to compete and often outperform state-of-the-art RNN architectures.
- The proposed module uses fewer parameters than LSTM, RHN, GRU, and other state-of-the-art architectures.
- Gradient propagation issues during DMU's training are alleviated by the design.
- Depth of the memory state transformation does not improve network's performance.
- Proposed learning rate scaling improves the results of the model, especially so for the deeper memory processing FNNs.

4. Novel Evolution Strategy-based neural network optimizer

In the previous chapter, we explored how specific neural architectures can solve gradient propagation issues. Alternatively, a second-order optimizer can be used to train arbitrary architectures. In this chapter, we explore such an optimizer, proposed in P3. The algorithm, called neural Differential Evolution Strategy (nDES), is based on a DES algorithm, which in turn is a crossover between DE and CMA-ES. We adapted DES to handle significantly larger search space, effectively allowing it to train neural networks.

In our approach, the algorithm uses a population of neural networks, each represented by a flattened, one-dimensional vector of parameters. We proposed a different initialization strategy for the population based on the Glorot and Bengio (2010) initialization. We introduced a batching technique to overcome the need to evaluate each individual on an entire dataset. The dataset is split into random batches prior to the training. During training, each individual gets assigned a batch in a cyclical manner. An Exponentially Weighted Moving Average (EWMA) of the fitness on the batch is tracked for each batch. This allows nDES to select the best individuals, i.e., the ones that yield better fitness values on the batch in comparison to its EWMA, while significantly reducing the time of the evaluation of the population. If EWMA batching is used, the result of the optimization process is selected in a different manner. During the late stages of training, most of the fitnesses will be near zero as the process converges, and most individuals cannot improve their score compared to the EWMA. In such cases, the final population is evaluated on the entire validation dataset, and the best-performing individual is selected as the result of the optimization process.

In some cases, the gradient propagation issues are absent, so the gradient information could be used without influencing the training stability. We provided a way to use this information to speed up the convergence of nDES. This is done by introducing an additional mutation, which we call gradient-based rotation. If this operator is used, an additional backpropagation pass is calculated during the evaluation of the individual. Next, a classic SGD step is taken to modify the weights with respect to the gradient information. This technique is especially useful for larger architectures where the size of the population is smaller than the total number of optimized parameters in the network. An intuition behind this method in the aforementioned case is the following: metaheuristic tries to find a solution on the sub-hyperplane spanned by the population. At the same time, the gradient mutation operator rotates this sub-hyperplane.

We evaluated nDES in the context of convolutional neural network training using FashionMNIST (Xiao et al., 2017) dataset. We compared the networks optimized by nDES with the ones optimized by Adam. Additionally, we have tested the performance of gradient mutation. Moreover, we proposed and evaluated the approach in which the network is pretrained using Adam and fine-tuned using nDES once Adam-based training

converges. Regarding sequential data processing, we compared Adam and nDES on addition, multiplication, and parity problems proposed by Hochreiter and Schmidhuber (1997). Notably, we have used Jordan's RNN in these tasks to further focus on the instability problems in classic recurrent architectures.

Based on the experimental results, we have reached the following conclusions:

- nDES is able to successfully compete with the state-of-the-art gradient optimizers.
- Using an evolution strategy on problems with orders of magnitude higher than previously reported is feasible with proposed modifications.
- nDES allows for training classic RNN architectures on problems with high instability factor, unlike the state-of-the-art first-order methods.
- Combining the second-order approach with gradient information yields the best results.
- Long running time of the training makes nDES unfeasible for larger architectures.

5. Discussion and final remarks

5.1. Conclusion

In this thesis, we investigated mitigation strategies against instabilities during training neural networks using sequential data. We have used fixed state processing and custom CNN architecture to create the first fully automatic and objective skin allergic reaction classification method. Next, we introduced a novel least redundant gated recurrent neural network called DMU and showed that it often outperforms state-of-the-art competitors while achieving a reduction in the number of network's parameters. Additionally, we came to an interesting, counter-intuitive conclusion regarding the depth scaling of the state processing subnetwork in the proposed architecture. Finally, we proposed a new second-order optimizer nDES, which is based on the modified Evolution Strategy algorithm. We showed that by using nDES we can train vanilla RNN architectures to solve difficult sequential problems with significant internal state drift. All proposed solutions have been published and the code for nDES [1] and DMU [2] is freely available online.

5.2. Open questions

The introduction of methods described in this thesis led to several interesting future research opportunities, which we describe below.

How does DMU perform in combination with the attention mechanism? In this thesis, we put a great focus on gated recurrent neural architectures. As noted, they are usually outperformed by attention-based models such as transformers. Nevertheless, their low computational overhead merits their usage in settings with restricted hardware. Nevertheless, there are promising architectures, which combine the attention mechanism with RNN architectures (Wang et al., 2019; Lin et al., 2021; Liu et al., 2020). In our future research, we will evaluate the performance of the DMU architecture combined with the attention mechanism. In particular, we are interested in the effect of the depth of DMU's FNN on the performance of the entire architecture.

Why does DMU perform so well on synthetic tasks? In Chapter 3 we showed that DMU outperforms state-of-the-art alternatives such as GRU or LSTM, sometimes significantly so. The exact reason behind this performance gap is unknown. We want to better explore and understand DMU's training process on synthetic tasks. Gaining an insight into this optimization process could allow us to further improve the architecture.

[1] https://github.com/fuine/nDES/

[2] https://github.com/fuine/dmu

It should also give us a better understanding of the optimization process in the context of hard state drift problems.

How do models trained with nDES fare against adversarial attacks? The model's robustness against adversarial attacks has become an important issue in recent years, especially so in models deployed in the real world. As first-order optimization is currently the most popular way to train neural networks, the majority of attacks are prepared against networks trained in this manner. In future work, we will evaluate neural networks trained with nDES in the adversarial setting.

What is the possible speed-up of the nDES optimizer? As noted in Chapter 4, the main weakness of the nDES optimizer is the training time. However, the computational bottleneck of the algorithm, namely the evaluation loop, is embarrassingly parallel. We plan to rewrite this loop with the parallel execution and benchmark the resulting algorithm on multiple GPU units. We theorize that with such a rewrite we would be able to train some state-of-the-art neural architectures.

How would the introduction of deep state processing in our skin allergic recognition pipeline improve the results? In Chapter 3 we showed that a pipeline using a fixed, one-step state processing is able to correctly classify skin allergic reactions. However, the current approach has some limitations, namely the patient needs to wait roughly 15 minutes after injection for the assessment, and pre- and post-injection images need to be correlated with a heuristic approach, which is not perfect. To mitigate these issues we will explore the possibility of classification based on the video sequence of the reaction. Our hypothesis is that we can reduce the assessment time to approximately 5 minutes. Moreover, processing the video sequence with a deep neural network could improve the results as it should contain more information, particularly with regard to the dynamics of the blood flow around the injection point.

6. Academic Achievements

This thesis describes the academic achievements published in articles **[P1]**-**[P3]**. Additionally, the PhD candidate has the following academic achievements:

Articles

- **Neumann Ł.**, Nowak R., Okuniewski R., Wawrzyński P.: Machine Learning-Based Predictions of Customers' Decisions in Car Insurance, Applied Artificial Intelligence, vol. 33, no. 9, 2019, pp. 817-828, DOI:10.1080/08839514.2019.1630151
- Nowak R., **Neumann Ł.**, Franus W., Dąmbski M., Smółkowski A., Zawistowski P.: Machine learning models for predicting customer decision in motor claims settlements, In: Proceedings of SPIE: Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, vol. 11176, 2019, SPIE - The International Society for Optics and Photonics, ISBN 9781510630659, 111761U-1-111761U-6, DOI:10.1117/12.2536523
- **Neumann Ł.**, Nowak R.: Heuristic hyperparameter optimization for multilayer perceptron with one hidden layer, In: Proceedings of SPIE: Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018, vol. 10808, 2018, SPIE - The International Society for Optics and Photonics, ISBN 9781510622036, 108082A-1-108082A-8, DOI:10.1117/12.2501569
- Kuśmirek W., Nowak R., **Neumann Ł.**: New tool to assemble repetitive regions using next-generation sequencing data, In: Proceedings of SPIE: The International Society for Optical Engineering , vol. 10445, 2017, SPIE - The International Society for Optics and Photonics, ISBN 978-151061354-6, 104452V-1-104452V-8, DOI:10.1117/12.2280030
- **Neumann Ł.**, Nowak R., Kuśmirek W.: Katome: de novo DNA assembler implemented in rust, In: Proceedings of SPIE: Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2017, Proceedings of SPIE: The International Society for Optical Engineering , vol. 10445, 2017, SPIE - The International Society for Optics and Photonics, ISBN 978-151061354-6, 104451D-1-104451D-8, DOI:10.1117/12.2280206
- Cichosz P., Jagodziński D., Matysiewicz M., **Neumann Ł.**, Nowak R., Okuniewski R., Oleszkiewicz W.: Novelty detection for breast cancer image classification, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, pp. 1003135-1-1003135-12, DOI:10.1117/12.2249183
- Jagodziński D., Matysiewicz M., **Neumann Ł.**, Nowak R., Okuniewski R., Oleszkiewicz W., Cichosz P.: Feature selection and definition for contours classification of thermograms in breast cancer detection, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments

2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, 100312U-1-100312U-9, DOI:10.1117/12.2249064

- Matysiewicz M., **Neumann Ł.**, Nowak R., Okuniewski R., Oleszkiewicz W., Cichosz P., Jagodziński D.: Automatic recognition of thermographic examinations for early detection of breast cancer, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, 100312X-1-100312X-7, DOI:10.1117/12.2249067
- **Neumann Ł.**, Nowak R., Okuniewski R., Oleszkiewicz W., Cichosz P., Jagodziński D., Matysiewicz M.: Preprocessing for classification of thermograms in breast cancer detection, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, 100313A-1-100313A-8, DOI:10.1117/12.2249307
- Nowak R., Okuniewski R., Oleszkiewicz W., Cichosz P., Jagodziński D., Matysiewicz M., **Neumann Ł.**: Asymmetry features for classification of thermograms in breast cancer detection, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, 100312W-1-100312W-8, DOI:10.1117/12.2249066
- Okuniewski R., Nowak R., Cichosz P., Jagodziński D., Matysiewicz M., **Neumann Ł.**, Oleszkiewicz W.: Contour classification in thermographic images for detection of breast cancer, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, 100312V-1-100312V-8, DOI:10.1117/12.2249065
- Oleszkiewicz W., Cichosz P., Jagodziński D., Matysiewicz M., **Neumann Ł.**, Nowak R., Okuniewski R.: Application of SVM classifier in thermographic image classification for early detection of breast cancer, In: Proc. SPIE. 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, vol. 10031, 2016, SPIE , ISBN 9781510604858, 100312T-7-1-100312T-7-, DOI:10.1117/12.2249063

Additional 48 co-authored articles published as a member of the ALICE Collaboration.

Posters

- Neumann, Ł., Gambin, T., Kuśmirek, W., & Nowak, R. (2019). Deep learning approach to rare CNV detection, Polish Bioinformatics Society Symposium 2019
- Neumann, Ł., Gambin, T., Kuśmirek, W., & Nowak, R. (2019). Deep learning approach to rare CNV detection, Polskie Porozumienie na rzecz Rozwoju Sztucznej Inteligencji (PP-RAI) 2019
- Neumann, Ł., Lepak, Ł., & Wawrzyński, P. (2023). Least Redundant Gated Recurrent Neural Network, International Joint Conference on Neural Networks 2023

Projects

- Nucleotide methylation classification using machine learning methods on the Oxford Nanopore based data. Project leader: mgr inż. Neumann Łukasz, 01-09-2020 - 28-02-2021

- Recurrent neural networks for processing of sequential and graph data. Project leader: dr hab. inż. Wawrzyński Paweł, 01-01-2022 - 31-12-2023
- Search engine for job offers with automatic correlation of bidders and applicants. Project leader: mgr inż. Dariusz Jagodziński, 01-11-2021 - 31-12-2023
- Thermal and visual image analysis to classify skin alergic response using machine learning and computer vision algorithms. Project leader: dr hab. inż. Nowak Robert Marek, 15-06-2018 - 31-12-2020
- Higosense - telemedical system for remote pediatric diagnostics based on an innovative multi-sensory device and automatic interpretation algorithms. Project leader: dr hab. inż. Nowak Robert Marek, 25-01-2019 - 13-05-2019
- The use of artificial intelligence algorithms in the car insurance market to claims handling process. Project leader: dr hab. inż. Nowak Robert Marek, 04-09-2018 - 28-02-2019
- Machine learning algorithm to analyze auto insurance data. Project leader: dr hab. inż. Nowak Robert Marek, 04-09-2017 - 08-12-2017
- Thermographic image classification for early detection of breast cancer. Project leader: dr hab. inż. Nowak Robert Marek, 04-01-2016 - 01-12-2016

Bibliography

Ampazis, N. and Perantonis, S. J. (2002). Two highly efficient second-order algorithms for training feedforward networks. *IEEE Transactions on Neural Networks*, 13(5):1064–1074.

Anzengruber, F., Alotaibi, F., Kaufmann, L. S., Ghosh, A., Oswald, M. R., Maul, J.-T., Meier, B., French, L. E., Bonmarin, M., and Navarini, A. A. (2019). Thermography: High sensitivity and specificity diagnosing contact dermatitis in patch testing. *Allergology International*, 68(2):254 – 258.

Arabas, J. and Jagodziński, D. (2020). Toward a matrix-free covariance matrix adaptation evolution strategy. *IEEE Trans. on Evolutionary Computation*, 24(1):84–98.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128.

Ba, J., Grosse, R., and Martens, J. (2017). Distributed second-order optimization using kronecker-factored approximations. In *International Conference on Learning Representations*.

Baillie, A., Biagioni, P., FORSYTH, A., GARIOCH, J. J., and McPherson, D. (1990). Thermographic assessment of patch-test responses. *British Journal of Dermatology*, 122(3):351–360.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563.

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.

Berahas, A. S. and Takáč, M. (2020). A robust multi-batch l-bfgs method for machine learning. *Optimization Methods and Software*, 35(1):191–219.

Bilski, J., Kowalczyk, B., and Grzanek, K. (2018). The parallel modification to the levenberg-marquardt algorithm. In *Artificial Intelligence and Soft Computing: 17th International Conference, ICAISC 2018, Zakopane, Poland, June 3-7, 2018, Proceedings, Part I 17*, pages 15–24. Springer.

Bilski, J., Kowalczyk, B., Marchlewska, A., and Zurada, J. M. (2020). Local levenberg-marquardt algorithm for learning feedforwad neural networks. *Journal of Artificial Intelligence and Soft Computing Research*, 10(4):299–316.

Bollapragada, R., Nocedal, J., Mudigere, D., Shi, H.-J., and Tang, P. T. P. (2018). A progressive batching l-bfgs method for machine learning. In *International Conference on Machine Learning*, pages 620–629. PMLR.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*.

Box, G. E. and Jenkins, G. M. (1970). Time series analysis forecasting and control. Technical report, WISCONSIN UNIV MADISON DEPT OF STATISTICS.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P.,

Lee, Y. T., Li, Y., Lundberg, S., et al. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Cauchy, A. et al. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538.

Chang, B., Chen, M., Haber, E., and Chi, E. (2019). AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*.

Chiara, M., D'Erchia, A. M., Gissi, C., Manzari, C., Parisi, A., Resta, N., Zambelli, F., Picardi, E., Pavesi, G., Horner, D. S., et al. (2021). Next generation sequencing of sars-cov-2 genomes: challenges, applications and opportunities. *Briefings in Bioinformatics*, 22(2):616–630.

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*.

Choi, K., Fazekas, G., and Sandler, M. (2016). Text-based lstm networks for automatic music composition. *arXiv preprint arXiv:1604.05358*.

Chowdhary, K. (2020). Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. In *ICML*, pages 2067–2075.

Ciccone, M., Gallieri, M., Masci, J., Osendorfer, C., and Gomez, F. (2018). Nais-net: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems*, volume 31, pages 3025–3035.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. (2016). Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*.

Curry, H. B. (1944). The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261.

Dagum, P., Galper, A., and Horvitz, E. (1992). Dynamic network models for forecasting. In *Uncertainty in artificial intelligence*, pages 41–48. Elsevier.

Dagum, P., Galper, A., Horvitz, E., and Seiver, A. (1995). Uncertain reasoning and forecasting. *International Journal of Forecasting*, 11(1):73–87.

Daskalakis, C. and Panageas, I. (2018). The limit points of (optimistic) gradient descent in min-max optimization. *Advances in neural information processing systems*, 31.

Dey, R. and Salem, F. M. (2017). Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE.

Dietterich, T. G. (2002). Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*, pages 15–30. Springer.

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.

Erichson, N. B., Azencot, O., Queiruga, A., Hodgkinson, L., and Mahoney, M. W. (2020). Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*.

Fathi, M., Haghi Kashani, M., Jameii, S. M., and Mahdipour, E. (2022). Big data analytics

in weather forecasting: A systematic review. *Archives of Computational Methods in Engineering*, 29(2):1247–1275.

Freeth, T., Bitsakis, Y., Moussas, X., Seiradakis, J. H., Tselikas, A., Mangou, H., Zafeiropoulou, M., Hadland, R., Bate, D., Ramsey, A., et al. (2006). Decoding the ancient greek astronomical calculator known as the antikythera mechanism. *Nature*, 444(7119):587–591.

Gauss, C. F. (1809). *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser.

Gers, F. A. and Schmidhuber, E. (2001). Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE transactions on neural networks*, 12(6):1333–1340.

Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002). Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Granger, C. W. and Joyeux, R. (1980). An introduction to long-memory time series models and fractional differencing. *Journal of time series analysis*, 1(1):15–29.

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv:1308.0850.

Gupta, A., Müller, A. T., Huisman, B. J., Fuchs, J. A., Schneider, P., and Schneider, G. (2018). Generative recurrent networks for de novo drug design. *Molecular informatics*, 37(1-2):1700111.

Gurjarpadhye, A. A., Parekh, M. B., Dubnika, A., Rajadas, J., and Inayathullah, M. (2015). Infrared imaging tools for diagnostic applications in dermatology. *SM journal of clinical and medical imaging*, 1(1):1.

Hansen, C., Hansen, C., Maystre, L., Mehrotra, R., Brost, B., Tomasi, F., and Lalmas, M. (2020). Contextual and sequential user embeddings for large-scale music recommendation. In *ACM Conf. on Recommender Systems*, pages 53–62.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Heinzerling, L., Mari, A., Bergmann, K.-C., Bresciani, M., Burbach, G., Darsow, U., Durham, S., Fokkens, W., Gjomarkaj, M., Haahtela, T., et al. (2013). The skin prick test–european standards. *Clinical and translational allergy*, 3(1):3.

Hewamalage, H., Bergmeir, C., and Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427.

Hochreiter, S., Kolen, J. F., and Kremer, S. C. (2001). *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. Wiley-IEEE Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hosking, J. R. M. (1981). Fractional differencing. *Biometrika*, 68(1):165–176.

Jagodziński, D. and Arabas, J. (2017). A differential evolution strategy. In *2017 IEEE congress on evolutionary computation (CEC)*, pages 1872–1876. IEEE.

Jia, Z., Lin, Y., Wang, J., Feng, Z., Xie, X., and Chen, C. (2021). Hetemotionnet: Two-stream heterogeneous graph recurrent neural network for multi-modal emotion recognition. In *ACM Int. Conf. on Multimedia*, pages 1047–1056.

Jordan, M. I. (1986). Serial order: A parallel, distributed processing approach. *Advances in Connectionist Theory Speech*, 121(ICS-8604):471–495.

Justo, X., Díaz, I., Gil, J., and Gastaminza, G. (2016). Prick test: evolution towards automated reading. *Allergy*, 71(8):1095–1102.

Kag, A. and Saligrama, V. (2021). Training recurrent neural networks via forward propagation through time. In *ICML*, pages 5189–5200.

Kag, A., Zhang, Z., and Saligrama, V. (2020). RNNs incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. In *ICLR*.

Koumakis, L. (2020). Deep learning models in genomics; are we there yet? *Computational and Structural Biotechnology Journal*, 18:1466–1473.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Lambert, C., Sarrat, A., Bienvenu, F., Brabant, S., Nicaise-Roland, P., Alyanakian, M.-A., Apoil, P.-A., Capron, C., Couderc, R., Evrard, B., et al. (2015). The importance of en iso 15189 accreditation of allergen-specific ige determination for reliable in vitro allergy diagnosis. *Allergy*, 70(2):180–186.

Larenas-Linnemann, D., Luna-Pech, J. A., and Mösges, R. (2017). Debates in allergy medicine: Allergy skin testing cannot be replaced by molecular diagnosis in the near future. *World Allergy Organization Journal*, 10(1):32.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Lesk, A. M. (2017). *Introduction to genomics*. Oxford University Press.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168.

Levy, J. J., Titus, A. J., Petersen, C. L., Chen, Y., Salas, L. A., and Christensen, B. C. (2020). Methylnet: an automated and modular deep learning approach for dna methylation analysis. *BMC bioinformatics*, 21(1):1–15.

Lin, J. C.-W., Shao, Y., Djenouri, Y., and Yun, U. (2021). Asrnn: A recurrent neural network with an attention model for sequence labeling. *Knowledge-Based Systems*, 212:106548.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Liu, Z., Lu, C., Huang, H., Lyu, S., and Tao, Z. (2020). Hierarchical multi-granularity attention- based hybrid neural network for text classification. *IEEE Access*, 8:149362–149371.

Llugsi, R., El Yacoubi, S., Fontaine, A., and Lupera, P. (2021). Comparison between adam, adamax and adam w optimizers to implement a weather forecast based on

neural networks for the andean city of quito. In *2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6. IEEE.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization.

Manning, C. D. (2022). Human language understanding & reasoning. *Daedalus*, 151(2):127–138.

ManyThings (2020). http://www.manythings.org/anki/. Retrieved 2020-05-05.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441.

Martens, J., Ba, J., and Johnson, M. (2018). Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*.

Martens, J. et al. (2010). Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

Mor, B., Garhwal, S., and Kumar, A. (2021). A systematic review of hidden markov models and their applications. *Archives of computational methods in engineering*, 28:1429–1448.

Moreno, E. M., Moreno, V., Laffond, E., Gracia-Bara, M. T., Muñoz-Bellido, F. J., Macías, E. M., Curto, B., Campanon, M. V., de Arriba, S., Martin, C., et al. (2020). Usefulness of an artificial neural network in the prediction of β-lactam allergy. *The Journal of Allergy and Clinical Immunology: In Practice*, 8(9):2974–2982.

Mozer, M. C. (1995). A focused backpropagation algorithm for temporal. *Backpropagation: Theory, architectures, and applications*, 137.

Nelson, H. S., Lahr, J., Buchmeier, A., and McCormick, D. (1998). Evaluation of devices for skin prick testing. *Journal of allergy and clinical immunology*, 101(2):153–156.

Nesterov, Y. (2003). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.

Nguyen, T. M., Baraniuk, R. G., Bertozzi, A. L., Osher, S. J., and Wang, B. (2020). Momentumrnn: Integrating momentum into recurrent neural networks. *arXiv preprint arXiv:2006.06919*.

Oore, S., Simon, I., Dieleman, S., Eck, D., and Simonyan, K. (2020). This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, 32:955–967.

OpenAI (2023). Gpt-4 technical report. *ARXIV.ORG*.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2014). How to construct deep recurrent neural networks. In *ICLR*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318.

Ramirez-GarciaLuna, J. L., Rangel-Berridi, K., Bartlett, R., Fraser, R. D., and Martinez-Jimenez, M. A. (2022). Use of infrared thermal imaging for assessing acute inflammatory changes: A case series. *Cureus*, 14(9).

Rechenberg, I. (1973). Evolutionsstrategie. *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution*.

Risi, S. and Togelius, J. (2015). Neuroevolution in games: State of the art and open chal-

lenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41.

Robinson, A. J. and Fallside, F. (1987). The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University, Engineering Department.

Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, Heidelberg.

Rok, T., Rokita, E., Tatoń, G., Guzik, T., and Śliwa, T. (2017). Thermographic imaging as alternative method in allergy diagnosis. *Journal of Thermal Analysis and Calorimetry*, 127(2):1163–1170.

Rok, T., Rokita, E., Tatoń, G., Guzik, T., and Śliwa, T. (2016). Thermographic assessment of skin prick tests in comparison with the routine evaluation methods. *Advances in Dermatology and Allergology / Postępy Dermatologii i Alergologii*, 33(3):193–198.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing.

Routhier, E. and Mozziconacci, J. (2022). Genomics enters the deep learning era. *PeerJ*, 10:e13613.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Rusch, T. K., Mishra, S., Erichson, N. B., and Mahoney, M. W. (2022). Long expressive memory for sequence modeling. In *International Conference on Learning Representations*.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Schwefel, H.-P. and Schwefel, H.-P. (1977). *Evolutionsstrategien für die numerische optimierung*. Springer.

Shanthi, T., Sabeenian, R., and Anand, R. (2020). Automatic diagnosis of skin diseases using convolution neural network. *Microprocessors and Microsystems*, 76:103074.

Shir, O. M. and Yehudayoff, A. (2020). On the covariance-hessian relation in evolution strategies. *Theoretical Computer Science*, 801:157–174.

Stanev, E., Dencheva, M., Lyapina, M., and Forghani, P. (2020). Thermographic examination of prick test reactions with local anesthetic. *Journal of Thermal Analysis and Calorimetry*, 140(1):225–231.

Stanley, K., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1:24–35.

Storn, R. and Price, K. (1995). Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *J. of Global Opt.*, 23.

Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

Tang, Z., Jiang, F., Gong, M., Li, H., Wu, Y., Yu, F., Wang, Z., and Wang, M. (2021). Skfac:

Training neural networks with faster kronecker-factored approximate curvature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13479–13487.

Tatoeba (2020). https://tatoeba.org. Retrieved 2020-05-05.

Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models. *ARXIV.ORG*.

Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y., and Singer, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(9).

Vanitha, N. and Geetha, M. (2021). A study on deep learning methods for skin disease classification. *International Journal of Engineering and Management Research*, 11(2):48–52.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NIPS*.

Wang, Z., Ma, Y., Liu, Z., and Tang, J. (2019). R-transformer: Recurrent neural network enhanced transformer. arXiv:1907.05572.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*. Survey Certification.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. ph. d. thesis, harvard university, cambridge, ma, 1974.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356.

Whittle, P. (1951). *Hypothesis testing in time series analysis*, volume 4. Almqvist & Wiksells boktr.

Wu, Y., Schuster, M., Chen, Z., Le, Q., Quoc, V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., and Gao, Q. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. arXiv:1409.2329.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2017). Recurrent highway networks. In *ICML*.

Appendices

A. List of Symbols and Abbreviations

ARFIMA – Autoregressive Fractionally Integrated Moving Average
ARIMA – Autoregressive Integrated Moving Average
ARMA – Autoregressive Moving Average
Adam – Adaptive Moment Estimation
BPTT – Backpropagation Through Time
CMA-ES – Covariance Matrix Adaptation Evolution Strategy
CNN – Convolutional Neural Network
CTRNN – Continuous-Time Recurrent Neural Networks
DBN – Dynamic Bayesian Network
DES – Differential Evolution Strategy
DE – Differential Evolution
DMU – Deep Memory Update
DT(S)-RNN – Deep Transition RNNs With Skip Connections
DT-RNN – Deep Transition RNN
EA – Evolutionary Algorithm
ES – Evolution Strategy
EWMA – Exponentially Weighted Moving Average
FNN – Feedforward Neural Network
GF-RNN – Gated-Feedback RNN
GRU – Gated Recurrent Unit
HMM – Hidden Markov Model
K-FAC – Kronecker-Factored Approximate Curvature
L-BFGS – Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm
LM – Levenberg–Marquardt Algorithm
LSTM – Long Short-Term Memory
LWIR – Low-Wavelength Infrared
nDES – neural Differential Evolution Strategy
RHN – Recurrent Highway Networks
RNN – Recurrent Neural Network
SGD – Stochastic Gradient Descent

B. List of Publications

B.1. Thermography based skin allergic reaction recognition by convolutional neural networks

Title	Thermography based skin allergic reaction recognition by convolutional neural netw
Authors	Łukasz Neumann, Robert Nowak, Jacek Stępień, Ewelina Chmielewska, Patryk Pankiewicz, Radosław Solan, Karina Jahnz-Różyk
Journal	Scientific Reports
Volume	12
Year	2022
DOI	10.1038/s41598-022-06460-9
Ministerial score	140
Pages	1 - 10

www.nature.com/scientificreports

scientific reports

Check for updates

OPEN

Thermography based skin allergic reaction recognition by convolutional neural networks

Łukasz Neumann[1]✉, Robert Nowak[1], Jacek Stępień[2], Ewelina Chmielewska[1], Patryk Pankiewicz[1], Radosław Solan[2] & Karina Jahnz-Różyk[3]

In this work we present an automated approach to allergy recognition based on neural networks. Allergic reaction classification is an important task in modern medicine. Currently it is done by humans, which has obvious drawbacks, such as subjectivity in the process. We propose an automated method to classify prick allergic reactions using correlated visible-spectrum and thermal images of a patient's forearm. We test our model on a real-life dataset of 100 patients (1584 separate allergen injections). Our solution yields good results—0.98 ROC AUC; 0.97 AP; 93.6% accuracy. Additionally, we present a method to segment separate allergen injection areas from the image of the patient's forearm (multiple injections per forearm). The proposed approach can possibly reduce the time of an examination, while taking into consideration more information than possible by human staff.

According to the World Health Organization (WHO), allergy is the third most common disease, it is classified as a threat to civilization. The twenty-first century is called the age of allergy epidemics. Experts at the European Academy of Allergy and Clinical Immunology predict that by 2025 over 50% of the European population will suffer from various types of allergies[1]. Unfortunately, more than half of allergy sufferers lack proper diagnosis and, consequently, also treatment[2]. In this context, a significant problem is the insufficient number of allergy specialists, for which effective compensation is possible by increasing the degree of automation of allergy diagnosis as well as by transferring tests from specialized centers to doctors and primary care centers.

Along with medical history, allergy tests may be able to confirm whether a particular substance is causing symptoms. Currently in clinical practice the widely used in vitro laboratory test allows indirect diagnostics of only one type of allergy by determining the level of specific IgE antibodies in human serum or plasma involved in the type I (immediate) allergic reaction[3]. The second technique is an in vivo functional method in the form of skin tests, allowing for the diagnosis of both type I (skin prick test—SPT) and type IV (delayed - patch test) allergies[4]. In this method tiny amounts of allergen are dropped into a small puncture made by a lancet. The area around the puncture is then observed for signs of an allergic reaction. The wheal and erythema regions are marked and measured using a ruler and related by trained professionals to the size of the histamine control and reported on a 0-5 scale. Because skin tests of various types allow mapping the actual response of the patient's immune system to the allergen it is assumed that they are a clinically reliable predictor of individual hypersensitivity to the tested substances[5].

Unfortunately, both methods have limitations, can lead to wrong diagnoses, are invasive and require trained professionals to perform the test and to interpret results.

The skin test results are observed by the doctor and, based on a subjective visual assessment of the symptoms, associated with an allergic skin reaction. This applies the difficulty of establishing a standardized measure for the observed key skin allergic reaction in the form of the so-called allergic bubble. The bubble size depends on the technique of puncturing the patient's skin during allergen application[6]. Moreover, a bubble is formed under the influence of fluid penetration from the skin's microvessels dilated by the released histamine, but it may also appear in other situations such as toxic dermatitis or even at the site of a negative test; a bubble is non-specific symptom.

In the last decade, new methods have been intensively sought to produce readings of skin allergy tests, using reliable markers that ensure repeatability of the results. There is significant potential for non-invasive far infrared imaging (LWIR), for which relatively shallowly located allergic skin reactions are completely transparent and can easily be recorded with appropriately sensitive instruments[7]. Infrared imaging has been helpful in non-invasive diagnosis of breast cancer, melanoma skin cancer, and skin burns[8].

[1]Institute of Computer Science, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland. [2]Milton Essex S.A., ul. J. P. Woronicza 31/348, 02-640 Warsaw, Poland. [3]Military Institute of Medicine, ul. Szaserów 128, 04-141 Warsaw, Poland. ✉email: lukasz.neumann@pw.edu.pl

LWIR skin imaging enables recording the hyperthermic allergic reaction accompanying the posthistamine perfusion effect)[9]. The sensitivity, specificity and accuracy of thermovision tests were assessed as high[10], with very high agreement between the reference method used in the clinic and in vitro sIgE results.

Our solution uses a visible and an infrared camera simultaneously, therefore allergic bubble and an skin allergic reaction could be both observed. The data are collected using a new device, where visible and thermal pictures are calibrated at pixel level. The analysis uses a series of pictures taken both before and after applying the allergens. Moreover, the general medical investigation data, like age, gender, body temperature, are used.

The system works on the images of the entire forearm, with multiple allergens per forearm. The algorithms generate fragments of image for single application, called segments, and classify each of them separately. The segmentation process is based on U-Net architecture, while classification model is a custom convolutional architecture.

To the best of our knowledge, there are not any comparable state-of-the-art solutions. Some artificial intelligence models supporting allergists are available. For example, in[11] neural network supports the diagnosis of the β-Lactam Allergy. Visible-spectrum images were classified by neural network models to detect more widespread skin diseases[12,13]. There were attempts to correlate thermal and allergic reactions[10,14,15], but they were purely exploratory and did not introduce an end-to-end automatic approach to recognizing allergic reactions. One study that attempted to classify reactions used the patch test approach in which allergen-soaked pads were put on the patient's back, and the underlying skin was analyzed using FLIR ONE application[16]. Notably, both studies were done on datasets substantially smaller than ours.

The contribution of this paper can be summarized in the following points:

(I) Usage of thermographic images to the recognition of allergies in humans;
(II) Novel, automatic, and objective method to mark allergies in prick tests;
(III) Comparison of different light spectras (thermal and visible) as an input to the neural network classifier.

We show that using the proposed method we've obtained nearly perfect results in terms of AUC (0.98) and AP (0.97). As previously stated, there are not any state-of-the-art solutions. One similar research tested patch-based approach and reported significantly worse results (AUC 0.85). However, it is unclear if these methods can be directly compared, as they differ in the way allergen is introduced to the patient's body.

This article is organized as follows. "Input" section depicts the input to the system. "Segmentation method" section depicts the segmentation algorithm, "Classification methods" section provides the neural network description used for classification, as well as training and validating techniques. "Results" section describes the dataset of thermal and visual images with full medical documentation, as well as the results. Finally, the discussion is provided in "Discussion" section.

Input

The system takes three main components as an input, namely a pair of thermographic images, a pair of visible-spectrum images and tabular data. All of them show the patient's forearms at two moments in time, called a series, before allergen application and 15 min after allergen application.

Visible-spectrum images and thermographic ones, which are taken at the same time, are correlated on the pixel level. There is not such correlation between series. Device used to produce these images is described in "Dataset" section. Samples of the images are shown in Fig. 1.

Thermographic photos have a single channel. Each pixel corresponds to a temperature reading from the thermographic camera. Visible spectrum images focus on a patient's skin and hives which in some cases appear as result of allergen injections. The second use of these images is to localize areas where allergens have been injected, using marks drawn on the arm by the person performing the examination.

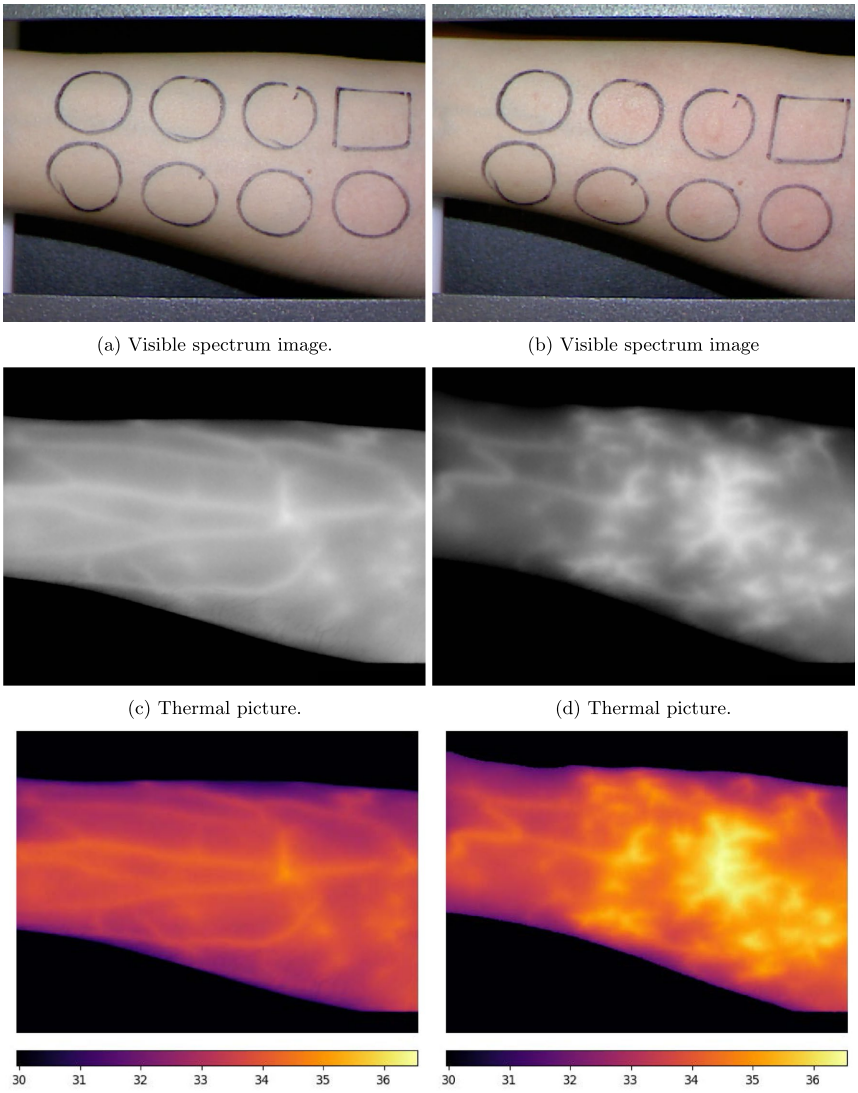
The data describes patient's sex, age, weight, height and body temperature measured at the beginning of an examination.

Segmentation method

Segmentation problem overview. The purpose of the segmentation process is to find the general areas in which allergens were applied, because the dataset does not contain such information. It's necessary to find or approximate the positions of all allergens on the forearm, because the ground truth is directly tied to each allergen separately and the classifier evaluates each allergen separately. The positions of the allergens are marked on the skin on the patient forearms' as squares (histamine) and circles (other allergens and negative control) by a nurse before the examination. As patients' forearms differ in shape it is not possible to use a fixed grid pattern to find allergen positions. Moreover, a patient has the ability to slightly change the angle of the forearm once it is fixed in the machine, as well as move it forwards/backwards (that is along the horizontal axis on the images). This is depicted in Fig. 1a,b.

The main segmentation problems are hairy forearms, incomplete marking for application positions, and different marker colors. Due to these problems classic segmentation methods, such as color-based segmentation or Canny edge detection[17], do not yield satisfactory results.

U-Nets. We use the U-Net[18] model as a basis for the segmentation process. We used the original architecture proposed by the authors. We manually mark all the allergen application areas by saving the injection point and a point on the highlighter marker. Based on this information we create binary masks, where each application area is marked by a circle. This is a simplification, as the real shapes of highlighter markers differ slightly (most notably rectangle for the histamine), however the resulting model yields acceptable results, as depicted in Fig. 2.



(a) Visible spectrum image. (b) Visible spectrum image

(c) Thermal picture. (d) Thermal picture.

(e) Colored thermal picture. (f) Colored thermal picture.

Figure 1. Photos of patient's forearm before (**a**, **c**, **e**) and after (**b**, **d**, **f**) allergen application. Colorbars in (**e**, **f**) show the temperature scale in Celsius degrees.

We use the ADAM[19] optimizer with binary cross-entropy (BCE) loss. The model is trained on images downscaled to 512 × 385 (that is, halved in size) with pixel values normalized to the range [0, 1]. We train the model for 30 epochs.

Images that are the result of U-Net segmentation are then fed to the contour-searching algorithm[20]. Next, we filter out contours with appropriate size and shape (based on the bounding box).

Finally, in cases where not all markers are found, we use a heuristic approach to approximate the allergen grid based on the segments found and fill in the missing ones. The algorithm chooses segments using contours

and bounding rectangles with appropriate ratio and size. Next, segments are arranged in the grid by splitting them into two lines and finding proper positions by taking the offset from the right side. After that, the process searches for a minimal length vector between the positioned segments and fills in the missing ones, adding such vectors to the center of neighboring segments.

Classification methods

Delta thermal images. Thermal images tend to contain features that can be indistinguishable from allergic reaction (e.g. blood vessels). We call image created by subtracting pre-reaction image from the post-reaction one a 'delta' thermal image. Ideally such image should contain only thermal changes caused by the allergic reaction, assuming that patient's environment does not change. To create delta image pre- and post-reaction images need to be aligned. This is because it is not possible to take an image of a patient's forearm in exactly the same position two separate times. Patients have some leeway to change the angle of the forearm, they can also rotate it slightly, as well as position it differently with respect to the arm restraints placed near the elbow and wrist of the forearm. This effect can be observed in Fig. 1a,b.

We tested two approaches to image alignment. In both techniques we exploit the fact that after the segmentation phase we have eight separate markers per image. The basic premise is to find a transform which would align two sets of markers between pre-reaction and post-reaction images. Note that this requires sensible segmentation results - in order for this approach to work we need to ensure that enough markers are found on both images and that these are matching markers. The more markers are found the better the alignment results.

In the first technique a homography matrix is estimated based on the sets of markers.

The second approach is simpler in terms of degrees of freedom. First, linear regression is fitted on each set of markers. This gives us two lines, each for pre- and post-reaction images. Next, a transformation consisting of translations and rotations is calculated. This transformation should minimize the distance between two lines.

Regardless of the technique used, once the transformation matrix is estimated, it is used to transform the pre-reaction image. Finally, a delta thermal image is created by means of subtraction.

Classifier input, preprocessing and augmentation. The classification model works on a single segment images, that is for each found segment (allergen) on an image we cut out a separate region of interest and save it. We use segments found on the visible-spectrum images to cut out matching segments on thermal images, under the assumption that corresponding visible-spectrum and thermal images are pixel-perfectly correlated. For each segment found, the region of interest (ROI) is a 300×300 square, with the center taken from the center of the segment. An average size of the bounding box for the segment is 85 pixels in our dataset ($\sigma = 6$, min 60, max 104), so ROI includes a substantial padding. This helps us to account for the minor inaccuracies in the segmentation method, as well as add context to information about the surroundings for the network. See "Discussion" section for more information about why this is important.

A sample in the dataset for the classification task consists of patient's attributes and two 300×300 images—a visible-spectrum one and a corresponding delta image. Attributes used for the classification are recorded from an interview—sex, age, weight and body temperature. Note that all samples for a single patient will share the values of these attributes, as they are not allergen specific. The ground truth for each sample is a binary label created based on the diagnosis of doctor-allergologist.

Patients' attributes are rescaled to the range [0, 1]. Images are normalized with respect to the mean and standard deviation. Statistics used to normalize and rescale data are calculated on the training dataset and used for the test dataset.

The following augmentation methods are used for the images: random horizontal and vertical flips, random rotation for up to 45 degrees, and random translation up to 4 pixels, random zoom up to 1.3 ratio. Each transformation has a 50% chance of being used with parameters for specific transformation drawn from uniform distribution. Notably, if the model is trained on both visible-spectrum and thermal images then images for each sample are augmented identically (i.e. each sample has two images and both of them will be augmented in the same way, both in terms of which augmentation methods are used and specific parameters for these transformations).

Neural classifier. Classification is based on a convolutional neural network. As described in "Classifier input, preprocessing and augmentation" section a single sample consists of two images—one with three channels and one single-channeled, together with a patient's attributes. In our experiments we test using either one of the images, or both. In the latter case (both images used) we fuse images together after preprocessing and augmentation steps into a single four-channel tensor. The patient's attributes are fed into the first fully-connected layer.

The proposed architecture of the convolutional subnetwork is depicted in Table 1. We use average pooling layers with 3×3 kernel and stride 2. All convolutional layers use 3×3 kernels and are followed by batch normalization. The number of input channels is changed on the basis of the input data—four channels for both images used, one channel for thermal-only classification and three channels for visible-spectrum-only one. The fully connected subnetwork comprises of a Dense layer with 64 output channels followed by a LeakyReLU, Dropout and Dense layer with a single output channel.

We use Leaky-ReLU activation functions for all convolutional and dense layers except for the output layer, which uses softmax activation. ADAM optimizer with decoupled weight decay regularization[21] is used. Weight decay is set to 10^{-4}, while the negative slope of Leaky-ReLU is 10^{-2} and the learning rate is set to 0.001. The dropout value between two dense layers is 0.5. Cross-entropy loss is used as a loss function.

Training is early-stopped after 80 epochs, as the model has a tendency to overfit, as seen on Fig. 4.

The proposed architecture was selected as the best performing from among several state-of-the-art architectures: MobileNetV2[22], DenseNet-BC-76, DenseNet-BC-121[23], ResNet-18[24], where each architecture was used

Output channels	Pooling after convolution
32	Yes
64	Yes
64	
128	
128	Yes
256	
256	Yes
256	
256	Yes
256	
256	Yes

Table 1. Output channels for the convolutional part of the network. All layers (including pooling) have 3 × 3 kernels. All convolutional layers are followed with batch normalization and LeakyReLU activation.

without the original fully-connected layer(s). Instead their convolutional parts were used, with four-channel inputs. All models used the same fully-connected layers, as proposed in our architecture.

Ethics. The data collection procedure was approved by Komisja Bioetyki przy Wojskowym Instytucie Medycznym, protocol number 15/WIM/2017 on the 15.03.2017. The procedure was performed with accordance to the guidelines and regulations of the said commision. Informed consent was obtained from all participants.

Results

Dataset. The data were collected throughout years 2018–19 in the Clinic of Internal Medicine, Pneumonology, Allergology and Clinical Immunology of the Military Institute of Medicine, Warsaw, Poland. To illustrate allergic hyperthermia, the InfraScan^{TM} instrument developed by Milton Essex SA was used. Milton Essex SA is a company partially funding this research. InfraScan^{TM} has a high-resolution thermovision measuring system with an uncooled microbolometric matrix, enabling the recording of hyperthermic skin reactions with a diameter of less than 0.3mm. The instrument was developed for capturing allergic reactions, technical data of the thermovision system can be found in Supplementary Tables S1 and S2 online.

This device consists of a visible-light camera correlated with thermodetectors at pixel level and a frame with restraints near the patient's wrist and elbow. The restraints help immobilize the forearm and allow for smaller variations in the position/tilt of the forearm on the images. Despite these measures, patients still can slightly rotate the forearm and medical staff needs to make sure that restraints are put in similar places each time the image is taken.

To gather the images, first, the allergen application fields were marked on each of patient's forearms. Next, images (both thermal and visible-spectrum) of the patient's forearms were taken before the allergen application. The allergens were then applied, and the patient was asked to wait for 15 min in a sitting position. After this rest period, the images of both forearms were again taken. To account for various environmental factors the following steps were followed throughout the data gathering process:

1. Prior to the examination the device was placed in the examination room for 60 min to stabilize the temperature of its cover and detection hardware.
2. The examination was performed in a closed, air conditioned room without intensive air flow. The temperature and humidity in the room were stabilized and measured respectively 22 °C and 45%. Specialized air conditioning and humidifier systems were used to ensure stable conditions throughout the examination process.
3. Acquisition of images in each sequence was preceded by an automatic, single-point recalibration procedure. This procedure uses a blackbody temperature reference placed inside the head of the device. Additionally, it also corrects thermal drift based on data from sensors placed around microbolometric matrix and proprietary algorithms.
4. Patients were qualified for the examination based on the various medical premises. Additionally, all physiological aspects of the thermoregulation process were taken into account. These included drugs and other substances influencing body temperature and thermogenesis. Before the examination, each patient spent 15 min in a waiting room with a specialized air-conditioning humidification system with a temperature and humidity precisely like the temperature and humidity in the examination room. During the examination, the patient was in the examination room.

The details of the data gathering process were described in the study protocol MESX/ISC/09/18 (Infrared Imaging of Field of Allergic Reaction).

The dataset consists of interviews and photographs gathered from 100 participants (47 men and 53 women), who underwent a series of skin prick allergy tests with 12 allergens. In total 404 thermal and 404 visible spectrum images were gathered. All photos are sized 1024 × 770 pixels. Samples were split equally between two regions: right and left hand. For control purposes, a histamine (positive sample) and negative control sample were used

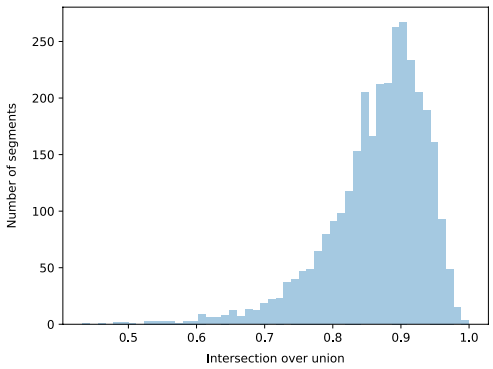


Figure 2. Normalized intersection area between manual and predicted segments.

along with allergens. In total eight allergen application fields were placed on the patient's forearm, which resulted in 32 samples from all regions and series. The whole dataset contains 1584 allergic reaction samples with 501 (31.6%) positive and 1083 (68.4%) negative cases. A total of 1600 segments were obtained, including hyperthermic allergic reactions as well as histamine and negative control, however some were discarded due to technical issues in the questionnaire.

The last part of the dataset is a collection of interviews. It contains basic information about each patient and diagnoses of allergic reactions. All the substances used, along with histamine and negative control samples, are listed in an interview. The region they are associated with and their position on the patient's forearm, which are part of the data collected, enable the matching of the doctor's diagnosis to fragments of images corresponding to proper reactions. This diagnosis is used to create binary ground-truth labels for the classification task.

To obtain segmentations for the entire dataset we run a leave-one-out cross-validation on the U-Net model and save the calculated segments.

Evaluation methods. We use Intersection over Union (IoU) to estimate segmentation results. For each segment we calculate the intersection area between a manual segmentation and an automatic one, as well as area of union. IoU is then calculated as $\frac{intersection}{union}$.

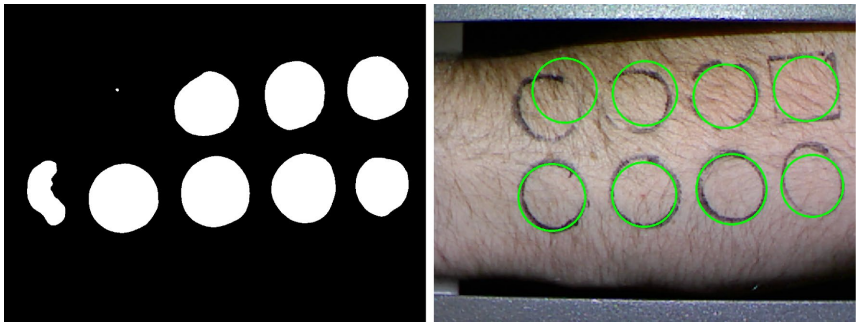
To evaluate our classification approach we use ten-fold cross-validation as well as leave-one-out cross-validation (LOOCV). To avoid a situation in which the model has seen other parts of the hand in the test set we can not simply stratify and fold over reactions. Instead we fold over patients, ensuring that the model has not seen either other parts of the hand or the patient's attributes. The mean and standard deviation used to normalize data are calculated on the training part of the data for each split. A model evaluation is based on the receiver operating characteristic (ROC) and precision-recall curve (PRC) and their corresponding numerical statistics—area under the curve (AUC) for ROC and average precision (AP) for PRC. Additionally, we check the accuracy of the model on a threshold that is selected to maximize the F1 score (in practice close to 0.5).

Segmentation. Results of segmentation are satisfactory, as presented in Figs. 2 and 3. In practice we do not need to achieve IoU of 1, as the manual segmentation is always a perfect circle, and due to scaling and arm position it will not always be the case. This fact partially explains why the IoU distribution is centered around 0.9. Moreover, as described in "Classifier input, preprocessing and augmentation" section we add a substantial margin around each segment found and so small deviations from the actual center of the segment should not influence the classification process too strongly.

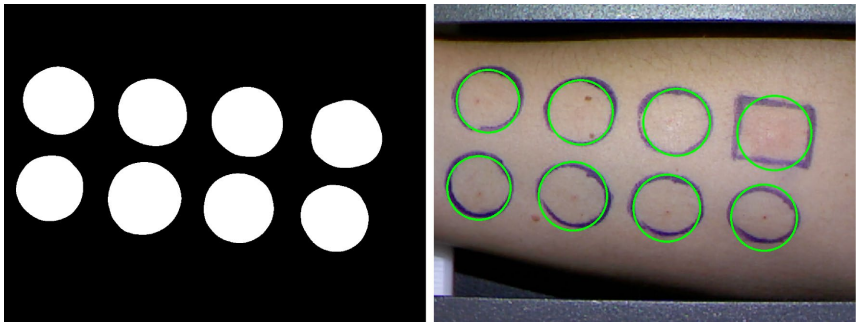
Classification. Overall we achieve roughly 93.5% accuracy on the dataset. Detailed results are depicted in Table 2. Notably, the ten-fold validation results are close to the leave-one-out validation results and so we report them for different experiments, as it takes considerably less time to calculate ten-fold validation results. Results indicate that U-Net based segmentation is comparable to manual segmentation. Interestingly, if the model is trained only on thermal images it performs as well as the one trained on both visible-spectrum and thermal images. A model that is trained on only visible-spectrum images performs worse (89.71% vs 93.24%) than the one trained on both types of images.

Loss and accuracy changes throughout the training process are shown in Fig. 4. They were calculated on a single validation split. Since the model has a slight tendency to overfit after approx. 100 epochs we use early stopping. Notably the exact moment of stopping is not as critical since the effect is gradual and there is a span of several dozen epochs during which the training process can be halted.

Precision recall curve and ROC curve are depicted in Fig. 5, while a confusion matrix for the thermal-only cross-validation results is presented in Table 3.



(a) UNet mask. (b) Segments.



(c) UNet mask. (d) Segments.

Figure 3. Sample segmentation results for problematic image. First row shows the result for a hairy forearm, while the second row shows the result for a different marker color (blue).

Input spectra						
Visible	Thermal	Segmentation	Validation	ROC AUC	PRC AP	Accuracy (%)
✓	✓	U-Net	Leave-one-out	0.975	0.956	93.50
✓	✓	manual	10-fold	0.970	0.952	92.79
✓	✓	U-Net	10-fold	0.978	0.961	93.24
✓	✗	U-Net	10-fold	0.940	0.880	89.71
✗	✓	U-Net	10-fold	**0.982**	**0.967**	**93.56**

Table 2. Validation results for different types of segmentation and input data. The best values are in bold.

Discussion

The results are promising in that we are able to achieve the same model performance both using only thermal images and thermal combined with visible spectrum images. Moreover, the model trained only on visible spectrum images yielded worse results. This indicates that thermographic photography can be used to classify allergic reactions. Additionally, this approach should work in the same way regardless of the skin color of the patient (assuming correct segmentation). One limitation of exclusively using thermal images is that the patient should have a normal heart and respiration rate, as high-intensity physical activity results in images similar to acute allergy reactions.

There are two main problems that influence the results of classification. Hair on the forearm is the first one, as it makes segmentation harder but notably more hair has distinctly lower temperature. This in turn means that, with thick enough hair, the image is somewhat obfuscated and it is harder to recognize allergic reaction patterns. The second problem stems from the fact that blood flows freely throughout the entire forearm, and so it happens

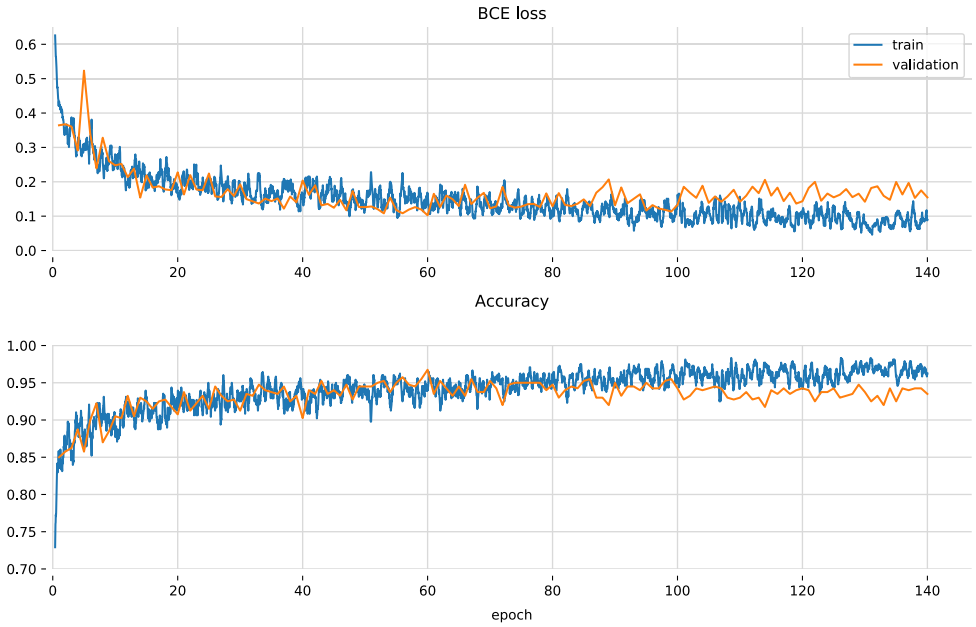
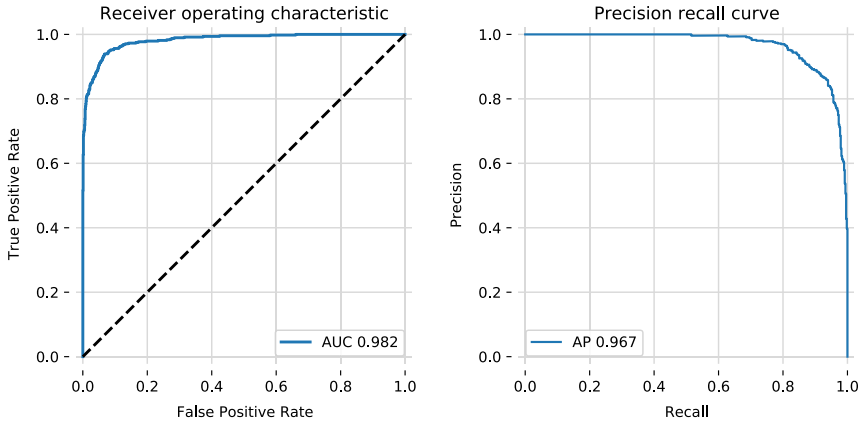


Figure 4. Loss and accuracy of the proposed model based on a single split in the cross-validation procedure.



(a) ROC curve (b) Precision recall curve

Figure 5. Both figures calculated on the results of the cross-validation procedure for the proposed model, which was built only on thermal images (last row in Table 2).

that an allergic reaction does not necessarily occur in the place of the injection. This effect can be seen in Fig. 1f, where it is not clear which allergen caused which micro-thermal surge in temperature.

It is possible that with a bigger dataset we could reduce the impact of these factors. One approach that we want to examine in the future is a model, which takes the image of the entire forearm as an input and predicts reaction for each allergen. Currently training such models is not feasible, as the dataset is too small in comparison to the size of a single image which should be used for such an experiment. Using entire-forearm images should help us

	Predicted diagnosis		
True diagnosis	Negative	Positive	Total
Negative	1042	61	1103
Positive	41	440	481
Total	1083	501	1584

Table 3. Confusion matrix for the ten-fold cross-validation, classifier built only on thermal images (last row in Table 2).

with the problem of blood flow. Another idea is to align images of the segments with respect to the blood flow. We experimented with that technique, but found that the results were worse, most probably due to the limited augmentation scope. If we align the images then random flips are no longer used and rotations are much smaller (between $[-10°, 10°]$). With such augmentations and the size of our current dataset model tended to aggressively overfit around the 20th epoch and achieved overall worse results. We want to note that with a bigger dataset this idea could actually boost the results.

In the future we also want to focus on researching the exact time that post-injection images are taken. The current allergological standard is the examination of patients after roughly 15 min post-injection. Our initial research shows that while this is the time required for the on-skin effects to show (i.e. hives and color change), it is not the case for the thermal reaction. We believe that it is possible to change the examination time to approximately 5 min, however this claim needs more research to validate it.

Additionally, we plan on assessing the robustness of the model with respect to high-frequency filtering. We theorize that it should be rather robust in this regard, since reactions do not have sharp edges and sudden gradient changes.

Received: 23 July 2021; Accepted: 31 January 2022
Published online: 16 February 2022

References

1. Calderon, M. A. *et al.* EAACI: A European declaration on immunotherapy. Designing the future of allergen specific immunotherapy. *Clin. Transl. Allergy* **2**(1), 20 (2012).
2. Muraro, A. *et al.* European symposium on precision medicine in allergy and airways diseases: Report of the European union parliament symposium (2015). *Allergy* **71**(5), 583–587 (2016).
3. Lambert, C. *et al.* The importance of EN ISO 15189 accreditation of allergen-specific IGE determination for reliable in vitro allergy diagnosis. *Allergy* **70**(2), 180–186 (2015).
4. Heinzerling, L. *et al.* The skin prick test-European standards. *Clin. Transl. Allergy* **3**(1), 3 (2013).
5. Larenas-Linnemann, D., Luna-Pech, J. A. & Mösges, R. Debates in allergy medicine: Allergy skin testing cannot be replaced by molecular diagnosis in the near future. *World Allergy Organ. J.* **10**(1), 32 (2017).
6. Nelson, H. S., Lahr, J., Buchmeier, A. & McCormick, D. Evaluation of devices for skin prick testing. *J. Allergy Clin. Immunol.* **101**(2), 153–156 (1998).
7. Justo, X., Díaz, I., Gil, J. J. & Gastaminza, G. Prick test: Evolution towards automated reading. *Allergy* **71**(8), 1095–1102 (2016).
8. Gurjarpadhye, A. A., Parekh, M. B., Dubnika, A., Rajadas, J. & Inayathullah, M. Infrared imaging tools for diagnostic applications in dermatology. *SM J. Clin. Med. imaging* **1**(1), 1 (2015).
9. Baillie, A. J., Biagioni, P. A., Forsyth, A., Garioch, J. J. & McPherson, D. Thermographic assessment of patch-test responses. *Br. J. Dermatol.* **122**(3), 351–360 (1990).
10. Rok, T., Rokita, E., Tatoń, G., Guzik, T. & Śliwa, T. Thermographic imaging as alternative method in allergy diagnosis. *J. Thermal Anal. Calorim.* **127**(2), 1163–1170 (2017).
11. Moreno, E. M. *et al.* Usefulness of an artificial neural network in the prediction of β-lactam allergy. *J. Allergy Clin. Immunol. Pract.* **8**(9), 2974–2982 (2020).
12. Shanthi, T., Sabeenian, R. S. & Anand, R. Automatic diagnosis of skin diseases using convolution neural network. *Microprocess. Microsyst.* **76**, 103074 (2020).
13. Vanitha, N. & Geetha, M. A study on deep learning methods for skin disease classification. *Int. J. Eng. Manag. Res.* **11**(2), 48–52 (2021).
14. Rok, T., Rokita, E., Tatoń, G., Guzik, T. & Śliwa, T. Thermographic assessment of skin prick tests in comparison with the routine evaluation methods. *Adv. Dermatol. Allergol.* **33**(3), 193–198 (2016).
15. Stanev, E., Dencheva, M., Lyapina, M. & Forghani, P. Thermographic examination of prick test reactions with local anesthetic. *J. Therm. Anal. Calorim.* **140**(1), 225–231 (2020).
16. Anzengruber, F. *et al.* Thermography: High sensitivity and specificity diagnosing contact dermatitis in patch testing. *Allergol. Int.* **68**(2), 254–258 (2019).
17. Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI–8**(6), 679–698 (1986).
18. Ronneberger, O., Fischer, P. & Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015* (eds Navab, N. *et al.*) 234–241 (Springer, 2015).
19. Kingma, D., & Adam, J. B.: A method for stochastic optimization. In *International Conference on Learning Representations*, vol. 12 (2014).
20. Suzuki, S. *et al.* Topological structural analysis of digitized binary images by border following. *Comput. Vis. Graph. Image Process.* **30**(1), 32–46 (1985).
21. Loshchilov, I., & Hutter, F. Decoupled weight decay regularization. arXiv:1711.05101 (2017).
22. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. -C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* 4510–4520 (2018).
23. Huang, G., Liu, Z., van der Maaten, L. & Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708 (2017).

24. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 (2016).

Acknowledgements
This work was partially funded by Milton Essex SA and Warsaw University of Technology Grant SzIR-1.

Author contributions
Ł.N. designed image alignment algorithm, neural classifier and U-Net segmentation. P.P. designed heuristic approach to approximate the allergen grid in the case of missing segments. Ł.N., R.N., P.P., and E.C. implemented the solution and carried out experiments. R.N. and J.S. contributed to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript. J.S. and R.S. gathered the data. R.S., K.J.R. assigned labels for data. All authors checked and approved manuscript.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-022-06460-9.

Correspondence and requests for materials should be addressed to Ł.N.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

B.2. Deep Neuroevolution: Training Neural Networks Using a Matrix-Free Evolution Strategy

Title	Deep Neuroevolution: Training Neural Networks Using a Matrix-Free Evolution Strategy
Authors	Dariusz Jagodziński, Łukasz Neumann, Paweł Zawistowski [1]
Conference	Neural Information Processing: 28th International Conference (ICONIP 2021)
Year	2021
DOI	10.1007/978-3-030-92185-9_43
Ministerial score	140

Deep Neuroevolution: Training Neural Networks using a Matrix-free Evolution Strategy

Dariusz Jagodziński[1][0000−0001−7938−5916], Łukasz Neumann[1][0000−0003−1759−3036], and Paweł Zawistowski[1][0000−0002−0273−7060]✉*

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, Warsaw, Poland
d.jagodzinski@elka.pw.edu.pl,
{lukasz.neumann,pawel.zawistowski}@pw.edu.pl

Abstract. In this paper, we discuss an evolutionary method for training deep neural networks. The proposed solution is based on the Differential Evolution Strategy (`DES`) – an algorithm that is a crossover between Differential Evolution (DE) and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We combine this approach with Xavier's coefficient-based population initialization, batch processing, and gradient-based mutations — the resulting weight optimizer is called neural Differential Evolution Strategy (`nDES`). Our algorithm yields results comparable to Adaptive Moment Estimation `ADAM` for a convolutional network training task (50K parameters) on the `FashionMNIST` dataset. We show that combining both methods results in better models than those obtained after training by either of these algorithms alone. Furthermore, `nDES` significantly outperforms `ADAM` on three classic toy recurrent neural network problems. The proposed solution is scalable in an embarrassingly parallel way. For reproducibility purposes, we provide a reference implementation written in Python.

Keywords: neuroevolution · neural network · deep learning · differential evolution · genetic algorithm.

1 Introduction

Deep artificial neural networks (DNNs) are among the most prominent breakthroughs in modern computer science. This success is largely founded on an effective way of establishing weights in DNNs. The backpropagation algorithm allows neural networks to be trained using a low computational cost gradient-based method. It enables all the weight updates to be simultaneously computed, using just one forward pass through the network, followed by a backward one.

Strong algorithmic efficiency orientation comes with several problems. The first-order methods used for weight optimization give no guarantees that the procedure will find the global minimum if the cost function is multimodal. Optimization may fall into the basin of attraction of one local optimum and prematurely

* All authors contributed equally.

converge. Computing the gradient one layer at a time and iterating backwards from the last may prevent the first layer's weights from changing the value of the gradient due to the vanishing gradient problem [11]. Gradient decreases exponentially with network depth, while the first layer's weight updates can be vanishingly small, or in some cases, the training process may completely stop. Backpropagation computes gradients by the chain rule, which leads to certain limitations of the neurons' activation functions [15, Chapter 7].

The restrictions imposed by the adopted learning method forced researchers to pursue new network layer types and connection schemes to address the vanishing gradient problem. This led to architectures like residual networks [10], or long short-term memory networks (LSTM) [12]. Also, standard activation functions like sigmoid and hyperbolic tangent ceased to be used in favour of the rectified linear unit (ReLU) [13], which has a constant derivative and has started to be the default activation when developing multi-layer perceptrons (MLPs).

Despite the progress, the root cause of the gradient optimization problems remains unchanged. Therefore, instead of struggling with the consequences of using a given optimization method, it might be more effective to use a different one altogether. One alternative lies in the field of neuroevolution [17], which uses evolutionary algorithms (EAs) to generate (or rather evolve) artificial neural networks.

This paper presents a neuroevolution method for deep neural network training based on an evolution strategy. We are motivated by previous research [1] that indicates the possibility of searching through the possible solution space with the contour fitting property without using any costly matrix algebra operations. Network parameters can be established according to the second-order optimization method based on the implicitly used Hessian matrix of DNNs' weight search space. The idea of this method is to use the Differential Evolution Strategy (`DES` [1]) algorithm, which is a crossover between Differential Evolution (DE) [18] and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [9], with the use of batch processing and gradient-based mutation. We call the method the neural Differential Evolution Strategy — `nDES`. The two main contributions of this work are as follows:

- the paper introduces a new ES called `nDES`, tailored to high dimensional optimization tasks typical for neural network training,
- we explore the effectiveness of batch-processing and gradient mutations in the proposed method,
- we empirically analyze the properties of `nDES` by evaluating it against the well established gradient-based `ADAM` optimizer and show that our method achieves superior performance on synthetic tasks, which are considered hard for gradient-based methods while obtaining results comparable to `ADAM` in settings "easier" for the latter method,

The paper is organized in the following way. In Section 2 we provide relevant information regarding `DES` and finally introduce `nDES`. The method is then verified empirically in experiments described and discussed in Section 3. Section 4 concludes the paper and describes possible directions for future work.

2 Proposed Method

This Section introduces the (`nDES`) method, an evolutionary neural network training approach. Before formulating `nDES`, we provide some necessary background information that underlines both the desirable properties and limitations of evolutionary strategies from the deep learning perspective. These prompt the need for a new method.

2.1 Groundwork

Evolution strategies are popular randomized search heuristics in $\mathbb{R}^n$ that excel in the global optimization of continuous landscapes. Their mechanism is primarily characterized by the mutation operator, whose variance is drawn from a multivariate normal distribution using an evolving covariance matrix. It has been hypothesized that this adapted matrix approximates the inverse Hessian of the search landscape. This hypothesis has been proven for a static model relying on a quadratic approximation [16]. The covariance matrix over selected decision vectors in strategy $(1, \lambda)$ has the same eigenvectors as the Hessian matrix. When the population size is increased, the covariance becomes proportional to the inverse of the Hessian.

The process of generating points with multivariate normal distribution is relatively costly. Evolution strategies with a covariance matrix estimation have an internal computational complexity of at least $\mathcal{O}(n^2)$. CMA-ES is a popular evolution strategy within the field of global optimization, and it has a basic complexity $\mathcal{O}(n^3)$ [8]. The basic limitation is the number of degrees of freedom of the covariance matrix in the n-dimensional space. The $C^{(t)}$ matrix has $\frac{n^2+n}{2}$ parameters that must be updated to determine the $C^{(t+1)}$ matrix. So at least $\frac{n^2+n}{2}$ calculations in each generation are needed, just to determine the new covariance matrix. Generating individuals with a multivariate normal distribution with a given covariance matrix has a computational complexity of $\mathcal{O}(n^2)$. The implementation of a multivariate normal distribution generator $\mathcal{N}(m, C)$ with a given full covariance matrix C and expected value m is a computationally non-trivial task. However, the formula can be transformed as follows:

$$\begin{aligned} \mathcal{N}(m, C) &\sim m + \mathcal{N}(0, C) \\ &\sim m + C^{\frac{1}{2}}\mathcal{N}(0, I) \end{aligned} \tag{1}$$

Spectral decomposition of a covariance matrix C allows $C^{\frac{1}{2}}$ to be factorized by using a series of transformations and substitutions [7]. The following formula can be obtained:

$$C^{\frac{1}{2}} = \mathbf{BDB}^T \tag{2}$$

where:

B is an orthogonal matrix ($B^{-1} = B^T$ and $BB^T = I$), in which columns form an orthonormal basis of eigenvectors of matrix C

D is a diagonal matrix with square roots of eigenvalues of C as diagonal elements. $\mathbf{D}^2 = \mathrm{DD} = diag(d_1, \ldots, d_n)^2 = diag(d_1^2, \ldots, d_n^2)$, where d_n^2 is n-th eigenvalue of matrix C

From Eqs. (1) and (2), the following equation can be obtained [7]:

$$\begin{aligned} \mathcal{N}(m, C) &\sim m + C^{\frac{1}{2}}\mathcal{N}(0, I) \\ &\sim m + BD\underbrace{B^T\mathcal{N}(0, I)}_{\mathcal{N}(0,I)} \\ &\sim m + BD\mathcal{N}(0, I) \end{aligned} \tag{3}$$

The formula (3) allows the generation of points with multivariate normal distribution to be simplified by using spherical (isotropic) normal distribution $\mathcal{N}(0, I)$. According to that formula, each generation of a single point requires the multiplication of the orthogonal matrix $\mathbf{B}$, the diagonal matrix $\mathbf{D}$ and the n-dimensional result vector $\mathcal{N}(0, \mathbf{I})$. The process of matrix multiplication with a Coppersmith and Winograd algorithm [5] has a computational complexity of at least $\mathcal{O}(n^{2.376})$, but the complexity of matrix C factorization into a BDB^T is $\mathcal{O}(n^3)$.

Typically, evolution strategies are benchmarked on problems on a scale of up to hundreds of dimensions [2]. This is in stark contrast to the number of parameters of state-of-the-art neural networks. Here, the high computational complexity of evolution strategies makes their use virtually impossible. However, there is a possibility to generate new points using multivariate Gaussian distribution without an explicitly defined covariance matrix. (`DES`) [1] is an algorithm that is a crossover between DE and CMA-ES. It uses combinations of difference vectors between archived individuals and univariate Gaussian random vectors along directions of past shifts of population midpoints (Algorithm 1 Line 14). According to the experimental results, `DES` reveals a linear convergence rate for quadratic functions in a broad spectrum of Hessian matrix condition numbers. The authors of this method also experimentally verified that `DES` tends to perform contour fitting. The authors of `DES` have proved in their article that it is possible to achieve $w_i^{(t+1)} \sim \mathcal{N}(m, C)$ without an explicitly calculated or estimated covariance matrix.

Overcoming the computational difficulties connected with applying `DES` to larger dimensionalities is thus an auspicious direction for research. Potentially, this could lead to a method that has the desirable properties of very effective second-order approaches.

2.2 nDES Method Formulation

On a high level, the (`nDES`) method is a variant of `DES` specifically tailored for solving high dimensional optimization tasks typical for neural network training. Because of `DES` primary usability as evolutionary algorithm, it may manifest some numerical and memory complexity problems for high dimensional optimization. We introduce several crucial modifications to the original metaheuristic to make it applicable in such circumstances. To further exploit the fact that

1 $t \leftarrow 1$
2 initialize$\Big(W^{(1)} = \Big\{\mathbf{w}^{(1)}_{1,\ldots,\lambda} : \mathbf{w}^{(1)}_i \sim \mathcal{N}(0, \frac{6}{f_{in}+f_{out}})\Big\}\Big)$
3 **while** *!stop* **do**
4 evaluate$\Big(W^{(t)}; \hat{Q}\Big)$
5 $\mathbf{m}^{(t+1)} = \frac{1}{\mu}\sum_{i=1}^{\mu} \mathbf{w}^{(1)}_i$
6 $\mathbf{\Delta}^{(t)} \leftarrow \mathbf{m}^{(t+1)} - \mathbf{m}^{(t)}$
7 **if** $t = 1$ **then**
8 $\mathbf{p}^{(t)} \leftarrow \mathbf{\Delta}^{(t)}$
9 **else**
10 $\mathbf{p}^{(t)} \leftarrow (1 - c_c)\mathbf{p}^{(t-1)} + \sqrt{\mu c_c(2 - c_c)}\mathbf{\Delta}^{(t)}$
11 **for** $i = 1, \ldots, \lambda$ **do**
12 pick at random $\tau_1, \tau_2, \tau_3 \in \{1, ..., H\}$
13 $j, k \sim U(1, ..., \mu)$
14 $\mathbf{d}^{(t)}_i \leftarrow \sqrt{\frac{c_d}{2}}\Big(\mathbf{w}^{(t-\tau_1)}_j - \mathbf{w}^{(t-\tau_1)}_k\Big)$
15 $+ \sqrt{c_d}\mathbf{\Delta}^{(t-\tau_2)} \cdot N(0, 1) \; + \; \sqrt{1 - c_d}\mathbf{p}^{(t-\tau_3)} \cdot N(0, 1)$
16 $\mathbf{w}^{(t+1)}_i \leftarrow \mathbf{m}^{(t+1)} + \mathbf{d}^{(t)}_i$
17 **if** *gradientMutation* **then**
18 $\mathbf{w}^{(t+1)}_i \leftarrow \mathbf{w}^{(t+1)}_i + \epsilon\nabla\hat{Q}(w^{(t)}_i)$
19 **else if** *randomNoise* **then**
20 $\mathbf{w}^{(t+1)}_i \leftarrow \mathbf{w}^{(t+1)}_i + (1 - c_{cov})^{t/2} \cdot N(\mathbf{0}, \mathbf{I})$
21 $t \leftarrow t + 1$
22 **return** $\mathbf{w}_{best}$

Algorithm 1: Outline of the neural Differential Evolution Strategy

in such optimization scenarios, the gradient is usually available, we also propose ways to utilize this information in the algorithm.

Algorithm 1 presents the `nDES` outline and reveals a structure somewhat typical to evolution strategies: an initial population consisting of individuals is created (Line 2). This is later iteratively processed by the outer loop (Lines 3 to 21) and finally, after the stop criterion gets triggered, the best-found individual is returned (Line 22). When applied to neural networks, each individual in the population corresponds to the trained network's weights. We discuss the details of this process below.

Each individual in the first population is initialized (Line 2) according to a multivariate normal distribution, with zero mean and standard deviation equal to Xavier's coefficient for each layer of the underlying network ([6]). In other words, standard deviation differs for each layer — it is proportional to the inverse of the total number of input (f_{in}) and output (f_{out}) connections to that layer. This approach takes into account the underlying structure of the optimized models and speeds up convergence.

A pass through the main loop of the algorithm (Lines 3 to 21) corresponds to a single iteration of the method. Each iteration begins with evaluating the

individual networks according to the loss function $\hat{Q}$ (Line 4). If the training dataset is too big to calculate this function in a single forward pass, we introduce the *batching technique*, which involves multiple tweaks to the method. Before the optimization, the training dataset is split into batches, which stay constant throughout the training process. Next, an Exponentially Weighted Moving Average (EWMA) value is set to zero for each batch. Each individual in the population gets assigned to a data batch — the assignment is cyclical, in the order the batches were created. The individual's fitness is the difference between the loss function evaluated on the given batch and its EWMA value. After all points have been evaluated, the EWMA for each batch is updated with the new fitness values. The update follows the formula below:

$$\bar{Q}_t = \frac{\alpha_t}{n_t} \sum_{i}^{n_t} \hat{Q}_i + (1 - \alpha_t)\, \bar{Q}_{t-1} \tag{4}$$

$$\alpha_t = \frac{1}{\sqrt[3]{t}} \tag{5}$$

where $\bar{Q}_t$ is the EWMA value for the batch in the t-th iteration, $\hat{Q}_i$ is the fitness value for the i-th point that evaluated the batch, n_t is the total number the batch has been evaluated in the t-th iteration, and α_t is the multiplier value in the t-th iteration. Using EWMA-based fitness helps `nDES` to select the best points (i.e., ones that improve the current state) as opposed to favouring points evaluated on batches that were easy to classify. Note that this technique influences the procedure used to return the best individual found (Line 22), which is described below.

After evaluating the population, we use μ best individuals to calculate the center of the population and its shift from the previous iteration, which undergoes exponential smoothing (Lines 5 to 10). In the next steps, the new population gets initialized (Lines 11 to 20). The construction of each individual involves parameters drawn from three historical iterations (Line 12). These allow us to construct the difference vector (Line 14) and mutate the population center to form the new individual (Lines 17 to 20).

The `nDES` method uses one or two mutation operators. The first one is standard differential mutation, conducted by adding the difference vector to the population center (Line 16). The second one is optional and comes in two types: random noise perturbation (Line 20) or gradient-based rotation (Line 18), which we discuss below.

The rotation-based approach is dedicated to the tasks in which the gradient can easily be computed and does not cause numeric issues. Technically, these gradients are obtained as follows: after evaluating an individual (i.e., calculating the cost function), we calculate the backpropagation pass. The mutation then incorporates a classic SGD step, which is especially useful when the population size is smaller than the optimized network's number of parameters. Under such circumstances, the population spans a sub-hyperplane in the space of possible solutions (since $\lambda < N$). In such a case, one can think about the above tech-

nique as using the gradient information to rotate the sub-hyperplane, while nDES optimizes solutions within it.

To select the best solution ($\mathbf{w}_{best}$) for the optimization process (Line 22), different approaches are used depending on batching. Without batching, each point is evaluated on the entire dataset, so we select the point with the lowest fitness in the history as the best one. However, in the other case, the EWMA modification makes this approach unfeasible for nDES. The reason is that the modified fitness value would be the lowest for the point that showed the most improvement on a particular batch of data. Such a point would most probably not be the best solution, as the fitness values' improvements tend to drop near zero in the late stages of the training (i.e., there is relatively little progress as the optimization process starts to converge). Therefore, we evaluate all points in the final population on the validation set and return the best-performing one.

To better specialize nDES for various tasks, we introduce three different variants of the method:

- nDES: optimizing from scratch, with differential and random noise mutations,
- nDES-G: optimizing from scratch using differential and rotational mutations,
- nDES-B: bootstrapping the optimization with a gradient method and then optimizing this pre-trained network with nDES using only differential mutation.

In the nDES-B case, we first train the model using a gradient method (in our experiments, ADAM) with early stopping to prevent overfitting. Next, we use the weights of the model to initialize the population of our evolution strategy. Specifically, the mean of the sampling distribution for the initial population is set using the weights of the pre-trained model. Afterwards, normal nDES optimization is used. The motivation here is to check whether it is possible to make any improvements to a model for which a gradient optimizer has apparently converged. Note that using gradients within nDES-B would potentially harm the results in this case. The reason is that the gradient-based method uses early stopping – thus, the gradients calculated for the point at which the method ended could overfit the model rather than improve it.

3 Experiments

We conducted a series of experiments to evaluate the proposed method's performance on training neural networks. Below, we report the results obtained while training a CNN on one of the popular benchmark datasets to showcase how nDES handles highly dimensional optimization tasks. We also include a section dedicated to multiple RNN toy problems, which are very difficult for gradient-based methods.

All the experiments were conducted using an implementation created in PyTorch v. 1.5 framework [14] and a machine with an Intel Core i7-6850K processor, 128GB RAM, and a single GTX 1080 Ti GPU card.

3.1 Training CNNs

The CNN experiments were conducted using the model summarized in Table 1 trained on the `FashionMNIST` [20] dataset. The neural network consisted of two convolutional layers with soft-sign activation and max pooling, and two linear layers – the first one with soft-sign and the second with soft-max activation functions. This architecture has 50,034 trainable parameters in total, which defines quite a challenging task in terms of metaheuristic optimization.

To thoroughly analyze the results, we will analyze multiple aspects connected with the training process and the models it produced. Firstly, we will report on the generated models' accuracy to get a high-level overview of their relative performance. Furthermore, to check whether there are any structural differences in their behaviour, we will compare their confusion matrices and analyze their robustness to adversarial noise. Finally, to focus on the stability of the training process, we will present learning curves from different training runs.

Table 1. Architecture of the neural network used to classify the `FashionMNIST` dataset

Layer	Layer's output dim.	Param. #
`Conv2d-1`	$20 \times 24 \times 24$	520
`Conv2d-2`	$32 \times 8 \times 8$	16,032
`Linear-3`	64	32,832
`Linear-4`	10	650

Accuracies obtained for the `FashionMNIST` dataset are depicted in Table 2. `nDES` was run with the following hyperparameters: $c_{cum} = 0.96$; history $= 16$; batch size $= 64$; validation size $= 10000$. The dataset was normalized with respect to the mean and standard deviation prior to the optimization process, and no further augmentation techniques were used. The boundaries for weights were $[-2, 2]$.

To evaluate the training stability, in Fig. 1 (left) we also provide loss and accuracy learning curves obtained during multiple runs of `nDES`. From the plots, we may conclude that the training process seems stable and repeatable. In the case of `nDES-B`, the learning curves are presented in Fig. 1(right). At first sight, these might appear less stable; however, this is because the algorithm starts from a point already optimized using `ADAM`, and thus the plot's scale is different.

Table 2. Experimental results obtained on the `FashionMNIST` dataset. We report accuracy values along with their standard deviations for each of the methods considered

Method	Test acc. mean	Test acc. σ	Test loss mean	Test loss σ
`ADAM`	89.89	0.17	0.2854	0.0031
`nDES`	86.44	0.24	0.3836	0.0048
`nDES-G`	89.15	0.37	0.3253	0.0064
`nDES-B`	90.67	0.17	0.2761	0.0032

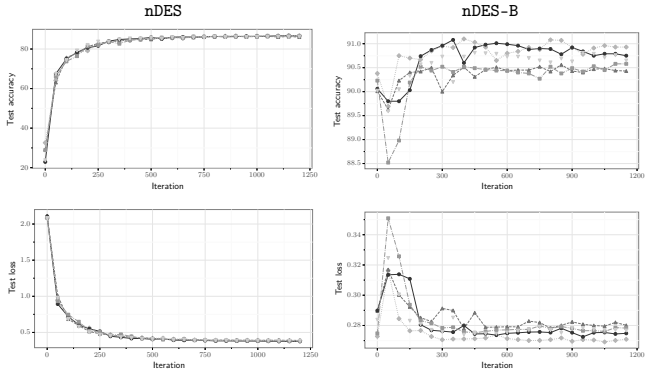


Fig. 1. Results of repeated training runs on the `FashionMNIST` dataset presented on individual curves. The left column depicts `nDES` results and the right one (`nDES-B`) the case in which the model is first trained with `ADAM` and then with `nDES`.

One measure in which `ADAM` and `nDES` differ significantly is the execution time. On average, training the model using `ADAM` in this experiment took about one minute, while using `nDES` took about 55 hours. We do not present a precise performance comparison, as it would require more in-depth analysis. This difference stems from the fact that in order for `nDES` to work properly, it needs to make significantly more passes through the training dataset than `ADAM`. This problem is, however, embarrassingly parallel [19], and we discuss it in Section 4.

3.2 Training RNNs

RNN models are useful in tasks connected with sequence modelling. One crucial problem that escalates when these sequences become longer is vanishing/exploding gradients, which impair the performance of gradient-based methods. As previous research indicates [3], metaheuristics seem to be more robust against this problem. To evaluate whether this also holds in `nDES`, we conducted experiments on three synthetic datasets taken from the classic [3, 12] papers — these are:

- the parity problem: given a sequence of 1's and -1, the model has to predict a class — 1 if the sequence contains an odd number of 1's and -1 otherwise,
- the addition problem: given a sequence of real numbers (within [-1,1]) and a binary mask, the task is to predict the sum of numbers for which the mask values are equal to 1,
- the multiplication problem: similar to the addition problem, the model has to predict the product of its masked input sequence.

During all three experiments, we used a neural network consisting of a single recurrent layer with four neurons, each having a hyperbolic tangent activation

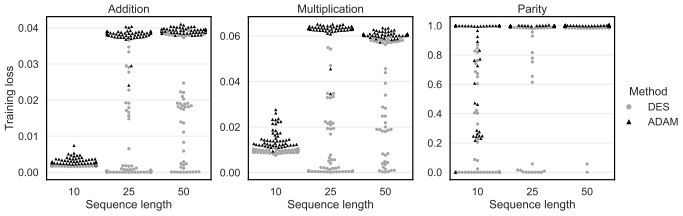


Fig. 2. Results obtained for all sequence modelling problems

function and an output layer with a single, linearly activated neuron. All neurons in the recurrent layer have the same input, whose size depends on the task at hand. This architecture was chosen arbitrarily, as to our surprise, all of the classic papers describing these toy problems lack the precise descriptions of the model.

To compare `nDES` with a gradient method, we used the `ADAM` optimizer on all problems. We used mean squared error (MSE) loss for all problems and the results are presented in Fig. 2 and Table 3. Although these tasks are challenging for gradient-based methods, `nDES` successfully solved them. When analyzing the plots, it can be seen that the difficulty of the toy problems increased along with sequence size. Furthermore, not all optimization runs found the actual minimas (for which the loss equals zero). In all cases, `nDES` was more effective — `ADAM` could only barely cope with the shortest sequences. One interesting case is with the addition and multiplication problems, for which the longer sequences turned out to be easier for `nDES` than the shortest ones. This phenomenon was especially visible in the case of the latter.

Table 3. Results obtained for all sequence modelling problems — we report the obtained means and standard deviations. In each case, `nDES` obtained better results

		Loss mean		Loss std dev.	
Problem	Seq. length	`ADAM`	`nDES`	`ADAM`	`nDES`
addition	10	0.0033	0.0018	0.0010	0.0002
	25	0.0379	0.0070	0.0024	0.0112
	50	0.0393	0.0179	0.0008	0.0148
multiplication	10	0.0152	0.0093	0.0044	0.0007
	25	0.0625	0.0111	0.0048	0.0152
	50	0.0607	0.0320	0.0013	0.0237
parity	10	0.7583	0.1894	0.3469	0.3019
	25	0.9994	0.7485	0.0021	0.4003
	50	1.0004	0.9494	0.0022	0.1881

4 Closing Remarks

The changes to DES proposed in this paper lead to a method that can be useful in neural network training. Training CNNs with nDES leads to models with comparable accuracy to a modern gradient-based method, which experimentally proves that nDES can be a useful algorithm within the field of deep neuroevolution. From an optimization task perspective, using nDES during the conducted experiments, we were able to solve optimization tasks that had orders of magnitude more dimensions than the ones in previous literature concerning DES reports. Thus, our method resolves the computational complexity problems in evolution strategies, allowing them to be used on dimensionalities not manageable before.

Apart from experiments focused on pushing the dimensionality boundaries, the promising results obtained for RNNs also suggest an exciting research direction. Note that whereas CNN architectures seem to be a perfect match for gradient-based optimization methods, for RNNs, specialized architectures (like LSTM [12] or the more recent gated recurrent unit (GRU) [4]) have been developed to overcome the limitations of these methods and make training possible. The excellent performance obtained by nDES on challenging RNN tasks suggests that it is worth evaluating on a larger scale how well modern metaheuristic methods cope in such a setting. The possibility to effectively train RNN models without the need to use structural workarounds for gradient optimizers' deficiencies could significantly impact domains like natural language processing or recommender systems. Applying nDES to more real-world RNN architectures is thus also a direction worth pursuing.

The source code of nDES implementation and all experiments is available at https://github.com/fuine/nDES.

Further development of nDES will focus on solving the encountered problems. One particular source of problems, which limited the scope of experiments in this paper, is that nDES relies on the history of the population to perform well. Moreover, the population size should be multiple times larger than the problem's dimensionality, as suggested by the authors of DES ($\lambda = 4n$). Both these factors result in significant memory requirements and suggest that scaling nDES with the size of the architectures might pose a real difficulty, particularly in the case of deep convolutional models known for their sheer number of parameters. However, the parts of nDES that are the most computationally expensive (i.e., evaluation of the population) are embarrassingly parallel. Therefore, further increases in task dimensionality, and thus experiments on more complicated neural networks, seem to be only a technical hurdle — one resolvable by using multiple GPU cards and parallelizing the evaluation loop.

References

1. Arabas, J., Jagodziński, D.: Toward a matrix-free covariance matrix adaptation evolution strategy. IEEE Trans. on Evolutionary Computation **24**(1), 84–98 (2020)
2. Awad, N.H., Ali, M.Z., Liang, J.J., Qu, B.Y., Suganthan, P.N.: Problem definitions and evaluation criteria for the cec 2017 special session and competition on single

objective bound constrained real-parameter numerical optimization. Tech. rep., Nanyang Technological University, Singapore (11 2016)
3. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2), 157–166 (1994)
4. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
5. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation **9**(3), 251 – 280 (1990). https://doi.org/https://doi.org/10.1016/S0747-7171(08)80013-2, computational algebraic complexity editorial
6. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proc. of the 13 Int. Conf. on Artificial Intelligence and Statistics. vol. 9, pp. 249–256. PMLR, Chia Laguna Resort, Sardinia, Italy (13–15 May 2010)
7. Hansen, N.: The CMA Evolution Strategy: A Tutorial (2005), `https://hal.inria.fr/hal-01297037`, arXiv e-prints, arXiv:1604.00772, 2016, pp.1-39
8. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**, 159–195 (06 2001). https://doi.org/10.1162/106365601750190398
9. Hansen, N.: The CMA evolution strategy: a comparing review. In: Towards a new evolutionary computation: advances on estimation of distribution algorithms, pp. 75–102. Springer (2006)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
11. Hochreiter, S., Kolen, J.F., Kremer, S.C.: Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp. 237–243. Wiley-IEEE Press (2001)
12. Hochreiter, S., Schmidhuber, J.: Long Short-term Memory. Neural computation **9**, 1735–80 (Dec 1997). https://doi.org/10.1162/neco.1997.9.8.1735
13. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines vinod nair. In: Proceedings of ICML. vol. 27, pp. 807–814 (06 2010)
14. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Adv. in Neural Inf. Proc. Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
15. Rojas, R.: Neural Networks: A Systematic Introduction. Springer-Verlag, Berlin, Heidelberg (1996)
16. Shir, O., Yehudayoff , A.: On the covariance-hessian relation in evolution strategies. Theoretical Computer Science. Elsevier **801**, 157–174 (2020)
17. Stanley, K., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1**, 24–35 (01 2019). https://doi.org/10.1038/s42256-018-0006-z
18. Storn, R., Price, K.: Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. J. of Global Opt. **23** (01 1995)
19. Wikipedia contributors: Embarrassingly parallel — Wikipedia, the free encyclopedia (2020), `https://en.wikipedia.org/wiki/Embarrassingly_parallel`, [Online; accessed 10-September-2020]
20. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)

B.3. Least Redundant Gated Recurrent Neural Network

Title	Least Redundant Gated Recurrent Neural Network
Authors	Łukasz Neumann, Łukasz Lepak, Paweł Wawrzyński
Conference	International Joint Conference on Neural Networks (IJCNN 2023)
Year	2023
Ministerial score	140

Least Redundant Gated Recurrent Neural Network

1st Łukasz Neumann
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland
lukasz.neumann@pw.edu.pl

2nd Łukasz Lepak
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland
lukasz.lepak.dokt@pw.edu.pl

3rd 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). [1]. 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

[1]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 $t = 1, 2, \ldots$. At each time, the module is fed with the input $x_t \in \mathbb{R}^m$ and produces the vector $h_t \in \mathbb{R}^d$, which is both its memory state and its output.

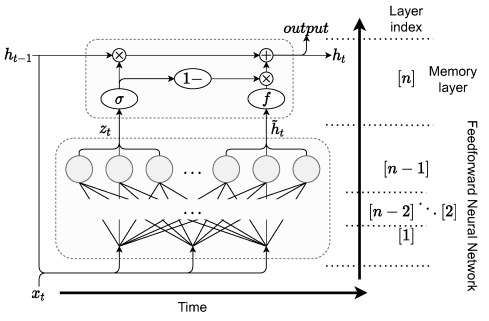


Fig. 1. Structure of Deep Memory Update module. The module comprises the feedforward neural network, which can arbitrarily process the state and a memory layer. The output of the module is also its hidden state.

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 $\widehat{h}_t \in \mathbb{R}^d$ determines the direction in which the state should change.

A pair of i-th elements of z_t and $\widehat{h}_t$ 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 $\widehat{h}_t$. The memory state update takes the form

$$\langle z_t, \widehat{h}_t \rangle = \text{FNN}(h_{t-1}, x_t) \tag{1}$$

$$h_t = h_{t-1} \circ \sigma(z_t) + f(\widehat{h}_t) \circ (\mathbf{1} - \sigma(z_t)), \tag{2}$$

where "$\circ$" denotes the elementwise product, $\mathbf{1}$ is a vector of ones, σ is a unipolar soft step function, e.g. the logistic sigmoid,

$$\sigma_i(z) = \frac{e^{z_i}}{1 + e^{z_i}} \text{ for } z_i \in \mathbb{R}, \tag{3}$$

and f is an activation function, e.g.

$$f_i(z) = \tanh(z_i) \text{ for } z_i \in \mathbb{R}. \tag{4}$$

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, β_{DMU}, equal to

$$\beta_{\text{DMU}} = \frac{\beta}{2n}. \tag{5}$$

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.
- σ takes the form (3). Therefore, $\sigma'(z) = \sigma(z)(1-\sigma(z)) < 1 - \sigma(z)$.
- Vectors considered are in row form.

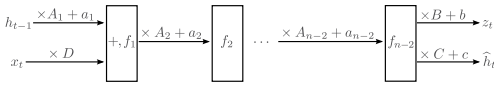


Fig. 2. Structure of the feedforward module inside DMU. A_i, B, C and D denote weight matrices, a_i, b and c denote vectors of biases and f_i denotes activation functions.

Note that h_t are in fact weighted averages, over $i \geq 0$, of $f(\widehat{h}_{t-i})$ 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:

$$\begin{aligned}
&\frac{\mathrm{d}L_{t'}}{\mathrm{d}h_{t-1}} = \frac{\mathrm{d}L_{t'}}{\mathrm{d}h_t}\frac{\mathrm{d}h_t}{\mathrm{d}h_{t-1}} \\
&= \frac{\mathrm{d}L_{t'}}{\mathrm{d}h_t}\frac{\mathrm{d}}{\mathrm{d}h_{t-1}}\left(h_{t-1} \circ \sigma(z_t) + f(\widehat{h}_t) \circ (\mathbf{1} - \sigma(z_t))\right) \\
&= \frac{\mathrm{d}L_{t'}}{\mathrm{d}h_t}\Bigg(\operatorname{diag}(\sigma(z_t)) + [h_{t-1}] \circ \frac{\mathrm{d}z_t}{\mathrm{d}h_{t-1}} \circ [\sigma'(z_t)] + \frac{\mathrm{d}\widehat{h}_t}{\mathrm{d}h_{t-1}} \circ \\
&\qquad \circ \left[f'(\widehat{h}_t)\right] \circ [\mathbf{1} - \sigma(z_t)] - \left[f(\widehat{h}_t)\right] \circ \frac{\mathrm{d}z_t}{\mathrm{d}h_{t-1}} \circ [\sigma'(z_t)]\Bigg) \\
&= \frac{\mathrm{d}L_{t'}}{\mathrm{d}h_t}\Bigg(\operatorname{diag}(\sigma(z_t)) + \left[h_{t-1} - f(\widehat{h}_t)\right] \circ \frac{\mathrm{d}z_t}{\mathrm{d}h_{t-1}} \circ [\sigma'(z_t)] \\
&\qquad + \frac{\mathrm{d}\widehat{h}_t}{\mathrm{d}h_{t-1}} \circ \left[f'(\widehat{h}_t)\right] \circ [\mathbf{1} - \sigma(z_t)]\Bigg),
\end{aligned} \tag{6}$$

where $\operatorname{diag}(v)$ 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):

$$\frac{\mathrm{d}z_t}{\mathrm{d}h_{t-1}} \cong B^T \left(\prod_{i=1}^{n-2} A_i\right)^T, \quad \frac{\mathrm{d}\widehat{h}_t}{\mathrm{d}h_{t-1}} \cong C^T \left(\prod_{i=1}^{n-2} A_i\right)^T. \tag{7}$$

Considering that $\sigma(z_t) \in (0, 1)$, $h_{t-1} - f(\widehat{h}_t) \in (-2, 2)$, $\sigma'(z_t) \in (0, 1-\sigma(z_t))$, $f'(\widehat{h}_t) \in (0, 1)$, we obtain the following condition on non-increasing gradient:

- $\star$ 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 ($\star$) 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 RNH. 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.

		train			test		
N	model	best	μ	σ	best	μ	σ
1	RHN	3.34	3.38	0.08	**3.552**	**3.598**	**0.037**
	DMU	**2.94**	**2.96**	**0.04**	3.57	3.63	0.05
2	RHN	3.39	3.41	0.10	3.55	3.61	0.06
	DMU	**3.02**	**3.09**	**0.07**	**3.49**	**3.55**	**0.04**
5	RHN	3.44	3.68	0.16	3.73	3.85	0.11
	DMU	**3.22**	**3.21**	**0.06**	**3.63**	**3.69**	**0.03**
10	RHN	3.70	3.93	**0.14**	**3.90**	4.08	0.12
	DMU	**3.52**	**3.54**	0.15	3.95	**4.04**	**0.09**

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.

		train			test		
N	model	best	μ	σ	best	μ	σ
1	RHN	61.10	66.53	6.07	106.11	110.39	4.36
	DMU	**56.63**	**59.42**	**3.14**	**105.35**	**106.13**	**0.48**
2	RHN	**62.90**	66.32	4.23	**104.89**	**109.83**	4.04
	DMU	64.42	**64.86**	**1.43**	109.60	110.13	**0.47**
5	RHN	**82.33**	**86.10**	3.83	123.16	124.97	1.92
	DMU	92.40	94.06	**1.11**	**117.92**	**120.37**	**1.46**
10	RHN	**85.97**	149.43	86.87	**124.46**	171.60	60.38
	DMU	118.20	**119.48**	**1.53**	130.05	**131.83**	**1.44**

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.

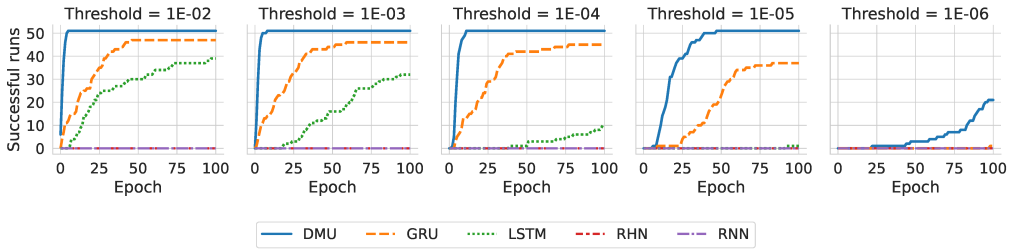


Fig. 3. Adding: Results of 51 runs, five graphs for different loss thresholds, a curve presents how many runs reach a given loss threshold at a given training epoch.

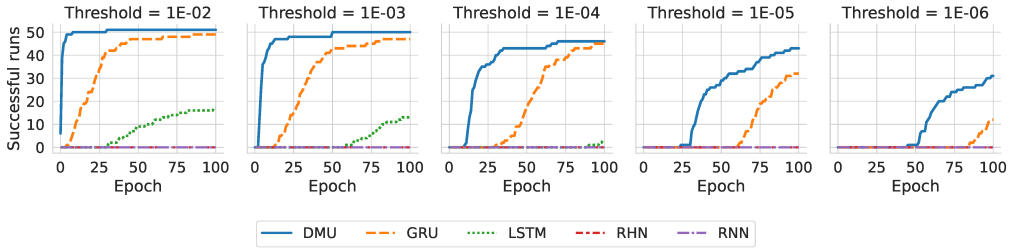


Fig. 4. TempOrd: Results of 51 runs, five graphs for different loss thresholds.

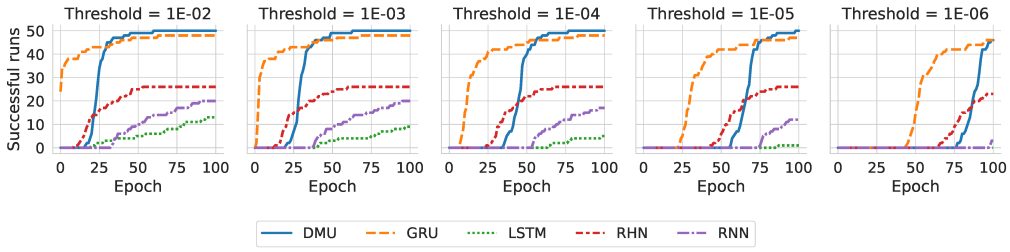


Fig. 5. NoiseSeq: Results of 51 runs, five graphs for different loss thresholds.

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 best	train μ	train σ	test best	test μ	test σ
Spa2Eng	1	RHN	8.26	7.72	0.29	**5.80**	6.31	0.36
		DMU	**7.13**	**7.24**	**0.17**	5.89	**6.05**	**0.14**
	2	RHN	**8.82**	**8.50**	**0.22**	**6.53**	**7.10**	0.36
		DMU	9.07	8.52	0.38	6.91	7.18	**0.32**
	5	RHN	12.38	24.68	12.48	12.34	47.60	41.68
		DMU	**7.74**	**7.91**	**0.35**	**7.50**	**8.02**	**0.35**
	10	RHN	58.74	58.73	0.92	110.54	141.77	56.01
		DMU	**8.78**	**8.84**	**0.26**	**7.99**	**8.40**	**0.23**
Por2Eng	1	RHN	3.95	3.98	**0.09**	3.65	3.80	0.16
		DMU	**3.91**	**3.93**	0.11	**3.54**	**3.68**	**0.10**
	2	RHN	**4.29**	**4.33**	**0.06**	**3.67**	**3.93**	0.18
		DMU	4.63	4.45	0.14	3.74	3.97	**0.14**
	5	RHN	6.25	7.50	0.83	6.55	7.62	0.85
		DMU	**4.65**	**4.74**	**0.09**	**4.56**	**4.69**	**0.10**
	10	RHN	48.58	48.35	0.37	79.10	99.86	23.80
		DMU	**5.12**	**5.29**	**0.18**	**4.92**	**5.06**	**0.12**
Ger2Eng	1	RHN	4.66	4.63	0.16	4.27	4.35	**0.06**
		DMU	**4.28**	**4.50**	**0.12**	**4.10**	**4.21**	0.08
	2	RHN	**5.26**	**5.10**	0.11	**4.41**	**4.59**	**0.10**
		DMU	5.38	5.28	**0.07**	4.56	4.79	0.19
	5	RHN	8.13	9.24	1.24	7.93	9.18	1.28
		DMU	**5.33**	**5.42**	**0.12**	**5.21**	**5.33**	**0.12**
	10	RHN	47.99	48.41	0.29	83.47	134.79	92.37
		DMU	**5.91**	**5.82**	**0.23**	**5.63**	**5.74**	**0.07**

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 state of the memory cells. Architectures based on the DMU

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
Antisymmteric 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]	**99.4%**	**96.3%**	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 best	train μ	train σ	test best	test μ	test σ
1	DMU-C	**2.72**	**2.81**	0.08	**3.34**	**3.38**	**0.04**
	DMU	2.94	2.96	**0.04**	3.57	3.63	0.05
2	DMU-C	**2.99**	**3.03**	0.20	**3.43**	**3.49**	0.04
	DMU	3.02	3.09	**0.07**	3.49	3.55	**0.04**
5	DMU-C	3.25	3.31	0.24	3.90	3.99	0.11
	DMU	**3.22**	**3.21**	**0.06**	**3.63**	**3.69**	**0.03**
10	DMU-C	3.75	nan	nan	4.26	5.03	0.63
	DMU	**3.52**	**3.54**	**0.15**	**3.95**	**4.04**	**0.09**

TABLE VI
NATLANG: VARIED LEARNING RATE ABLATION RESULTS — PERPLEXITY. DMU-C — DMU WITH AN EQUAL LEARNING RATE FOR ALL MODULES.

N	model	train best	train μ	train σ	test best	test μ	test σ
1	DMU-C	68.99	71.12	**1.78**	109.23	111.20	1.42
	DMU	**56.63**	**59.42**	3.14	**105.35**	**106.13**	**0.48**
2	DMU-C	81.31	81.01	1.69	117.56	118.17	0.65
	DMU	**64.42**	**64.86**	**1.43**	**109.60**	**110.13**	**0.47**
5	DMU-C	108.90	579.68	235.39	138.13	542.20	202.04
	DMU	**92.40**	**94.06**	**1.11**	**117.92**	**120.37**	**1.46**
10	DMU-C	696.30	697.57	**0.69**	642.18	642.91	**0.44**
	DMU	**118.20**	**119.48**	1.53	**130.05**	**131.83**	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 best	train μ	train σ	test best	test μ	test σ
Spa2Eng	1	DMU-C	0.91	0.85	0.14	**0.70**	0.67	0.04
		DMU	**0.93**	**0.93**	**0.00**	0.70	**0.69**	**0.01**
	2	DMU-C	0.84	0.76	0.20	0.66	0.60	0.11
		DMU	**0.94**	**0.94**	**0.01**	**0.69**	**0.69**	**0.01**
	5	DMU-C	0.71	0.66	0.06	0.62	0.59	0.03
		DMU	**0.84**	**0.83**	**0.01**	**0.66**	**0.65**	**0.01**
	10	DMU-C	0.34	0.30	0.02	0.35	0.31	0.02
		DMU	**0.76**	**0.75**	**0.01**	**0.63**	**0.63**	**0.00**
Por2Eng	1	DMU-C	**0.94**	0.93	0.02	**0.78**	0.77	0.01
		DMU	0.93	**0.94**	**0.01**	0.78	**0.77**	**0.00**
	2	DMU-C	0.94	0.93	0.02	0.75	0.75	0.01
		DMU	**0.96**	**0.95**	**0.01**	**0.78**	**0.77**	**0.00**
	5	DMU-C	0.77	0.63	0.20	0.70	0.59	0.16
		DMU	**0.86**	**0.86**	**0.00**	**0.73**	**0.73**	**0.01**
	10	DMU-C	0.32	0.32	**0.00**	0.32	0.32	**0.00**
		DMU	**0.81**	**0.80**	0.01	**0.71**	**0.71**	0.00
Ger2Eng	1	DMU-C	0.92	0.92	0.02	0.75	0.75	**0.00**
		DMU	**0.92**	**0.93**	**0.01**	**0.75**	**0.75**	0.00
	2	DMU-C	0.88	0.90	0.01	0.73	0.72	0.01
		DMU	**0.94**	**0.94**	**0.01**	**0.75**	**0.74**	**0.00**
	5	DMU-C	0.72	0.70	0.02	0.66	0.65	0.01
		DMU	**0.85**	**0.84**	**0.01**	**0.71**	**0.70**	**0.01**
	10	DMU-C	0.44	0.34	0.06	0.45	0.34	0.05
		DMU	**0.80**	**0.78**	**0.01**	**0.68**	**0.68**	**0.00**

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 *Interspeech*, 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 multi-modal 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. 106 548, 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. 149 362–149 371, 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 (indrnn): 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 Merrienboer, 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
COMPARISION OF DMU AND GRU

In the notation applied in this paper operation of a GRU [7] layer can be expressed as

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

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

$$\begin{aligned} \widehat{h}_t &= W_h x_t + U_h h_{t-1} + b_h \\ z_t &= W_x x_t + U_x h_{t-1} + b_x \\ h_t &= h_{t-1} \circ \sigma(z_t) + f(\widehat{h}_t) \circ (\mathbf{1} - \sigma(z_t)) \end{aligned} \tag{A.1}$$

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 THE FOR THE COMPARISON OF DIFFERENT NEURAL MODULES IN SYNTHETIC EXPERIMENTS.[1] RECURRENT BLOCK.

experiment		RNN	LSTM	GRU	RHN	DMU
NoiseSeq	rc. blk[1]	(5, 5)	(2, 2)	(2, 3)	((3, 3))	((5, 4))
	weights no.	595	880	687	672	573
Adding	rc. blk[1]	(5, 5)	(2, 2)	(3, 2)	((4, 3))	((5, 5))
	weights no.	111	99	108	136	106
TempOrd	rc. blk[1]	(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	hyperpameter	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, 4×NVidia™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	hyperpameter	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
		max epochs	50	50
	1	learning rate	0.01	0.005
		weight decay	0.0001	0.0001
		scheduler gamma	0.9	0.9
	2	learning rate	0.01	0.01
		weight decay	0.0001	0.0001
		scheduler gamma	0.9	0.9
	5	learning rate	0.01	0.003
		weight decay	0.0001	0.0001
		scheduler gamma	0.9	1.0
	10	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 $(x, a_1, \ldots, a_{n-2})$ or $(y, a_1, \ldots, a_{n-2})$, where $x, y, a_1, \ldots, a_{n-1}$ 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.