2018 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 17–20, 2018, AALBORG, DENMARK

RECURRENT NEURAL NETWORKS WITH FLEXIBLE GATES USING KERNEL ACTIVATION FUNCTIONS

Simone Scardapane[*†], *Steven Van Vaerenbergh*[‡], *Danilo Comminiello*[†], *Simone Totaro*[†], and *Aurelio Uncini*[†]

[†] Sapienza University of Rome, Italy [‡] University of Cantabria, Spain

ABSTRACT

Gated recurrent neural networks have achieved remarkable results in the analysis of sequential data. Inside these networks, gates are used to control the flow of information, allowing to model even very long-term dependencies in the data. In this paper, we investigate whether the original gate equation (a linear projection followed by an element-wise sigmoid) can be improved. In particular, we design a more flexible architecture, with a small number of adaptable parameters, which is able to model a wider range of gating functions than the classical one. To this end, we replace the sigmoid function in the standard gate with a non-parametric formulation extending the recently proposed kernel activation function (KAF), with the addition of a residual skip-connection. A set of experiments on sequential variants of the MNIST dataset shows that the adoption of this novel gate allows to improve accuracy with a negligible cost in terms of computational power and with a large speed-up in the number of training iterations.

Index Terms— Recurrent network, LSTM, GRU, Gate, Kernel activation function

1. INTRODUCTION

Recurrent neural networks (RNNs) have recently gained a large popularity in the analysis of sequential data, following more widespread success in the field of deep learning [1]. Among all possible RNNs, gated architectures (originating from the seminal work in [2]) have shown to be particularly suitable at handling long, or very long, temporal dependencies in the data. While the original long short-term memory (LSTM) network dates back to twenty years ago, recent advances in computational power have allowed to scale them to multi-layered and sequence-to-sequence configurations [3], achieving significant breakthroughs in multiple fields, e.g., neural machine translation [4].

Fundamentally, a gate is a multiplicative layer that learns to perform a 'soft selection' of some content (e.g., the hidden state of the RNN), allowing the gradient and the information to flow more easily through multiple time-steps while avoiding vanishing or exploding gradients. Despite their importance, however, the role and the use of gates inside RNNs remain open questions for research. The original LSTM network was designed with two gates to have a unitary derivative [2], which were later increased to three with the inclusion of a forget gate [5]. Subsequent research has experimented with a wide range of different configurations, including the gated recurrent unit (GRU) [4], merging two gates into a single update gate, or even simpler architectures having a single gate, such as the minimally gated unit [6], or the JANET model [7] (see also [8] for a large comparison of feasible variations). From a theoretical perspective, [9] has recently shown that gates naturally arise if we assume (axiomatically) a (quasi-)invariance to time transformations of the input data.

Note that, even considering this wide range of alternative formulations (mostly in terms of how many gates are needed for an optimal architecture), the basic design of a *single* gate has remained more or less constant, i.e., each gate is obtained by applying an element-wise sigmoid nonlinearity to a linear projection of the inputs and/or hidden states. Only a handful of works have explored alternative designs for this component, such as the inclusion of hidden layers [10], or skip-connections through the gates of different layers [11]. Motivated by the possibility of improving the performance of the RNNs, in this paper we propose an extended gate architecture, which is endowed with a larger expressiveness than the standard formulation. At the same time, we try to keep the computational overhead as small as possible. To this end, we focus on replacing the sigmoid operation, extending it with a non-parametric form that is adapted independently for each cell (and for each gate) inside the gated RNN.

Our starting point is noting that a lot of work has been done in the deep learning literature for designing flexible activation functions, that could replace standard hyperbolic tangents or rectified linear units (ReLUs). These include simple parametric schemes like the parametric ReLU [12], or more elaborate formulations where the flexibility of the functions can be determined as a hyper-parameter. The latter case include maxout networks [13], adaptive piecewise linear units [14], and kernel activation functions (KAFs) [15]. There is a good consensus in that endowing the functions with this flex-

* Corresponding author e-mail: simone.scardapane@uniroma1.it.

978-1-5386-5477-4/18/$31.00

ibility can enhance the performance of the network, possibly allowing to simplify the architecture of the neural network itself significantly [14]. However, the sigmoid function used inside a gate is different from a standard activation function, in that its behavior cannot be unrestricted (e.g., by taking negative values). Due to this, none of these proposals can be applied straightforwardly to the case of gates inside RNNs: for example, all functions based on rectifiers (such as the APL [14]) are unbounded over their domain [14].

To this end, in this paper we propose an extension of the basic KAF model. A KAF is a non-parametric activation function defined in terms of a kernel expansion over a fixed dictionary [15]. Here, we combine it with a bounded nonlinearity and a residual connection to make its behavior consistent with that of a gating function (more details in Section 3). As a result, our proposed flexible gate mimics exactly a standard sigmoid at the beginning of the optimization process, but thanks to the addition of a small number of adaptable parameters, it can adapt itself based on the training data to a much larger family of shapes (see Fig. 1 for some examples).

We evaluate the proposed model on a set of standard benchmarks involving sequential formulations of the MNIST dataset (e.g., where each image is processed pixel-by-pixel). We show that a gated RNN with our proposed flexible gate can achieve higher accuracy with a faster rate of convergence, while at the same time having a small computational overhead with respect to a standard formulation.

Paper outline

The rest of the paper is organized as follows. In Section 2 we introduce the GRU model (as a representative example of gated RNN). Next, the proposed gate with flexible sigmoids is described in Section 3. We evaluate the proposal in Section 4, before concluding in Section 5.

2. GATED RECURRENT NEURAL NETWORKS

2.1. Update equations

Consider a generic sequential task, where at each time step t we receive a new input $\mathbf{x}_t \in \mathbb{R}^d$. The evolution of a generic RNN can be described by the following equation:

$$\mathbf{h}_t = \phi\left(\mathbf{x}_t, \mathbf{h}_{t-1}\,;\,\boldsymbol{\theta}\right) , \tag{1}$$

where $\mathbf{h}_t$ represents the internal state of the RNN, $\boldsymbol{\theta}$ is the set of adaptable parameters, and $\phi(\cdot)$ is a generic update rule. Gated RNNs implement $\phi(\cdot)$ with the presence of one or more gating functions, which control the flow of information between time steps. As we stated in Section 1, different types of gated RNNs, with different number of gates, exist in the literature. For brevity, in the rest of the paper we will focus on the case of GRUs [4], although our method extends immediately to LSTMs and any other gated network described in the previous section. However, GRUs represent a good compromise between accuracy and number of gates (two compared to three as in the LSTM), which is why we choose it here.

A GRU cell updates its internal state $\mathbf{h}_{t-1}$ as follows:

$$\mathbf{u}_t = \sigma\left(\mathbf{W}_u\mathbf{x}_t + \mathbf{V}_u\mathbf{h}_{t-1} + \mathbf{b}_u\right) , \tag{2}$$

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_r\mathbf{x}_t + \mathbf{V}_r\mathbf{h}_{t-1} + \mathbf{b}_r\right) , \tag{3}$$

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \circ \mathbf{h}_{t-1} + \mathbf{u}_t \circ \tanh\left(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_t\left(\mathbf{r}_t \circ \mathbf{h}_{t-1}\right) + \mathbf{b}_h\right) , \tag{4}$$

where (2) and (3) are, respectively, the update gate and reset gate, $\circ$ is the element-wise multiplication, $\sigma(\cdot)$ is the standard sigmoid function, and the cell has 9 adaptable matrices given by $\boldsymbol{\theta} = \{\mathbf{W}_u, \mathbf{W}_f, \mathbf{W}_h, \mathbf{V}_u, \mathbf{V}_f, \mathbf{V}_h, \mathbf{b}_u, \mathbf{b}_f, \mathbf{b}_h\}$. Note that while the $\tanh(\cdot)$ function in (4) can be changed freely (e.g., to a ReLU function), the sigmoid function in the two gates is essential for having a correct behavior, i.e., the update vector $\mathbf{u}_t$ and reset vector $\mathbf{r}_t$ should always remain bounded in $[0, 1]$.

2.2. Training the network

GRUs can be used for a variety of tasks by properly manipulating the sequence of their internal states $\mathbf{h}_1, \mathbf{h}_2, \ldots$. Since in our experiments we consider the problem of classifying each sequence of data, we briefly describe here the details of the optimization approach. We underline, however, that the method we propose in the next section is agnostic to the actual task, as it acts on the basic GRU formulation.

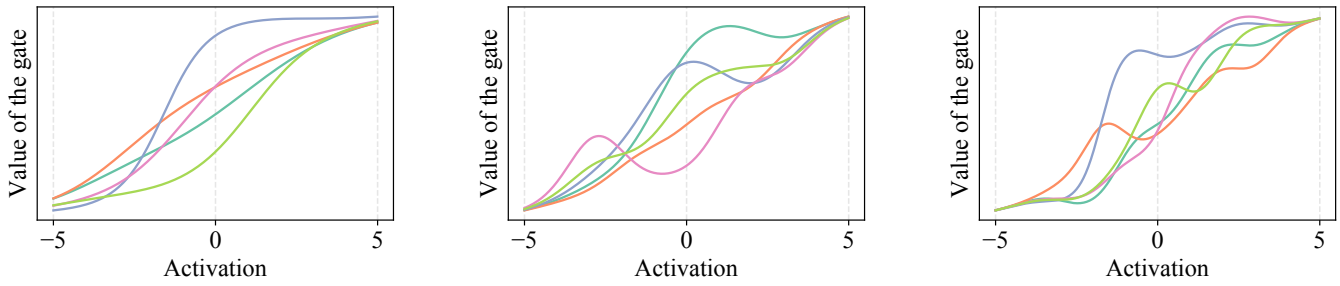
Suppose to have N different sequences $\left\{\mathbf{x}_t^i\right\}_{i=1}^N$, and for each of them a single class label $y^i = 1, \ldots, C$. Denote by $\mathbf{h}^i$ the internal state of the GRU after processing the i-th sequence. To obtain a classification, this is fed through another layer with a softmax activation function:

$$\widehat{\mathbf{y}}^i = \text{softmax}\left(\mathbf{A}\mathbf{h}^i + \mathbf{b}\right) , \tag{5}$$

with $\widehat{\mathbf{y}}^i$ having values over the C-dimensional simplex. The network is trained by minimizing the average cross-entropy between the real classes and the predicted classes:

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\left[y^i = c\right]\log\left(\widehat{y}_j^i\right) , \tag{6}$$

where $\widehat{y}_j^i$ is the j-th element of the prediction vector $\widehat{\mathbf{y}}^i$, and $[\cdot]$ is 1 if its argument is true, 0 otherwise. Minimization of (6) is obtained by unrolling the network over all time steps via back-propagation through time (BPTT) [1]. While this covers the basic mathematical framework, in practice several methods can be used to stabilize and improve the convergence of BPTT, including gradients' clipping [3], multiple variations of dropout [16], or regularizing appropriately weights and/or changes in activations during training [17].



(a) $\gamma = 1.0$ (b) $\gamma = 0.5$ (c) $\gamma = 0.1$

Fig. 1. Random samples of the proposed flexible gates with different bandwidths. In all cases we sample uniformly 10 points on the x-axis, while the mixing coefficients are sampled from a normal distribution. y-axis always goes from 0 to 1.

3. PROPOSED GATE WITH FLEXIBLE SIGMOID

What we propose in this paper is to replace the sigmoid in (2) and (3) with another (scalar) function with higher flexibility, while (a) keeping the overall 'sigmoid-like' behavior, and (b) maintaining a low computational overhead. Our proposal builds upon the KAF [15], which was originally designed as a replacement for standard activation functions, e.g., the ReLU. In this section we briefly describe the KAF formulation before introducing our extension.

3.1. Kernel activation functions

A KAF is defined as a one-dimensional kernel expansion:

$$\text{KAF}(s) = \sum_{i=1}^{D} \alpha_i \kappa\left(s, d_i\right) , \tag{7}$$

where s is a generic input to the activation function, $\kappa(\cdot,\cdot) : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a valid kernel function, $\{\alpha_i\}_{i=1}^{D}$ are called mixing coefficients, and $\{d_i\}_{i=1}^{D}$ are called the dictionary elements. To make back-propagation tractable, differently from a standard kernel method the D elements of the dictionary are fixed beforehand and shared across the entire network. In particular, we let D as a user-chosen hyper-parameter, and we sample D values over the x-axis, uniformly around zero. Basically, higher values of D will correspond to an increased flexibility of the function, together with an increase in the number of free parameters. Mixing coefficients are adapted through standard back-propagation, independently for every neuron, which can be realized efficiently through vectorized operations [15].

The kernel function $\kappa(\cdot,\cdot)$ only needs to respect the positive semi-definiteness property, and for our experiments we use the 1D Gaussian kernel defined as:

$$\kappa(s, d_i) = \exp\left\{-\gamma\left(s - d_i\right)^2\right\} , \tag{8}$$

where $\gamma \in \mathbb{R}$ is a kernel parameter, i.e., the inverse bandwidth. The parameter $\gamma > 0$ defines the range of influence of each α_i element. For selecting it, we adopt the rule-of-thumb proposed in [15]:

$$\gamma = \frac{1}{6\Delta^2} , \tag{9}$$

where Δ is the resolution of the dictionary elements. Additionally, we have found beneficial to let γ adapt independently for each KAF, always through back-propagation.

3.2. Flexible gating functions using KAFs

We cannot use (7) straightforwardly because (a) it is unbounded, and (b) using the Gaussian kernel, it goes to 0 for $s \rightarrow \pm\infty$ in both directions. We propose to alleviate these problems by using the following formulation for the flexible gate:

$$\sigma_{\text{KAF}}(s) = \sigma\left(\frac{1}{2}\text{KAF}(s) + \frac{1}{2}s\right) , \tag{10}$$

where the sigmoid σ on the right-hand side keeps the boundedness of the function, while the addition of the residual term s ensures that (10) behaves as a standard sigmoid outside the range of the dictionary. In Fig. 1 we show some realizations of (10) for different choices of the mixing coefficients and γ. It can be seen that the functions can represent a wide array of different shapes, all consistent with the general behavior of a gating function.

In fact, to simplify optimization we also initialize the mixing coefficients to approximate the identity function, so that the flexible gate behaves as a sigmoid in the initial stage of training. In order to do this, we apply kernel ridge regression on the dictionary to select the initial values for the mixing coefficients:

$$\boldsymbol{\alpha} = \left(\mathbf{K} + \varepsilon\mathbf{I}\right)^{-1}\mathbf{d} , \tag{11}$$

where $\boldsymbol{\alpha}$ is the vector of mixing coefficients, $\mathbf{d}$ is the vector of dictionary elements, $\mathbf{I}$ is the identity matrix of appropriate size, and $\varepsilon > 0$ is a small scalar coefficient to ensure stability of the matrix inversion (we use $\varepsilon = 10^{-4}$ in the experiments). By using this initialization, all gates will behave identically to

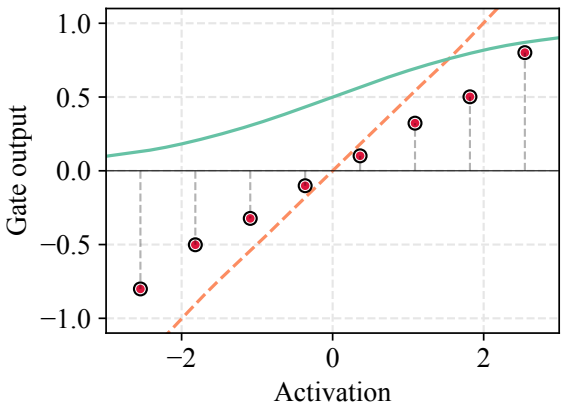


Fig. 2. Example of the proposed KAF gate when initialized as a standard sigmoid. The dashed line is KAF(s) in (10); markers show its mixing coefficients; the solid green line is the final output of the gate.

a standard sigmoid at the beginning (an example of initialization is shown in Fig. 2). In our proposed GRU, we then use a different set of mixing coefficients for each forget gate and update gate.

4. EXPERIMENTAL RESULTS

4.1. Experimental setup

To evaluate the proposed algorithm, we compare a standard GRU with a GRU endowed with flexible gates as in (10). We use a standard set of sequential benchmarks constructed from the MNIST[1] dataset, which are commonly used for testing long-term dependencies in gated RNNs [18]. MNIST is an image classification dataset composed of 60000 images for training (and 10000 for testing), each belonging to one out of ten classes. Each image is of dimension 28×28 with black-and-white pixels. From this, we construct three sequential problems:

Row-wise MNIST (R-MNIST) Each image is processed sequentially, row-by-row, i.e., we have sequences of length 28, each represented by the value of 28 pixels.

Pixel-wise MNIST (P-MNIST) Each image is represented as a sequence of 784 pixels, read from left to right and from top to bottom from the original image.

Permuted P-MNIST (PP-MNIST) Similar to P-MNIST, but the order of the pixels is shuffled using a (fixed) permutation matrix.

[1] http://yann.lecun.com/exdb/mnist/

Dataset	GRU (Standard)	GRU (proposed)
R-MNIST	98.29 ± 0.01	$\mathbf{98.67 \pm 0.02}$
P-MNIST	89.50 ± 5.64	$\mathbf{97.34 \pm 0.61}$
PP-MNIST	86.41 ± 6.71	$\mathbf{96.10 \pm 0.93}$

Table 1. Average test accuracy obtained by a standard GRU compared with a GRU endowed with the proposed flexible gates (standard deviation is shown in brackets).

P-MNIST and PP-MNIST are particularly challenging because of the need of processing relatively long-term dependencies in the data.

GRUs have an internal state of dimensionality 100, and we include an additional batch normalization step [1] before the output layer in (5) to stabilize training in the presence of long sequences. We train using the Adam optimization algorithm with BPTT on mini-batches of 32 elements, with an initial learning rate of 0.001, and we clip all gradients updates (in norm) to 1.0. For the proposed gating function, we initialize the dictionary from 10 elements equispaced in $[-4.0, 4.0]$.

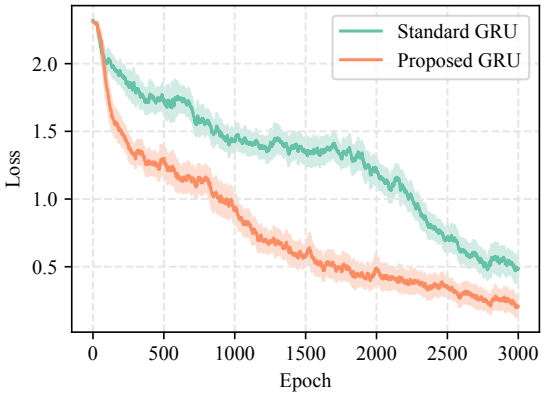
Early stopping is used to decide when to finish the optimization procedure. We keep the last 10000 elements of the training set as a validation part, and we compute the average accuracy of the model every 25 iterations, stopping whenever accuracy is not improving for at least 500 iterations. All the code is written in PyTorch and it is run using a Tesla K80 GPU on the Google Colaboratory platform.

4.2. Discussion of the results

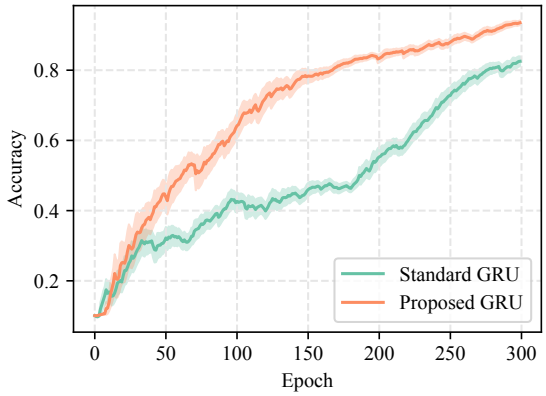
The results of the experiments averaged over 10 different runs are given in Tab. 1. We can see that the proposed GRU achieves higher test accuracy in all three cases (third column in Tab. 1). Interestingly, this difference is particularly significant for datasets with long temporal dependencies (P-MNIST and PP-MNIST). Here, the standard GRU also exhibits high standard deviations due to it converging to poorer minima in some cases. The proposed GRU is able to achieve high accuracy consistently over all runs.

We conjecture that this last result is also due to the higher flexibility allowed to the optimization procedure during training. To test this, we visualize in Fig. 3 the average loss and validation accuracy on the P-MNIST dataset of the two algorithms. We can see that the proposed GRU converges faster and more steadily, especially in the first half of training. This is consistent with the behavior found when using KAFs as activation functions, e.g., [15].

Fig 4 shows an histogram of the values of γ in (8) after training the proposed GRU on the P-MNIST dataset. Interestingly, the optimal architectures can benefit from a wide range of different bandwidths for the kernel. Looking back at



(a) Training loss



(b) Validation accuracy

Fig. 3. Convergence results on the P-MNIST dataset for a standard GRU and the proposed GRU. (a) Loss evolution on the training dataset (per iteration); (b) Validation accuracy (per epoch). The plots are focused on the first half of training. Shaded areas represent the variance.

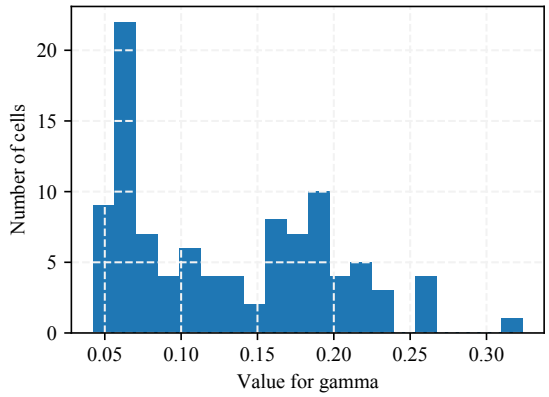


Fig. 4. Sample histogram of the values for γ in (8), after training, for the reset gate of the GRU.

Fig. 1, this translates in functions going from almost-linear to highly nonlinear behaviors.

4.3. Ablation study

To conclude our experimental section, we also perform a simple ablation study on the R-MNIST dataset, by training our proposed GRU with two modifications:

- **Rand**: we initialize the mixing coefficients randomly instead of following the identity initialization as in (11).
- **No-Residual**: we remove the residual connection from (10), leaving only $\sigma_{\text{KAF}}(s) = \sigma\left(\text{KAF}(s)\right)$.

Results of this set of experiments are provided in Fig. 5, where we also show with a red line the average test accuracy obtained by the standard GRU. We can see that removing the residual connection vastly degrades the performance, possibly because the resulting gating functions will revert to zero at their boundaries. Initializing the coefficients as the identity helps improving the accuracy by a lower margin, and also stabilizes it by reducing the variation of the results.

5. CONCLUSIONS

In this paper, we proposed an extension of the standard gating component used in most gated RNNs, e.g., LSTMs and GRUs. Specifically, we replace the element-wise sigmoid operation with a per-cell function endowed with a small number of parameters, that can adapt to the training data. To this end, we extend the kernel activation function in order to make its shape always consistent with a sigmoid-like behavior. The resulting function can be implemented easily in most deep learning frameworks, has a smooth behavior over its entire domain, and it imposes only a small computational overhead on the architecture. Experiments on a set of standard sequential problems with GRUs show that the proposed architecture achieve superior results (in terms of test accuracy), while at the same time converging faster (and reliably) in terms of number of iterations due to its increased flexibility.

Future research directions involve experimenting with other gated RNNs (possibly with different numbers of gates, layers, etc.), applications, and interpreting the resulting functions with respect to the task at hand. More in general, sigmoid-like functions are essential for many other deep learning components beside gated RNNs, including softmax functions for classification, attention-based architectures, and neural memories [19]. An interesting question is whether our

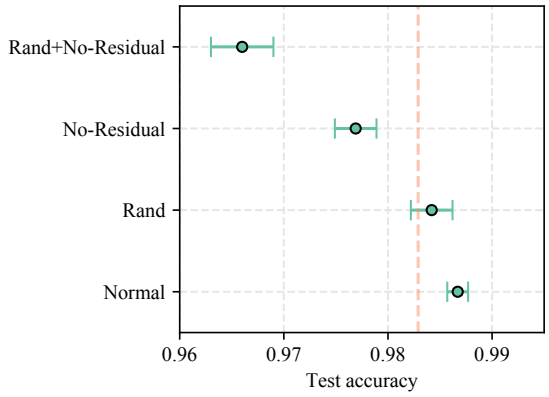


Fig. 5. Average results (in terms of test accuracy) of an ablation study on the R-MNIST dataset. **Rand**: we initialize the mixing coefficients randomly. **No-Residual**: we remove the residual connection in (10). With a dashed red line we show the performance of a standard GRU.

extended formulation can benefit (both in terms of accuracy and speed) these other architectures.

6. REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT Press Cambridge, 2016.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[4] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[5] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proc. IEEE International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2000, vol. 3, pp. 189–194.

[6] G.-B. Zhou, J. Wu, C.-L. Zhang, and Z.-H. Zhou, "Minimal gated unit for recurrent neural networks," *International Journal of Automation and Computing*, vol. 13, no. 3, pp. 226–234, 2016.

[7] J. van der Westhuizen and J. Lasenby, "The unreasonable effectiveness of the forget gate," *arXiv preprint arXiv:1804.04849*, 2018.

[8] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[9] C. Tallec and Y. Ollivier, "Can recurrent neural networks warp time?," in *Proc. 2018 International Conference on Learning Representations (ICLR)*, 2018.

[10] Y. Gao and D. Glowacka, "Deep gate recurrent neural network," *JMLR: Workshop and Conference Proceedings*, vol. 63, pp. 350–365, 2016.

[11] K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, "Depth-gated recurrent neural networks," *arXiv preprint arXiv:1508.03790*, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[13] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proc. 30th International Conference on Machine Learning (ICML)*, 2013.

[14] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.

[15] S. Scardapane, S. Van Vaerenbergh, S. Totaro, and A. Uncini, "Kafnets: kernel-based non-parametric activation functions for neural networks," *arXiv preprint arXiv:1707.04035*, 2017.

[16] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 1019–1027.

[17] D. Krueger and R. Memisevic, "Regularizing RNNs by stabilizing activations," in *Proc. 2016 International Conference on Learning Representations (ICLR)*, 2016.

[18] S. Zhang et al., "Architectural complexity measures of recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 1822–1830.

[19] A. Graves, G. Wayne, M. Reynolds, T. Harley, et al., "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471, 2016.