Neural Networks 23 (2010) 239–243

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Temporal-Kernel Recurrent Neural Networks

Ilya Sutskever *, Geoffrey Hinton

Department of Computer Science, University of Toronto, Canada

ARTICLE INFO

Article history:
Received 30 April 2008
Revised and accepted 26 October 2009

Keywords:
Recurrent Neural Networks
Fixed points
Long-term dependencies
Backpropagation through time
Supervised learning

ABSTRACT

A Recurrent Neural Network (RNN) is a powerful connectionist model that can be applied to many challenging sequential problems, including problems that naturally arise in language and speech. However, RNNs are extremely hard to train on problems that have long-term dependencies, where it is necessary to remember events for many timesteps before using them to make a prediction.

In this paper we consider the problem of training RNNs to predict sequences that exhibit significant long-term dependencies, focusing on a serial recall task where the RNN needs to remember a sequence of characters for a large number of steps before reconstructing it. We introduce the Temporal-Kernel Recurrent Neural Network (TKRNN), which is a variant of the RNN that can cope with long-term dependencies much more easily than a standard RNN, and show that the TKRNN develops short-term memory that successfully solves the serial recall task by representing the input string with a stable state of its hidden units.

1. Introduction

Recurrent Neural Networks (RNNs) are connectionist models that operate in discrete time using feedback connections. An RNN has a set of units, each taking a real value in each timestep, and a set of weighted connections between its units. The input units are set by the environment and the output units are computed using the connection weights and the hidden units.

RNNs have nonlinear dynamics, allowing them to behave in a highly complex manner. In principle, the states of the hidden units can store information through time in the form of a distributed representation and this distributed representation can be used many timesteps later to predict subsequent input vectors.

RNNs are appealing because of their range of potential applications: they can be applied to almost any problem with sequential structure, including the problems that arise naturally in speech, control, and natural language processing. Since RNNs can represent highly complex functions of sequences, these problems are likely to be solvable with some RNN, so a learning algorithm that can find this RNN would be very useful in practice.

Unfortunately, RNNs have proved to be difficult to learn with gradient descent, especially when the sought for RNN must use its units to store events for more than a few timesteps. Whenever events in the far past are relevant for predicting the current timestep, the problem is said to exhibit long-term dependencies. It is known (Bengio, Simard, & Frasconi, 1994; Hochreiter, 1991) that gradient descent has great difficulty in learning weights that make use of long-term dependencies, so the resulting RNNs are typically no more useful than a simple moving window.

In this paper, we address the problem of learning RNNs that successfully predict sequences that exhibit long-term dependencies. In particular, we focus on a serial recall task in which an arbitrary sequence of characters must be stored for a variable length of time until a cue is presented. After the cue is presented, the RNN must reproduce the stored sequence. The variable time delay makes it very hard to solve this problem using delay lines, so the RNN must learn to convert the arbitrary sequence to a stable distributed pattern of activity and then convert this stable pattern back into the appropriate sequence when the recall cue arrives.

Our main contribution is a new family of RNNs, the Temporal-Kernel Recurrent Neural Network (TKRNN), in which every unit is an efficient leaky integrator, which makes it easier to "notice" long-term dependencies: we demonstrate that the TKRNN can learn to use its units to store 35 bits of information for at least 50 timesteps in an immediate serial recall task (e.g., Botvinick & Plaut, 2006). The TKRNN learns to represent its input with a stable state of its hidden units, which is relevant to a line of research that uses the fixed points of biologically-plausible neural networks to represent useful information, such as shape (Amit, 1995) or eye position (Seung, 1996; Camperi & Wang, 1998), for extended periods of time. Our experiments show that the TKRNN's architecture is suitable for such problems, and that its performance is comparable to that of the Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) when applied to the same tasks.

* Corresponding author.
E-mail addresses: ilya@cs.utoronto.ca (I. Sutskever), hinton@cs.utoronto.ca (G. Hinton).

0893-6080/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2009.10.009

2. Standard Recurrent Neural Networks: Definitions

In this section we formally define the standard RNN (Werbos, 1990; Rumelhart, Hinton, & Williams, 1986). The RNN is a neural network that operates in time. At each time t, the value of the RNN's input units is given by $\mathbf{x}_t$, and the RNN computes the values of its hidden units ($\mathbf{y}_t$) and output units ($\mathbf{z}_t$) by the equations

$$\mathbf{y}_t = f\left(W_{\mathbf{y}\to\mathbf{y}}\mathbf{y}_{t-1} + W_{\mathbf{x}\to\mathbf{y}}\mathbf{x}_t\right) \tag{1}$$

$$\mathbf{z}_t = g\left(W_{\mathbf{z}\to\mathbf{y}}\mathbf{y}_t + W_{\mathbf{x}\to\mathbf{z}}\mathbf{x}_t\right) \tag{2}$$

where $W_{\mathbf{z}\to\mathbf{y}}$ is a matrix of size $n_z \times n_y$ of the weights of the connections between the hidden units $\mathbf{y}$ and the output units $\mathbf{z}$. A common choice for the functions $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^n$ is the sigmoid and the softmax functions, which are

$$f(\mathbf{x})^{(i)} = \frac{1}{1+\exp(-\mathbf{x}^{(i)})} \tag{3}$$

$$g(\mathbf{x})^{(i)} = \frac{\exp(\mathbf{x}^{(i)})}{\sum_{j=1}^{n}\exp(\mathbf{x}^{(j)})} \tag{4}$$

where in these equations $\mathbf{x}$ is a generic n-dimensional vector and $\mathbf{x}^{(i)}$ is its i'th coordinate. Other definitions of f and g are also possible. Thus, given a setting of the RNN's parameters, these equations completely determine the values of the RNN's hidden ($\{\mathbf{y}_t\}_t$) and output ($\{\mathbf{z}_t\}_t$) units for any sequence of input vectors $\{\mathbf{x}_t\}_t$.

An RNN can be trained to approximate a highly complex function on sequences using a collection of training input sequences $\{\mathbf{x}_t\}_t$ and their corresponding desired outputs $\{\mathbf{v}_t\}_t$. The problem of learning the RNN's connection weights is formulated as a problem of minimizing a cost function C that measures the RNN's deviation from perfect behavior:

$$C = \sum_{t=1}^{T} C_t = \sum_{t=1}^{T} c(\mathbf{z}_t, \mathbf{v}_t) \tag{5}$$

where T is the length of the sequence, and $c(\mathbf{z}_t, \mathbf{v}_t)$ is a measure of distance between the desired output and the actual output (we use the cross-entropy $c(\mathbf{x}, \mathbf{z}) = -\sum_i \mathbf{z}^{(i)} \log \mathbf{x}^{(i)}$).

3. Temporal-Kernel Recurrent Neural Networks

The backpropagation through time algorithm (BPTT) (Werbos, 1990; Rumelhart et al., 1986) can efficiently compute the gradient of the RNNs cost function (Eq. (5)), so it may seem that RNNs should be easy to learn. However, if the RNN must learn to remember events in the far past in order to make accurate predictions about the present where the relevant events are always separated by many timesteps, the RNN learned by BPTT will fail to use its hidden units to store the important relevant information from the past.

A theoretical analysis (Bengio et al., 1994; Hochreiter, 1991) shows that learning is hard because the past is separated from the present with a large number of nonlinearities, causing the gradient to get "diluted" and uninformative as it flows backwards through time. If we allow the gradient to skip timesteps as it flows backwards, it can influence the past more directly and be less diluted, which is easily achieved by adding direct connections between units that are separated in time: if k is not too large ($<k_0$), we connect $\mathbf{y}_{t-k}$ and $\mathbf{y}_t$ with connections whose weights are independent of t. By adding these connections, we essentially obtain the NARX RNN (Lin, Horne, Tino, & Giles, 2000).

These additional connections allow the NARX RNN to learn long-term regularities that are k_0 times more separated in time than the standard RNN, which can be substantial when k_0 is large. However, NARX RNNs have two drawbacks. First, NARX RNNs are k_0 times slower than RNNs (per iteration), and second, NARX RNNs have k_0 times more parameters than a standard RNN with the same number of units. This is particularly costly because k_0 often needs to be large, so the NARX RNN mitigates the problem of learning at the expense of being slower and larger.

Our contribution is a new family of RNNs that has many of the advantages of the NARX RNN without its disadvantages. We introduce the Temporal-Kernel Recurrent Neural Network (TKRNN), which is an RNN with direct connections between units in all timesteps (from $\mathbf{y}_t$ to $\mathbf{y}_{t'}$ for all $t' < t$), which is as efficient as the standard RNN and has almost the same number of parameters. We also introduce the TKRNN^{+n}, which is an RNN whose weights are the sum of the weights of n TKRNNs; the TKRNN^{+n} is n times slower than the TKRNN per forward/backward pass, but it finds considerably better solutions.

The main idea of the TKRNN is to make each of its units act as a leaky integrator while keeping the forward and the backward pass efficient. The equation that governs the TKRNN's hidden units is

$$\mathbf{y}_t{}^{(i)} = f\left(\sum_{k=1}^{t}\left(\sum_{j=1}^{n_y}(\lambda^{(j)})^{k-1}W_{\mathbf{y}\to\mathbf{y}}{}^{(j,i)}\mathbf{y}_{t-k}^{(j)} + \sum_{m=1}^{n_x}(\lambda^{(m)})^{k-1}W_{\mathbf{x}\to\mathbf{y}}{}^{(m,i)}\mathbf{x}_{t-k}^{(m)}\right)\right) \tag{6}$$

where $0 < \lambda^{(j)} < 1$ is an additional parameter for each unit j that determines the extent to which the unit can directly influence units in future timesteps, or, equivalently, the extent to which units are influenced by unit j's activities in previous timesteps.[1] An analogous equation defines the output units.

The TKRNN has connections between units in all timesteps which make the gradient flow through less nonlinearities, but rather than being arbitrary, the connections' weights are factored in space and time: the weight of the connection between unit $\mathbf{y}_t^{(j)}$ and $\mathbf{y}_{t-k}^{(i)}$ is $W_{\mathbf{y}\to\mathbf{y}}{}^{(j,i)}(\lambda^{(j)})^{k-1}$, which ensures that the number of parameters is small and that the forward and the backward passes can be performed efficiently. Eq. (6) implies that the units' input is an average of the units' past activities weighted by the exponential kernel.

Previous work in which units were similarly connected through time with a kernel include (Hinton & Brown, 2000; Natarajan, Huys, Dayan, & Zemel, 2008), although they were used in a different context. Notably, the Gamma model (De Vries & Principe, 1992) is a closely related RNN architecture that uses a related family of kernels.

The TKRNN's definition causes the forward and backward passes to be as efficient as those of a standard RNN: notice that Eq. (6) can be rewritten as

$$\mathbf{y}_t{}^{(i)} = f\left(\sum_{j=1}^{n_y}W_{\mathbf{y}\to\mathbf{y}}{}^{(j,i)}\sum_{k=1}^{t}(\lambda^{(j)})^{k-1}\mathbf{y}_{t-k}^{(j)} + \sum_{m=1}^{n_x}W_{\mathbf{x}\to\mathbf{y}}{}^{(m,i)}\sum_{k=1}^{t}(\lambda^{(m)})^{k-1}\mathbf{x}_{t-k}^{(m)}\right) \tag{7}$$

so if we define

$$\mathbf{S}_t^{\mathbf{y}(j)} = \sum_{k=1}^{t}(\lambda^{(j)})^{k-1}\mathbf{y}_{t-k}^{(j)} \tag{8}$$

$$\mathbf{S}_t^{\mathbf{x}(m)} = \sum_{k=1}^{t}(\lambda^{(m)})^{k-1}\mathbf{x}_{t-k}^{(m)} \tag{9}$$

[1] We slightly abuse notation and treat the variables $\lambda^{(j)}$ and $\lambda^{(m)}$ as distinct.

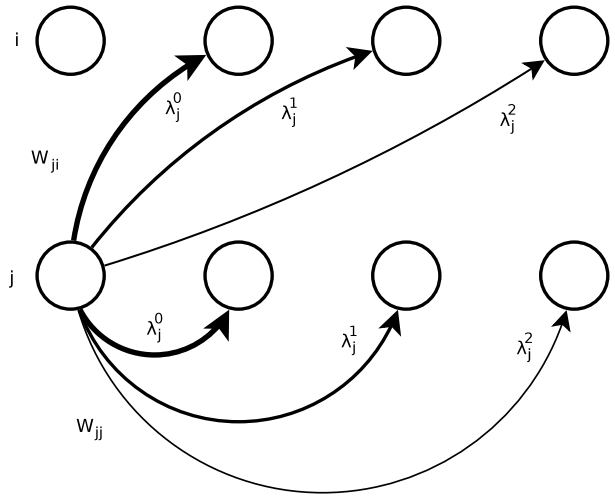

Fig. 1. This figure illustrates the TKRNN's connectivity: The more units are separated in time, the weaker is their connection.

where $\mathbf{S^y}_0$ and $\mathbf{S^x}_0$ are the zero vectors, then $\mathbf{S^y}_t$ and $\mathbf{S^x}_t$ can be easily computed by the equation

$$\mathbf{S^y}_t^{(j)} = \mathbf{y}_{t-1}^{(j)} + \lambda^{(j)}\mathbf{S^y}_{t-1}^{(j)} \tag{10}$$

$$\mathbf{S^x}_t^{(m)} = \mathbf{x}_{t-1}^{(m)} + \lambda^{(m)}\mathbf{S^x}_{t-1}^{(m)}. \tag{11}$$

The variables $\mathbf{S^y}_t$ and $\mathbf{S^x}_t$ act as leaky integrators in the equations above.

The TKRNN's backward pass is derived straightforwardly, and is computed using a similar recursive equation.[2]

3.1. Remarks on the TKRNN and the TKRNN$^+$

In this section we comment on the TKRNN's definition. Consider the exponential form of Eq. (6) (see also Fig. 1) which implies that units that are sufficiently separated in time have a negligible weight on their direct connection, so the TKRNN effectively connects each unit with only a finite (but large) number of timesteps directly. This prevents the TKRNN from learning long-term dependencies that are significantly outside the reach of its direct connections for the same reason an RNN cannot learn dependencies spanning more than a few timesteps (in sharp contrast to the LSTM; Hochreiter & Schmidhuber, 1997). This is partly alleviated by learning the λ's which enables the weights to choose the timescale to operate on. However, if the initial λ's are too small, then the gradient with respect to λ will be negligible, so it is essential to initialize the λ's in the right scale. As the λ's get closer to 1, the units become capable of directly extracting information from the very far past in an extremely "coarse" way, so the λ's should be as small as possible.

To enforce the constraint that $0 < \lambda^{(j)} < 1$, we parameterized λ's via the sigmoid function: $\lambda^{(j)} = 1/(1+\exp(-\ell^{(j)}))$, and learned the unconstrained ℓ variables. Note, further, that if we set $\lambda^{(j)}$ to 0 for every unit, we recover the standard RNN. Interestingly, the TKRNN's definition (as well as its backward pass) is valid when $\lambda^{(j)}$ is negative, which can make it easier for its units to detect abrupt changes in the past.

In addition, we can view the TKRNN$^+$ as a TKRNN whose exponential kernel from unit i to unit j is replaced with a weighted sum of k exponential kernels of different scales and magnitudes, whose shape is a function of both i and j, unlike the TKRNN where the kernel's shape is a function of j only. Sums of exponential kernels are significantly more expressive than a single exponential kernel (see Fig. 2), and enable the TKRNN$^+$ to have both long-term connections and detailed short-term dynamics.

And finally, the TKRNN's leaky integrators make it more biologically plausible than standard RNNs (Shepherd, 1998, chapter 2).

4. Related work

In this section we mention additional related work on sequence modeling with RNNs that are designed to cope with long-term dependencies.

The Echo-State Network (ESN) (Jaeger & Haas, 2004) uses a large number of neurons with random connectivity to compute many "basis functions" of both the input sequence and its past outputs. The learned output weights combine these basis functions in the best way to predict the output. During prediction, the true outputs (which are unknown) are replaced with the predicted outputs, which are fed to the recurrent units. ESNs work extremely well for predicting chaotic systems for a large number of timesteps, and their linear version has been rigorously analyzed (White, Lee, & Sompolinsky, 2004). The major advantage of the ESN is that its recurrent weights are not learned, so learning is extremely fast.

Perhaps surprisingly, standard RNNs trained by the Extended Kalman Filter (EKF) training procedure (Williams, 1992) are shown to successfully learn standard RNNs on problems exhibiting long-term dependencies (e.g., Prokhorov, Feldkamp, & Tyukin, 2002; Feldkamp, Prokhorov, & Feldkamp, 2003). However, while the EKF training procedure is a powerful learning procedure, it is applicable to only modestly-sized RNNs due to its computational complexity.

The Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997; Gers, Schraudolph, & Schmidhuber, 2002; Gers, Schmidhuber, & Cummins, 2000) is an RNN architecture that is capable of learning problems with significant long-term dependencies by using explicit memory units. The design of the memory units allows the gradient to freely flow backwards in time, possibly for an unlimited duration. The LSTM can solve problems where important pieces of information are always separated by at least thousands of timesteps, and was successfully applied to various tasks, including an instance of robotic control (Mayer, Gomez, Wierstra, Nagy, Knoll, & Schmidhuber, 2006). The LSTM's ability to cope with long-term dependencies is also confirmed in our experiments.

5. Experimental results

The goal of this section is to demonstrate that the TKRNN can learn to predict sequences that exhibit challenging long-term dependencies. For this purpose, we chose a difficult serial recall task (e.g., Botvinick & Plaut, 2006) which is to predict sequences containing two copies of a random 15 characters-long word (with an alphabet of 5 symbols) that are separated by a gap of at least 50 timesteps. Such sequences can be predicted only if the RNN remembers the word for a sufficient amount of time, which requires 35 bits of memory. We show that the TKRNN does well on this problem, and its performance is comparable to that of the LSTM.

5.1. Task and network details

We describe the serial recall task. To generate an example sequence, we choose a random 15-character word from a 5-character alphabet. It is followed by 40 spaces and an additional small random number of spaces which prevent the TKRNN from using direct delay lines. The random number is geometrically distributed

[2] Our python implementation of the TKRNN^{+n} and the experiments can be found in www.cs.utoronto.ca/~ilya/code/TKRNN.tar.

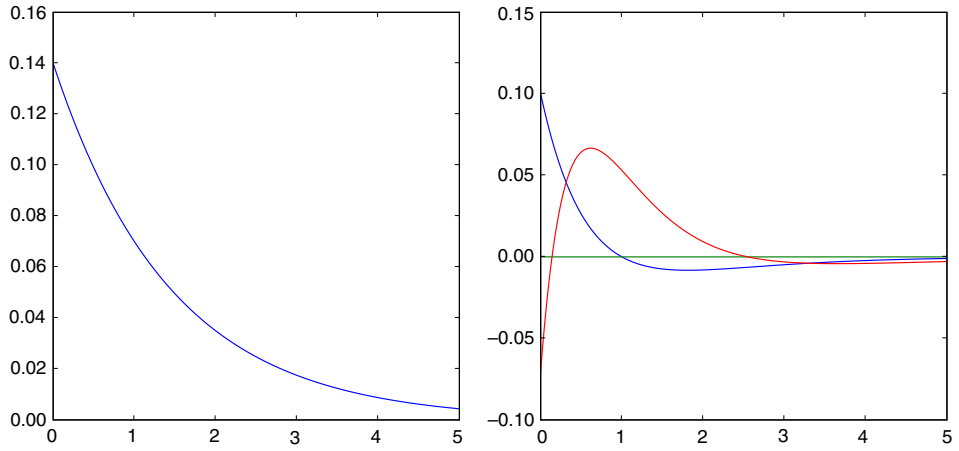

Fig. 2. The figure on the left shows an exponential kernel and figure on the right shows the sums of two and three exponential kernels. This illustrates the additional expressive power obtained from using sums of exponential kernels.

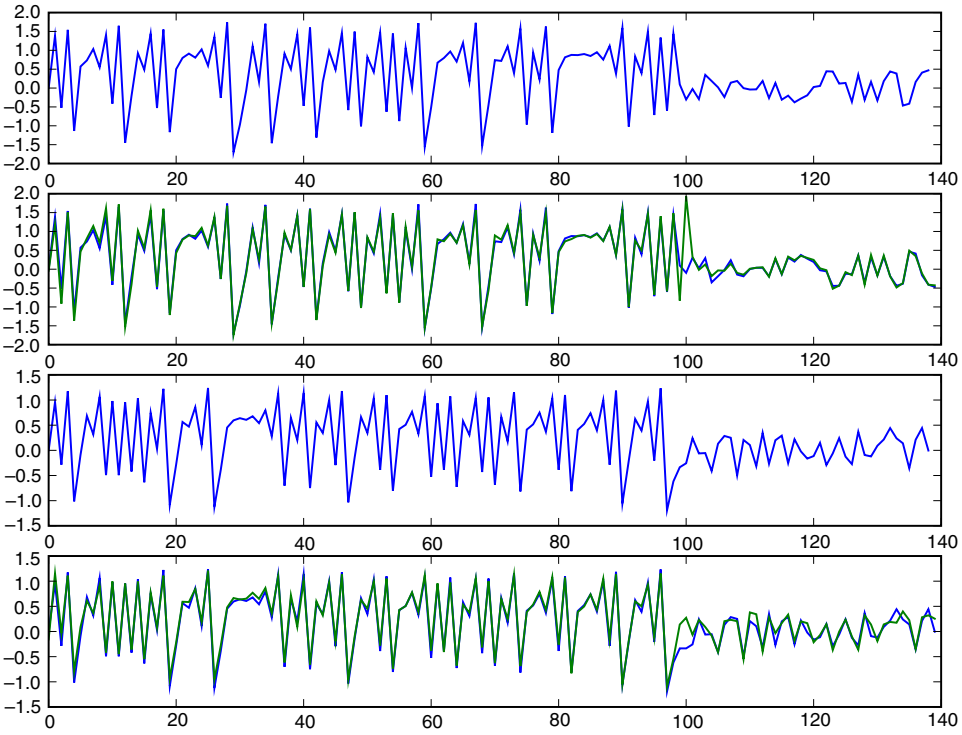

Fig. 3. The figure shows four panels. Panels 1 and 3 show an input sequence of the conditioned adaptive behavior task. Panels 2 and 4 show the correct output sequence for their corresponding input sequence (blue) and the outputs predicted by the TKRNN (green). In this task, the initial 100 timesteps of the sequence determine (as described in Section 3.1 of Feldkamp et al., 2003) one of two possible functions ($f(x) = x$ or $f(x) = -x$). The last 40 timesteps of the input are random, while the corresponding outputs are obtained by applying the encoded function f to each input. This figure shows that the TKRNN computes the function $f(x) = -x$ (panels 1, 2) less accurately than $f(x) = x$ (panels 3, 4). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

with mean 1.25.[3] The spaces are followed by a special character, 10 spaces, and another presentation of the same random word. The special character notifies the model of the second occurrence of the word starting 11 timesteps later. We truncate the rare sequences whose length exceeds 100 and use a 1-of-7 encoding to represent each symbol.

We used a TKRNN^{+5} with 100 hidden units and an LSTM with 50 memory blocks, each of which has two memory cells. We updated the parameters on every sequence for 10^6 weight updates; note that 10^6 is a very small fraction of 5^{15} so there is no danger of overfitting. The TKRNN's weights and the LSTM's weights were initialized with a spherical Gaussian with small variance, while the TKRNN's ℓ parameters were independently sampled from a mixture of a uniform [0, 1] and a uniform [0, 5] distribution. The TKRNN's learning rates were 10^{-5} for its standard weights and 10^{-7} for the ℓ parameters, while the LSTM's learning rate was set to 10^{-4}. A momentum of 0.9 was used for both models. Both models were trained to maximize the log probability of the next timestep given the previous timesteps.

We report results as follows: for each character in the second occurrence of x, we determine the most likely character according to the model, and report the fraction of characters predicted correctly. This way, the TKRNN predicts 79% and the LSTM predicts 81% of the characters correctly.

[3] The geometric distribution is defined by $P(n) = (1-p)p^n$ and its mean is $1/(1-p)$.

If, instead, we consider a prediction to be correct if the target answer is among the two most likely characters according to the model, then the TKRNN predicts 97% and the LSTM predicts 98% of the characters in the second occurrence of x correctly.

5.2. Additional experiments

While the paper's focus is on the serial recall task because of its obvious long-term dependencies, we briefly mention the results of two additional experiments. We applied the TKRNN to the "Conditioned adaptive behavior" task (Feldkamp et al., 2003, Section 3.1). In this task, the (essentially) first half of an input sequence determines a function which is applied to every timestep of the second half of the sequence, producing a sequence of target outputs for the network. This task is nontrivial because the network needs to notice a regularity in the first half of the sequence, to remember it for enough timesteps, and to repeatedly evaluate a function that depends on this memory.

The TKRNN successfully coped with the long-term dependency of the task and achieved a test RMSE of 0.16 (this is seen in Fig. 3), which is slightly better than the 0.2 test RMSE of the LSTM. Note that (Feldkamp et al., 2003) reports a 0.1 *training* RMSE on the same task, so our results are in the correct range. In contrast, the multiple quadratic function prediction task (Prokhorov et al., 2002), where every sequence is assigned a randomly chosen 2-variable quadratic function and consists of input–output examples from it, was much harder for the TKRNN: it failed to solve the task, while the LSTM succeeded (the LSTM results are reported by Younger, Hochreiter, and Conwell (2001)).[4] Nevertheless, a standard RNN was able to solve the multiple quadratic function prediction task when trained with the extended Kalman filter (EKF) (see Prokhorov et al., 2002); therefore, the TKRNN should also succeed if trained with the expensive EKF procedure because the standard RNN is a special case of the TKRNN (although the EKF will boost the performance of the LSTM as well).

6. Conclusions

We presented a new family of Recurrent Neural Networks and showed that their construction makes them suitable for problems that exhibit difficult long-term dependencies. The performance is usually comparable to the LSTM. In particular, our results show that leaky integrators are not as susceptible to the vanishing gradient problem as has generally been supposed and that these networks, which are biologically more plausible than LSTM, can learn to store information in the fixed points of the dynamics of their hidden state.

References

Amit, D. (1995). The Hebbian paradigm reintegrated: Local reverberations as internal representations. *Behavioral and Brain Sciences*, *18*(4), 617–657.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, *5*(2), 157–166.

Botvinick, M., & Plaut, D. (2006). Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, *113*(2), 201–233.

Camperi, M., & Wang, X. (1998). A model of visuospatial working memory in prefrontal cortex: Recurrent network and cellular bistability. *Journal of Computational Neuroscience*, *5*(4), 383–405.

De Vries, B., & Principe, J. (1992). Gamma model—A new neural model for temporal processing. *Neural Networks*, *5*(4), 565–576.

Feldkamp, L., Prokhorov, D., & Feldkamp, T. (2003). Simple and conditioned adaptive behavior from Kalman filter trained recurrent networks? *Neural Networks*, *16*(5–6), 683–689.

Gers, F., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, *12*(10), 2451–2471.

Gers, F., Schraudolph, N., & Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, *3*, 115–143.

Hinton, G., & Brown, A. (2000). Spiking Boltzmann machines. In *Advances in neural information processing systems 12: Proceedings of the 1999 conference*.

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Institut fur Informatik*, Technische Universitat, Munchen.

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

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, *304*(5667), 78–80.

Lin, T., Horne, B., Tino, P., & Giles, C. (2000). Learning long term dependencies in narx recurrent neural networks. *Recurrent Neural Networks: Design and Applications*, 133.

Mayer, H., Gomez, F., Wierstra, D., Nagy, I., Knoll, A., & Schmidhuber, J. (2006). A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *Intelligent robots and systems, 2006 IEEE/RSJ international conference on* (pp. 543–548).

Natarajan, R., Huys, Q., Dayan, P., & Zemel, R. (2008). Encoding and decoding spikes for dynamic stimuli. *Neural Computation*, 1–36.

Prokhorov, D., Feldkamp, L., & Tyukin, I. (2002). Adaptive behavior with fixed weights in RNN: An overview. In *Proceedings of international joint conference on neural networks*, (pp. 2018–2023) *Vol. 2*.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Seung, H. (1996). How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences*, *93*(23), 13339.

Shepherd, G. (1998). *The synaptic organization of the brain*. New York: Oxford University Press.

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

White, O., Lee, D., & Sompolinsky, H. (2004). Short-term memory in orthogonal neural networks. *Physical Review Letters*, *92*(14), 148102.

Williams, R. (1992). Training recurrent networks using the extended Kalman filter. In *Neural networks, 1, 1992. IJCNN. International joint conference on, Vol. 4.*

Younger, A., Hochreiter, S., & Conwell, P. (2001). Meta-learning with backpropagation. In *Neural networks,, 2001. Proceedings. IJCNN'01. International joint conference on, Vol. 3.*

[4] All experiments used an LSTM with 50 memory blocks consisting of 2 memory cells and a TKRNN$^+$5 with 30 hidden units. The online code contains the details of these experiments.