ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

Reservoir memory machines

Benjamin Paaßen and Alexander Schulz*

Machine Learning Group
Bielefeld University, Germany
Inspiration 1, 33619 Bielefeld - Germany

Abstract. In recent years, Neural Turing Machines have gathered attention by joining the flexibility of neural networks with the computational capabilities of Turing machines. However, Neural Turing Machines are notoriously hard to train, which limits their applicability. We propose reservoir memory machines, which are still able to solve some of the benchmark tests for Neural Turing Machines, but are much faster to train, requiring only an alignment algorithm and linear regression. Our model can also be seen as an extension of echo state networks with an external memory, enabling arbitrarily long storage without interference.

1 Introduction

While neural networks have achieved impressive successes in domains like image classification or machine translation, standard models still struggle with tasks that require very long-term memory without interference and would thus benefit from a separation of memory and computation [1, 2]. Neural Turing Machines (NTM) attempt to address these tasks by augmenting recurrent neural networks with an explicit memory to which the network has read and write access [1, 2]. Unfortunately, such models are notoriously hard to train, even compared to other deep learning models [2].

In our contribution, we propose to address this training problem by replacing the learned recurrent neural network controller of a NTM with an echo state network (ESN) [3]. In other words, we only learn the controller for the read and write head of our memory access as well as the output mapping, all of which is possible via standard linear regression. To construct the training data for our read and write head controllers, we only require a standard dynamic time warping alignment. We call this model a *reservoir memory machine* (RMM).

Our model can also be seen as an augmentation of echo state networks with an explicit external memory, such that input information can be stored for arbitarily long times without interference, whereas the maximum memory horizon for regular echo state networks is limited to the number of neurons in the reservoir [3, 4, 5].

In the remainder of this paper, we first refresh the reader's memory regarding standard ESNs, then formally define our own model - reservoir memory machines -, and finally show that our proposed model is sufficient to solve three benchmark tasks for Neural Turing Machines with much faster training.

*Funding by the Bielefeld Young Researchers' Fund and from BMBF within the project MechML under grant number 01IS18053E is gratefully acknowledged. We also thank Barbara Hammer for brilliant theoretical insights.

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

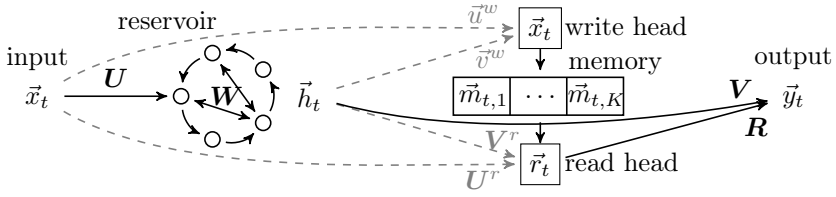

Figure 1: An illustration of reservoir memory machines. We first process the input (left) with a cycle reservoir with jumps (center left). We then use input and reservoir activations to control interaction with the memory (center right; gray connections). Finally, we feed reservoir activations and memory reads to the output (right).

2 Echo state networks

An echo state network (ESN) [3] is a recurrent network, i.e. the neural activations $\vec{h}_t \in \mathbb{R}^m$ at time t are computed as $\vec{h}_t = \tanh\Big(\boldsymbol{U} \cdot \vec{x}_t + \boldsymbol{W} \cdot \vec{h}_{t-1}\Big)$, where $\vec{x}_t \in \mathbb{R}^n$ is the input at time t, $\boldsymbol{U} \in \mathbb{R}^{m \times n}$ are the input weights, and $\boldsymbol{W} \in \mathbb{R}^{m \times m}$ are the recurrent weights of the network. The output $\vec{y}_t \in \mathbb{R}^L$ of the network at time t is computed as $\vec{y}_t = \boldsymbol{V} \cdot \vec{h}_t$, where $\boldsymbol{V} \in \mathbb{R}^{L \times m}$ are the output weights. ESNs have two distinct characteristics. First, $\boldsymbol{U}$ and $\boldsymbol{W}$ are not learned but kept fixed after initialization. This means that the activations $\vec{h}_1, \ldots, \vec{h}_T$ can be seen as a nonlinear preprocessing of the input, which makes learning $\boldsymbol{V}$ a generalized linear regression problem that can be solved analytically with the pseudo-inverse. Second, the recurrent weights $\boldsymbol{W}$ must ensure the *echo state property*, i.e. past influences must degrade over time [3, 6]. This property is necessary to ensure that the network's dynamic is independent of initial conditions and always adjusts to the input time series. On the other hand, it necessarily limits ESNs to *short term* memory tasks. In particular the memory is upper-bounded by the number of neurons n [3, 4]. This is the key limitation we aim to address.

In this paper, we employ the deterministic 'cycle reservoir with jumps' scheme to initialize $\boldsymbol{U}$ and $\boldsymbol{W}$ [6]. In this scheme, the entries of $\boldsymbol{U}$ are set to a constant value $u \in (-1, 1)$ with a sign determined by a fixed, aperiodic sequence (e.g. the digits of pi), and $\boldsymbol{W}$ is a sparse matrix with off-diagonal cycle connections $w_{i,i+1} = w_c \in [0, 1)$ and longer 'jump' connections $w_{i,i+l} = w_{i+l,i} = w_l \in [0, 1)$. Note that u, w_c, w_j, and $l \in \mathbb{N}$ are hyper-parameters of the model. Because this initialization is deterministic, we can compare different architectures more easily. In general, however, our architecture is agnostic regarding the initialization.

3 Reservoir memory machines

Our key contribution is an easy-to-train alternative to the Neural Turing Machine [1]. In particular, we propose to extend an ESN with an explicit memory, a write head, which can copy inputs into memory, and a read head, which can

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

read from the memory. We call this augmented ESN version a *reservoir memory machine* (RMM). A sketch of the RMM architecture is shown in Figure 1.

In more detail, the state of our system is now a quadruple $(\vec{h}_t, \boldsymbol{M}_t, k_t, l_t)$, where $\vec{h}_t$ are the reservoir activations as before, $\boldsymbol{M}_t \in \mathbb{R}^{K \times n}$ is the current memory state (of size K), and $k_t, l_t \in \{1, \ldots, K\}$ are the current position of the write and read head respectively.

The dynamics of the system are as follows. First, we copy the previous memory state, i.e. $\boldsymbol{M}_t \leftarrow \boldsymbol{M}_{t-1}$ (where $\boldsymbol{M}_{-1} = \boldsymbol{0}$). Then, we control the write head with the value $c^w_t = \vec{u}^w \cdot \vec{x}^t + \vec{v}^w \cdot \vec{h}_t$, where $\vec{u}^w \in \mathbb{R}^n$ and $\vec{v}^w \in \mathbb{R}^m$ are learnable parameters. If $c^w_t > 0$, we write to the memory, i.e. $\vec{m}_{t,k} \leftarrow \vec{x}_t$, and increment $k_t \leftarrow k_{t-1} + 1$ (re-setting k_t to 1 if it exceeds K). Otherwise, we leave the memory and k_t as is. Similarly, in each time step we control the read head with the vector $\vec{c}^r_t = \boldsymbol{U}^r \cdot \vec{x}_t + \boldsymbol{V}^r \cdot \vec{h}_t$, where $\boldsymbol{U}^r \in \mathbb{R}^{3 \times n}$ and $\boldsymbol{V}^r \in \mathbb{R}^{3 \times m}$ are learnable parameters. If $c^r_{t,1} = \max\{c^r_{t,1}, c^r_{t,2}, c^r_{t,3}\}$, the read head stays in the same location, i.e. $l_t \leftarrow l_{t-1}$; if $c^r_{t,2} = \max\{c^r_{t,1}, c^r_{t,2}, c^r_{t,3}\}$, we increment $l_t \leftarrow l_{t-1} + 1$ (re-setting l_t to 1 if it exceeds K); otherwise, we re-set $l_t \leftarrow 1$. We then set the memory read at time t as the l_tth row of $\boldsymbol{M}_t$, i.e. $\vec{r}_t \leftarrow \vec{m}_{t,l_t}$.

The output of the system at time t is $\vec{y}_t = \boldsymbol{V} \cdot \vec{h}_t + \boldsymbol{R} \cdot \vec{r}_t$, where $\boldsymbol{V} \in \mathbb{R}^{L \times m}$ and $\boldsymbol{R} \in \mathbb{R}^{L \times n}$ are learnable parameters. Note that our proposed model is a strict extension of an ESN because we can simply set $\boldsymbol{R} = \boldsymbol{0}$ and thus obtain a standard ESN. However, we can potentially solve *more* tasks.

Training: Because the output generation depends on the memory content, our first step is to train the write and read heads, i.e. the parameters $\vec{u}^w$, $\vec{v}^w$, $\boldsymbol{U}^r$, and $\boldsymbol{V}^r$. In more detail, we initialize $\boldsymbol{R}$ as the identity matrix (padded with zeros whenever necessary) and then identify for each output $\vec{y}_t$ the earliest input $\vec{x}_{\tau_t}$ that minimizes the distance $\|\boldsymbol{R} \cdot \vec{x}_{\tau_t} - \vec{y}_t\|$. Based on this, we generate an *ideal* control sequence for the write head $c^w_1, \ldots, c^w_T$ where $c^w_t = +1$ if $t \in \{\tau_1, \ldots, \tau_T\}$ and $c^w_t = -1$ otherwise. This control sequence serves as our teaching signal for training $\vec{u}^w$ and $\vec{v}^w$ via linear regression.

Next, we generate the tensor of all memory states $(\boldsymbol{M}_1, \ldots, \boldsymbol{M}_T) \in \mathbb{R}^{T \times K \times n}$ as described above. We then align this tensor with the output time series $\vec{y}_1, \ldots, \vec{y}_T$ via a variant of dynamic time warping with the recurrence: $d_{l,t} = \|\boldsymbol{R} \cdot \vec{m}_{t,l} - \vec{y}_t\| + \min\{d_{l,t+1}, d_{l+1,t+1}, d_{1,t+1}\}$, where the entries in the minimum correspond respectively to leaving the read-head location as is, incrementing it, or resetting it to one. The base case of this recurrence is $d_{l,T+1} = 0$ for all $l \in \{1, \ldots, K\}$. Note that $\min\{d_{1,1}, d_{2,1}\}$ then corresponds to the error we achieve by optimally moving the read head over the memory and always predicting the output $\boldsymbol{R} \cdot m_{t,l_t}$. Accordingly, backtracing yields a teaching signal to train the read head parameters $\boldsymbol{U}^r$ and $\boldsymbol{V}^r$ via linear regression.

Finally, we compute the sequence of memory reads $\vec{r}_1, \ldots, \vec{r}_T$ as described above, which we use to train both $\boldsymbol{V}$ and $\boldsymbol{R}$ via linear regression. Now, because we change $\boldsymbol{R}$, the optimal alignments in the previous steps may change as well. Accordingly, we repeat the training process until the loss increases or until convergence, yielding an alternating optimization algorithm.

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

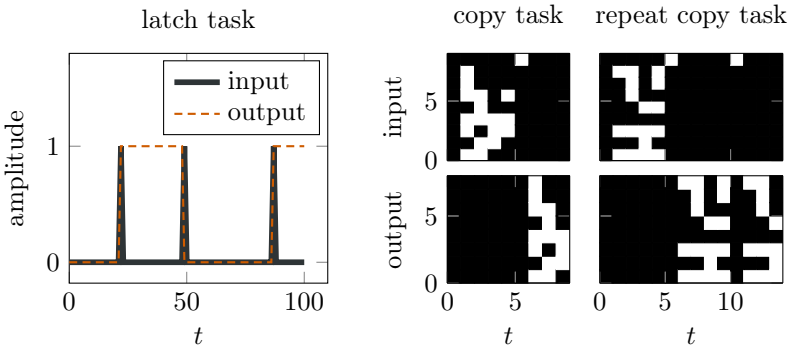

Figure 2: An example input and output sequence for all three data sets.

4 Experiments

In our experiments, we evaluate reservoir memory machines (RMMs) on three data sets that require storage of inputs over long times without interference:

The *latch* task requires to produce zeros until a spike in the input appears, after which the model should produce ones. For the next input spike, the model should switch back to zeros, and so on (Figure 2, left). We use three spikes with random positions and random sequence lengths of up to 200 time steps.

The *copy* data set [1] consists of 1-20 time steps with 8 random bits each, followed by a sequence end token in an additional feature. After this, the goal is to exactly copy the input while the remaining input is zero (Figure 2, center).

The *repeat copy* data set [1] extends the copy task by requiring the network to copy the input sequence multiple times (refer to Figure 2, right).

We compare RMMs to standard ESNs and to a novel variant which we dub echo state gated recurrent unit (ESGRU). This model uses the dynamic equations of a gated recurrent units [7] but keeps all weights fixed after initialization. To ensure that all variance is due to memory access only, we use the same reservoir for all networks, namely a cycle reservoir with jumps [6].

We evaluate in a 20 fold crossvalidation, generating 10 sequences per fold (i.e. 190 training sequences and 10 test sequences). For each model, we used a 3-fold nested crossvalidation for hyper-parameter optimization via random search with 10 trials. The detailed experimental code is available at `https://gitlab.ub.uni-bielefeld.de/bpaassen/reservoir-memory-machines`.

The generalization root mean square error (RMSE) of all models on all datasets is displayed in Table 1. For all datasets, RMMs achieve a low (albeit nonzero) error, indicating that RMMs are able to solve the tasks. Additionally, we note that both ESNs and ESGRUs are *not* able to solve the tasks, because they have significantly higher errors in all datasets ($p < 10^{-3}$ according to a Wilcoxon sign-rank test with Bonferroni correction). Note that a Neural Turing Machine achieves zero error on all tasks [2].

We investigate the solution strategy of the RMM model in more detail on the

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

Table 1: The average RMSE ($\pm$ standard deviation) across 20 crossvalidation folds for all models and all data sets. NTM results are copied from [2].

model	**latch**	**copy**	**repeat copy**
ESN	0.309 ± 0.049	0.358 ± 0.030	0.409 ± 0.038
ESGRU	0.402 ± 0.116	0.331 ± 0.011	0.375 ± 0.018
RMM	$< 10^{-3}$	0.027 ± 0.025	0.037 ± 0.067
NTM	n.a.	$< 10^{-3}$ [2]	$< 10^{-3}$ [2]

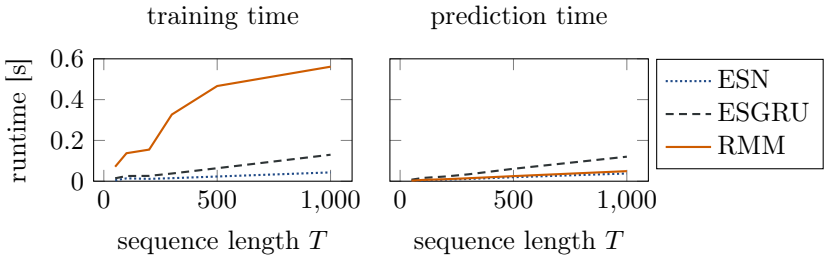

Figure 3: Runtime results for training (left) and prediction (right) of standard ESNs, ESGRUs, and RMMs for varying sequence length.

latch task. For this purpose, we use a trained RMM and let it extrapolate to a much longer sequence (see Figure 4, top) than seen in training (length 1700 vs. 200 with 8 vs. 3 spikes). We note that the RMM extrapolates perfectly (Figure 4, second row) with an error $< 10^{-3}$. In more detail, we observe that the model only writes to memory once, namely storing a 1 at the time of the first spike (Figure 4, third row), whereas the read head switches position at every spike (except the first one; Figure 4, bottom), thus producing the desired output.

To evaluate the runtime, we train ESNs, ESGRUs, and RMMs with a reservoir of 128 neurons each on a random 8-bit input sequence with varying length, the output sequence being shifted by one. We measure runtime on a consumer grade laptop with core i7 CPU. Figure 3 shows the runtime results. We find that RMMs roughly take 15 times longer to train compared to regular ESNs, which may be due to more needed linear regression runs and an inefficient alignment implementation. Still, even for long sequences we maintain training times well below a second. Prediction time is roughly comparable to a standard ESN and faster than an ESGRU. By comparison, training a NTM using the reference implementation [2] on the copy task took more than 30 minutes.

5 Conclusion

We have introduced reservoir memory machines (RMMs), which augment echo state networks with an external memory, a write head that copies data from

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

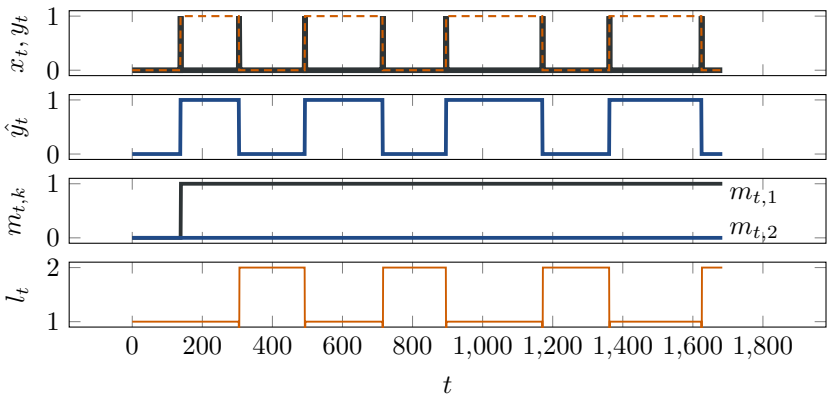

Figure 4: From top to bottom: A long sequence from the latch task with the input as solid, the output as dashed line; the prediction of the RMM; the memoy entries over time; and the read head position over time.

the input to the memory, and a read head which couples the memory to the output. We also provided a training algorithm for the write and read heads based on dynamic time warping and linear regression in an alternating optimization scheme. As such, our model retains the training simplicity of echo state networks, but extends its capabilities to some of the benchmark tasks of Neural Turing Machines. We emphasize that our model is still strictly less powerful because other benchmark tasks remain out of reach, especially those based on content-based addressing. Extending our model with such a mechanism is a task for future work. Further, we still require a formal proof that our proposed model is strictly more powerful than an ESN.

References

[1] Alex Graves, Greg Wayne, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[2] Mark Collier and Joeran Beel. Implementing Neural Turing Machines. In *Proceedings of the ICANN 2018*, pages 94–104, 2018.

[3] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[4] Igor Farkaš, Radomír Bosák, and Peter Gergeľ. Computational analysis of memory capacity in echo state networks. *Neural Networks*, 83:109 – 120, 2016.

[5] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Design of deep echo state networks. *Neural Networks*, 108:33 – 47, 2018.

[6] Ali Rodan and Peter Tiňo. Simple deterministically constructed cycle reservoirs with regular jumps. *Neural Computation*, 24(7):1822–1852, 2012.

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the EMNLP 2014*, pages 1724–1734, 2014.