A Clockwork RNN

Jan Koutník HKOU@IDSIA.CH
Klaus Greff KLAUS@IDSIA.CH
Faustino Gomez TINO@IDSIA.CH
Jürgen Schmidhuber JUERGEN@IDSIA.CH
IDSIA, USI&SUPSI, Manno-Lugano, CH-6928, Switzerland

Abstract

Sequence prediction and classification are ubiquitous and challenging problems in machine learning that can require identifying complex dependencies between temporally distant inputs. Recurrent Neural Networks (RNNs) have the ability, in theory, to cope with these temporal dependencies by virtue of the short-term memory implemented by their recurrent (feedback) connections. However, in practice they are difficult to train successfully when the long-term memory is required. This paper introduces a simple, yet powerful modification to the standard RNN architecture, the *Clockwork RNN* (CW-RNN), in which the hidden layer is partitioned into separate modules, each processing inputs at its own temporal granularity, making computations only at its prescribed clock rate. Rather than making the standard RNN models more complex, CW-RNN reduces the number of RNN parameters, improves the performance significantly in the tasks tested, and speeds up the network evaluation. The network is demonstrated in preliminary experiments involving two tasks: audio signal generation and TIMIT spoken word classification, where it outperforms both RNN and LSTM networks.

1. Introduction

Recurrent Neural Networks (RNNs; Robinson & Fallside, 1987; Werbos, 1988; Williams, 1989) are a class of connectionist models that possess internal state or *short term memory* due to recurrent feed-back connections, that make them suitable for dealing with sequential problems, such as speech classification, prediction and generation.

Standard RNNs trained with stochastic gradient descent have difficulty learning long-term dependencies (i.e. spanning more that 10 time-steps) encoded in the input sequences due to the *vanishing gradient* (Hochreiter, 1991; Hochreiter et al., 2001). The problem has been addressed for example by using a specialized neuron structure, or cell, in Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) that maintains constant backward flow in the error signal; second-order optimization methods (Martens & Sutskever, 2011) preserve the gradient by estimating its curvature; or using informed random initialization (Sutskever et al., 2013) which allows for training the networks with momentum and stochastic gradient descent only.

This paper presents a novel modification to the simple RNN (SRN; Elman, 1988) architecture and, *mutatis mutandis*, an associated error back-propagation through time (Rumelhart et al., 1986; Werbos, 1988; Williams, 1989) training algorithm, that show superior performance in the generation and classification of sequences that contain long-term dependencies. Here, the long-term dependency problem is solved by having different parts (modules) of the RNN hidden layer running at different clock speeds, timing their computation with different, discrete clock periods, hence the name Clockwork Recurrent Neural Network (CW-RNN). CW-RNN train and evaluate faster since not all modules are executed at every time step, and have a smaller number of weights compared to SRNs, because slower modules are not connected to faster ones.

CW-RNNs were tested on two supervised learning tasks: sequence generation where a target audio signal must be output by a network using no input; and spoken word classification using the TIMIT dataset. In these preliminary experiments, CW-RNN outperformed both SRN and LSTM with the same number of weights by a significant margin. The next section provides an overview of the related work, section 3 describes the CW-RNN architecture in detail and section 5 discusses the results of experiments in section 4 and future potential of Clockwork Recurrent Neural Networks.

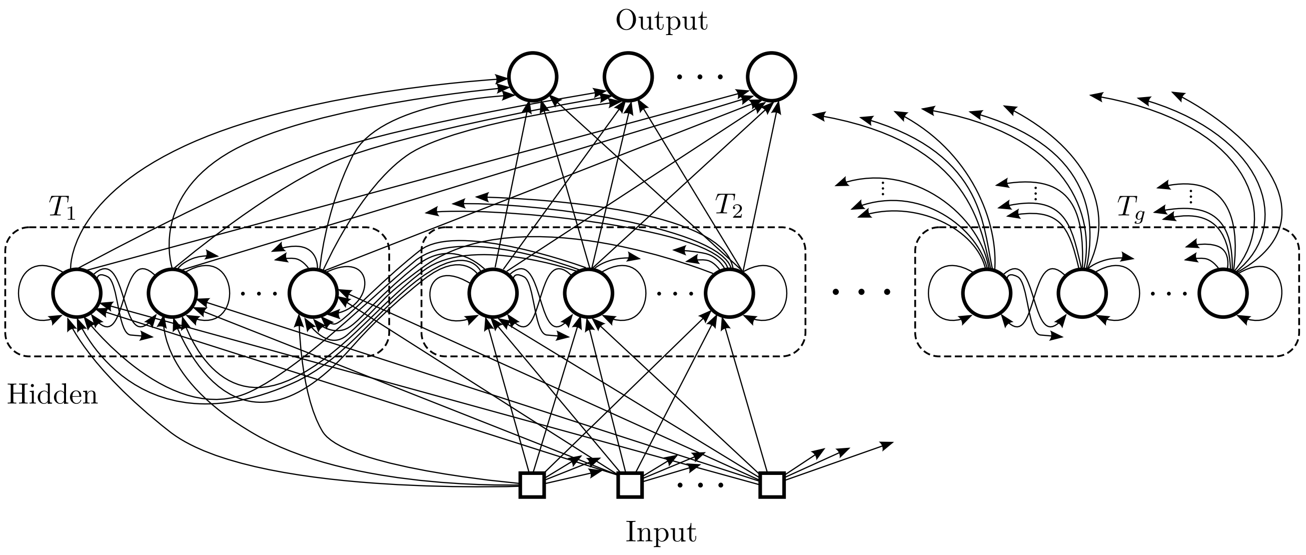

Figure 1. CW-RNN architecture is similar to a simple RNN with an input, output and hidden layer. The hidden layer is partitioned into g modules each with its own clock rate. Within each module the neurons are fully interconnected. Neurons in faster module i are connected to neurons in a slower module j only if a clock period $T_i < T_j$.

2. Related Work

Contributions to the sequence modeling and recognition that are relevant to CW-RNN are introduced in this section. The primary focus is on RNN extensions that deal with the problem of bridging long time lags.

One model that is similar in spirit to our approach is the NARX RNN[1] (Lin et al., 1996). But instead of simplifying the network, it introduces an additional sets of recurrent connections with time lags of 2,3..k time steps. These additional connections help to bridge long time lags, but introduce many additional parameters that make NARX RNN training more difficult and run k times slower.

Long Short-Term Memory (LSTM; Hochreiter & Schmidhuber, 1997) uses a specialized architecture that allows information to be stored in a linear unit called a *constant error carousel* (CEC) indefinitely. The cell containing the CEC has a set of multiplicative units (gates) connected to other cells that regulate when new information enters the CEC (input gate), when the activation of the CEC is output to the rest of the network (output gate), and when the activation decays or is ”forgotten” (forget gate). These networks have been very successful recently in speech and handwriting recognition (Graves et al., 2005; 2009; Sak et al., 2014).

Stacking LSTMs into several layers (Fernandez et al., 2007; Graves & Schmidhuber, 2009) aims for hierarchical sequence processing. Such a hierarchy, equipped with Connectionist Temporal Classification (CTC; Graves et al., 2006), performs simultaneous segmentation and recognition of sequences. Its deep variant currently holds the state-of-the-art result in phoneme recognition on the TIMIT database (Graves et al., 2013).

Temporal Transition Hierarchy (TTH; Ring, 1993) incrementally adds high-order neurons in order to build a memory that is used to disambiguate an input at the current time step. This approach can, in principle, bridge time intervals of any length, but with proportionally growing network size. The model was recently improved by adding recurrent connections (Ring, 2011) that prevent it from bloating by reusing the high-level nodes through the recurrent connections.

One of the earliest attempts to enable RNNs to handle long-term dependencies is the Reduced Description Network (Mozer, 1992; 1994). It uses leaky neurons whose activation changes only a bit in response to its inputs. This technique was recently picked up by Echo State Networks (ESN; Jaeger, 2002).

A similar technique has been used by Sutskever & Hinton (2010) to solve some serial recall tasks. These Temporal-Kernel RNNs add a connection from each neuron to itself that has a weight that decays exponentially in time. This is implemented in a way that can be computed efficiently, however, its performance is still inferior to LSTM.

Evolino (Schmidhuber et al., 2005; 2007) feeds the input to an RNN (which can be e.g. LSTM to cope with long time lags) and then transforms the RNN outputs to the target sequences via a optimal linear mapping, that is computed

[1]NARX stands for Non-linear Auto-Regressive model with eXogeneous inputs

analytically by pseudo-inverse. The RNN is trained by an evolutionary algorithm, therefore it does not suffer from the vanishing gradient problem. Evolino outperformed LSTM on a set of synthetic problems and was used to perform complex robotic manipulation (Mayer et al., 2006).

A modern theory of why RNNs fail to learn long-term dependencies is that simple gradient descent fails to optimize them correctly. One attempt to mitigate this problem is Hessian Free (HF) optimization (Martens & Sutskever, 2011), an adapted second-order training method that has been demonstrated to work well with RNNs. It allows RNNs to solve some long-term lag problems that were impossible with stochastic gradient descent. Their performance on rather synthetic, long-term memory benchmarks is approaching the performance of LSTM, though the number of optimization steps in HF-RNN is usually greater. Training networks by HF optimization is an orthogonal approach to the network architecture, so both LSTM and CW-RNN can still benefit from it.

HF optimization allowed for training of Multiplicative RNN (MRNN; Sutskever et al., 2011) that port the concept of multiplicative gating units to SRNs. The gating units are represented by a factored 3-way tensor in order to reduce the number of parameters. Extensive training of an MRNN for a number of days on a graphics cards provided impressive results in text generation tasks.

Training RNNs with Kalman filters (Williams, 1992) has shown advantages in bridging long time lags as well, although this approach is computationally unfeasible for larger networks.

The methods mentioned above are strictly synchronous–elements of the network clock at the same speed. The *Sequence Chunker*, Neural History Compressor or Hierarchical Temporal Memory (Schmidhuber, 1991; 1992) consists of a hierarchy or stack of RNN that may run at different time scales, but, unlike the simpler CW-RNN, it requires unsupervised event predictors: a higher-level RNN receives an input only when the lower-level RNN below is unable to predict it. Hence the clock of the higher level may speed up or slow down, depending on the current predictability of the input stream. This contrasts the CW-RNN, in which the clocks always run at the same speed, some slower, some faster.

3. A Clockwork Recurrent Neural Network

Clockwork Recurrent Neural Networks (CW-RNN) like SRNs, consist of input, hidden and output layers. There are forward connections from the input to hidden layer, and from the hidden to output layer, but, unlike the SRN,

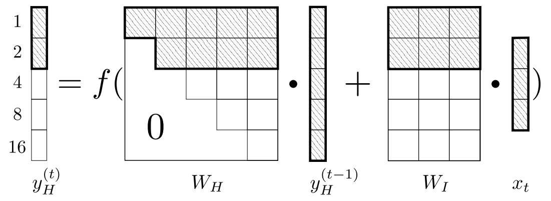

Figure 2. Calculation of the hidden unit activations at time step $t = 6$ in CW-RNN according to equation (1). Input and recurrent weight matrices are partitioned into blocks. Each block-row in $\mathbf{W}_H$ and $\mathbf{W}_I$ corresponds to the weights of a particular module. At time step $t = 6$, the first two modules with periods $T_1 = 1$ and $T_2 = 2$ get evaluated (highlighted parts of $\mathbf{W}_H$ and $\mathbf{W}_I$ are used) and the highlighted outputs are updated. Note that, while using exponential series of periods, the active parts of $\mathbf{W}_H$ and $\mathbf{W}_I$ are always contiguous.

the neurons in the hidden layer are partitioned into g modules of size k. Each of the modules is assigned a clock period $T_n \in \{T_1, \ldots, T_g\}$. Each module is internally fully-interconnected, but the recurrent connections from module j to module i exists only if the period T_i is smaller than period T_j. Sorting the modules by increasing period, the connections between modules propagate the hidden state *right-to-left*, from slower modules to faster modules, see Figure 1.

The standard RNN output, $y_O^{(t)}$, at a time step t is calculated using the following equations:

$$\mathbf{y}_H^{(t)} = f_H(\mathbf{W}_H \cdot \mathbf{y}^{(t-1)} + \mathbf{W}_I \cdot \mathbf{x}^{(t)}), \tag{1}$$

$$\mathbf{y}_O^{(t)} = f_O(\mathbf{W}_O \cdot \mathbf{y}_H^{(t)}), \tag{2}$$

where $\mathbf{W}_H$, $\mathbf{W}_I$ and $\mathbf{W}_O$ are the hidden, input and output weight matrices, $\mathbf{x}_t$ is the input vector at time step t, vectors $\mathbf{y}_H^{(t)}$ and $\mathbf{y}_H^{(t-1)}$ represent the hidden neuron activations at time steps t and $t-1$. Functions $f_H(.)$ and $f_O(.)$ are the non-linear activation functions. For simplicity, neuron biases are omitted in the equations.

The main difference between CW-RNN and an RNN is that at each CW-RNN time step t, only the output of modules i that satisfy $(t \text{ MOD } T_i) = 0$ are executed. The choice of the set of periods $\{T_1, \ldots, T_g\}$ is arbitrary. In this paper, we use the exponential series of periods: module i has clock period of $T_i = 2^{i-1}$.

Matrices $\mathbf{W}_H$ and $\mathbf{W}_I$ are partitioned into g blocks-rows:

$$\mathbf{W}_H = \begin{pmatrix} \mathbf{W}_{H_1} \\ \vdots \\ \mathbf{W}_{H_g} \end{pmatrix} \qquad \mathbf{W}_I = \begin{pmatrix} \mathbf{W}_{I_1} \\ \vdots \\ \mathbf{W}_{I_g} \end{pmatrix} \tag{3}$$

and $\mathbf{W}_H$ is a block-upper triangular matrix, where each block-row, $\mathbf{W}_{H_i}$, is partitioned into block-columns

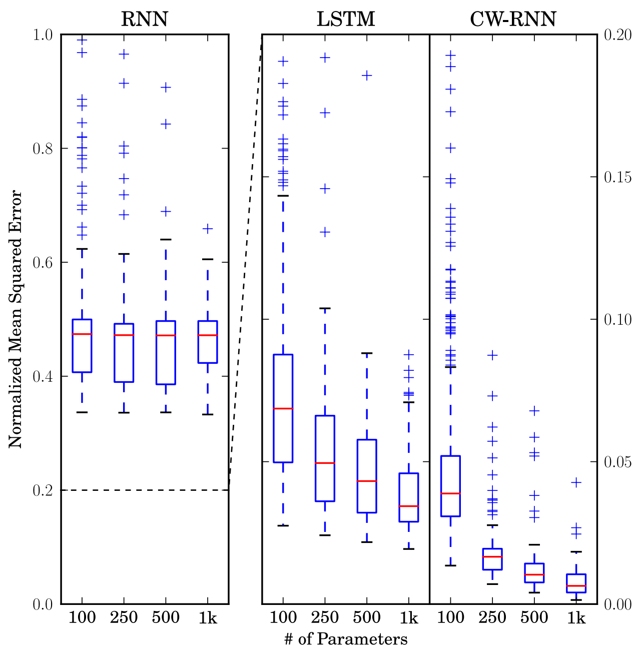

Figure 3. The normalized mean squared error for the sequence generation task, divided into one column per method, with one box-whisker (showing mean value, 25% and 75% quantiles, minimum, maximum and outliers) for every tested size of the network. Note that the plot for the standard RNN has a different scale than the other two.

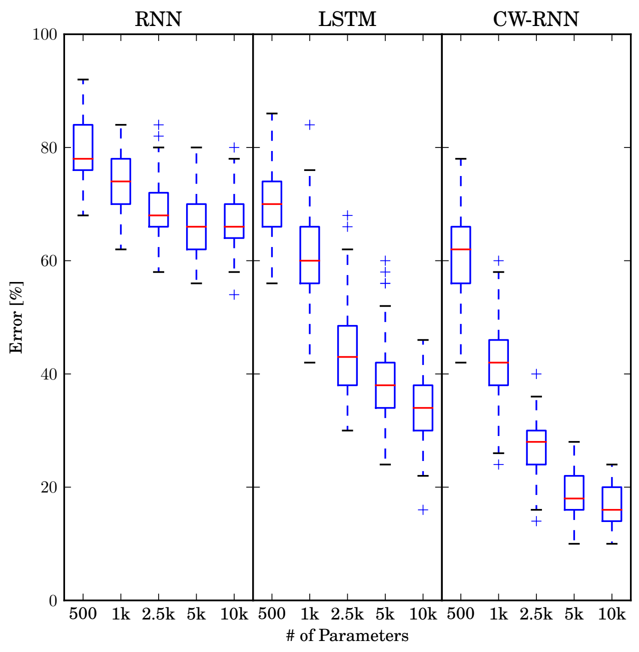

Figure 4. The classification error for the word classification task, divided into three columns (one per method), with one box-whisker for every tested network size.

$\{\mathbf{0}_1, \ldots, \mathbf{0}_{i-1}, \mathbf{W}_{\mathbf{H}_{\mathbf{i},\mathbf{i}}}, \ldots, \mathbf{W}_{\mathbf{H}_{\mathbf{i},\mathbf{g}}}\}$. At each forward pass time step, only the block-rows of $\mathbf{W}_H$ and $\mathbf{W}_I$ that correspond to the executed modules are used for evaluation in Equation (1):

$$\mathbf{W}_{H_i} = \begin{cases} \mathbf{W}_{H_i} & \text{for } (t \text{ MOD } T_i) = 0 \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{4}$$

and the corresponding parts of the output vector, $\mathbf{y}_H$, are updated. The other modules retain their output values from the previous time-step. Calculation of the hidden activation at time step $t = 6$ is illustrated in Figure 2.

As a result, the low-clock-rate modules process, retain and output the long-term information obtained from the input sequences (not being distracted by the high speed modules), whereas the high-speed modules focus on the local, high-frequency information (having the context provided by the low speed modules available).

The backward pass of the error propagation is similar to RNN as well. The only difference is that the error propagates only from modules that were executed at time step t. The error of non-activated modules gets copied back in time (similarly to copying the activations of nodes not activated at the time step t during the corresponding forward pass), where it is added to the back-propagated error.

CW-RNN runs much faster than a simple RNN with the same number of hidden nodes since not all modules are evaluated at every time step. The lower bound for the CW-RNN speedup compared to an RNN with the same number of neurons is $g/4$ in the case of this exponential clock setup, see Appendix for a detailed derivation.

4. Experiments

CW-RNNs were compared to the simple RNN (SRN) and LSTM networks. All networks have one hidden layer with the *tanh* activation function, and the number of nodes in the hidden layer was chosen to obtain (approximately) the same number of parameters for all three methods (in the case of CW-RNN, the clock periods were included in the parameter count).

Initial values for all the weights were drawn from a Gaussian distribution with zero mean and standard deviation of 0.1. Initial values of all internal state variables were set to 0. Each setup was run 100 times with different random initialization of parameters. All networks were trained using Stochastic Gradient Descent (SGD) with Nesterov-style momentum (Sutskever et al., 2013).

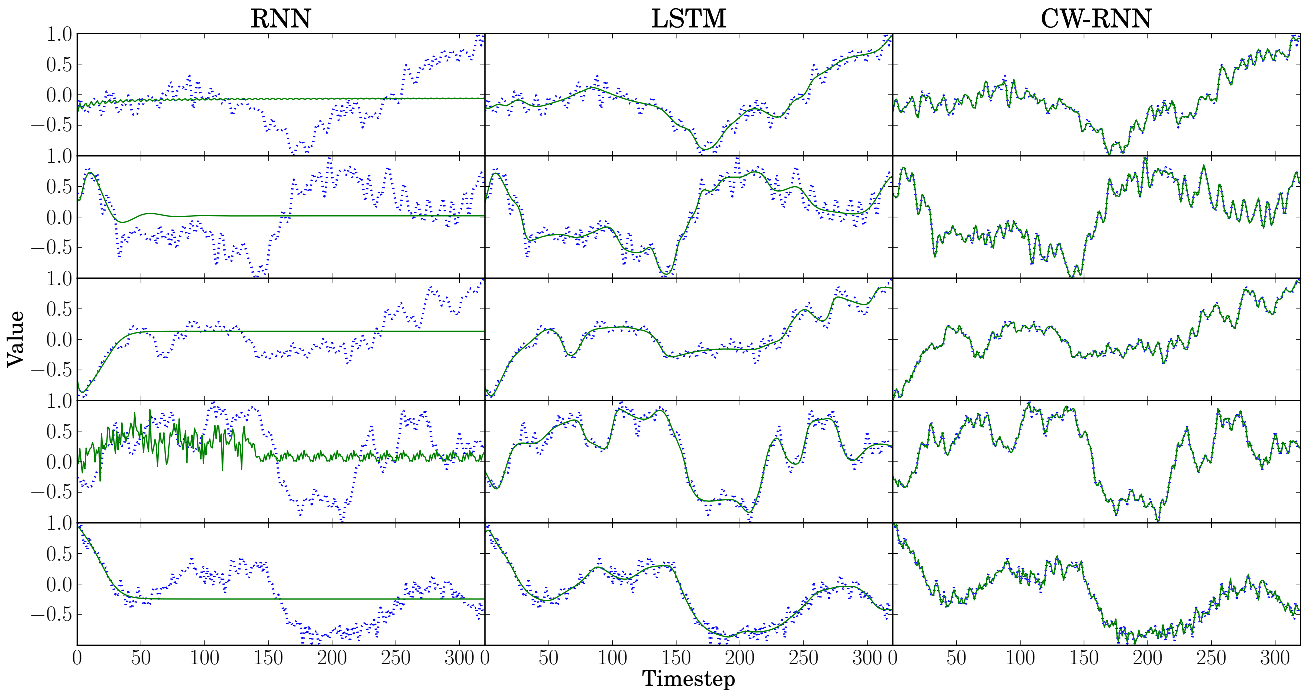

Figure 5. Output of the best performing network (solid, green) compared to the target signal (dotted, blue) for each method (column) and each training sequence (row). RNN tends to learn the first few steps of the sequence and then generates the mean of the remaining portion, while the output of LSTM resembles a sliding average, and CW-RNN approximates the sequence much more accurately.

4.1. Sequence Generation

The goal of this task is to train a recurrent neural network, that receives no input, to generate a target sequence as accurately as possible. The weights of the network can be seen as a (lossy) encoding of the whole sequence, which could be used for compression.

Five different target sequences were created by sampling a piece of music[2] at 44.1 Hz for 7 ms. The resulting sequences of 320 data points each were scaled to the interval $[-1, 1]$. In the following experiments we compare performance on these five sequences.

All networks used the same architecture: no inputs, one hidden layer and a single linear output neuron. Each network type was run with 4 different sizes: 100, 250, 500, and 1000 parameters, see Table 1 for the summary of number of hidden nodes. The networks were trained over 2000 epochs to minimize the mean squared error. After that time the error no longer decreased noticeably. Momentum was set to 0.95 while the learning rate was optimized separately for every method, but kept the same for all network sizes.

A learning rate of 3×10^{-4} was found to be optimal for RNN and CW-RNN while for LSTM 3×10^{-5} gave better results. For LSTM it was also crucial to initialize the bias of the forget gates to a high value (5 in this case) to encourage the long-term memory. The hidden units of CW-RNN were divided into nine equally sized groups with exponential clock-timings $\{1, 2, 4, \ldots, 256\}$.

[2]taken from the beginning of the first track *Manýrista* of album *Musica Deposita* by *Cuprum*

The results for the experiments are shown in Figure 3. It is obvious that RNNs fail to generate the target sequence, and they do not seem to improve with network size. LSTM does much better, and shows an improvement as the networks get bigger. CW-RNNs give by far the best results, with the smallest one being roughly on par with the second-biggest LSTM network. Also, all but the smallest CW-RNN have significantly less variance than all the other methods. To get an intuitive understanding of what is happening, Figure 5 shows the output of the best network of each type on each one of the five audio samples. The average error of the best networks is summarized in Table 3 (row 1).

4.2. Spoken Word Classification

The second task is sequence classification instead of generation. Each sequence contains an audio signal of one spoken word from the TIMIT Speech Recognition Benchmark (Garofolo et al., 1993). The dataset contains 25 dif-

Table 1. Number of hidden neurons (cells in the case of LSTM) for RNN, LSTM and CW-RNN for each network size specified in terms of the number of parameters (weights) for the sequence generation task.

# of Parameters	RNN	LSTM	CW-RNN
100	9	4	11
250	15	7	19
500	22	10	27
1 000	31	15	40

ferent words (classes) arranged in 5 clusters based on their suffix. Because of the suffix-similarity the network needs to learn long-term dependencies in order to disambiguate the words. The words are:

Cluster 1: `making, walking, cooking, looking, working`

Cluster 2: `biblical, cyclical, technical, classical, critical`

Cluster 3: `tradition, addition, audition, recognition, competition`

Cluster 4: `musicians, discussions, regulations, accusations, conditions`

Cluster 5: `subway, leeway, freeway, highway, hallway`

For every word there are 7 examples from different speakers, which were partitioned into 5 for training and 2 for testing, for a total of 175 sequences (125 train, 50 test). Each sequence element consists of 12-dimensional MFCC vector (Mermelstein, 1976) plus energy, sampled every 10 ms over a 25 ms window with a pre-emphasis coefficient of 0.97. Each of the 13 channels was then normalized to have zero mean and unit variance over the whole training set.

All network types used the same architecture: 13 inputs, a single hidden and a softmax output layer with 25 units. Five hidden layer sizes were chosen such that the total number of parameters for the whole network is roughly 0.5k, 1k, 2.5k, 5k, and 10k.

All networks used a learning rate of 3×10^{-4}, a momentum of 0.9, and were trained to minimize the Multinomial Cross Entropy Error. Every experiment was repeated 100 times with different random initializations.

Because the dataset is so small, Gaussian noise with a standard deviation of 0.6 was added to the inputs during training to guard against overfitting. Training was stopped once the error on the *noise-free* training set did not decrease for 5 epochs. To obtain good results with LSTM, it was again important to initialize the forget gate bias to 5. For the CW-RNN the neurons were divided evenly into 7 groups with exponentially increasing periods: $\{1, 2, 4, 8, 16, 32, 64\}$.

Table 2. Number of hidden neurons (cells in the case of LSTM) for RNN, LSTM and CW-RNN for each network size specified in terms of the number of parameters (weights) for the spoken word classification task.

# of Parameters	RNN	LSTM	CW-RNN
500	10	5	10
1000	18	8	19
2500	34	17	40
5000	54	26	65
10000	84	41	102

Table 3. Mean error and standard deviation (averaged over 100 runs) for the largest (best) LSTM, RNN and CW-RNN on both tasks. CW-RNN is 5.7× better than LSTM on Task 4.1, sequence generation, and more than 2× better than LSTM on Task 4.2, spoken word classification.

Task		RNN	LSTM	CW-RNN
4.1	NMSE	0.46±0.08	0.04±0.01	0.007±0.004
4.2	Error [%]	66.8±4.7	34.2±5.6	16.8±3.5

Figure 4 shows the classification error of the different networks on the word classification task. Here again, RNNs perform the worst, followed by LSTMs, which give substantially better results, especially with more parameters. CW-RNNs beat both RNN and LSTM networks by a considerable margin of 8-20% on average irrespective of the number of parameters. The error of the largest networks is summarized in Table 3 (row 2).

5. Discussion

The experimental results show that the simple mechanism of running subsets of neurons at different speeds allows an RNN to efficiently learn the different dynamic time-scales inherent in complex signals.

Other functions could be used to set the module periods: linear, Fibonacci, logarithmic series, or even fixed random periods. These were not considered in this paper because the intuitive setup of using an exponential series worked well in these preliminary experiments. Another option would be to learn the periods as well, which, to use error back-propagation would require a differentiable modulo function for triggering the clocks. Alternatively, one could train the clocks (together with the weights) using evolutionary algorithms which do not require a closed form for the gradient.

Note that the lowest period in the network can be greater than 1. Such a network would not be able to change its output at every time step, which may be useful as a low-pass filter when the data contains noise.

Also, the modules do not have to be all of the same size. One could adjust them according to the expected information in the input sequences, by e.g. using frequency analysis of the data and setting up modules sizes and clocks proportional to the spectrum.

Grouping hidden neurons into modules is a partway to having each weight have its own clock. Initial experiments, not included in this paper, have shown that such networks are hard to train and do not provide good results.

CW-RNN showed superior performance on the speech data classification among all three models tested. Note that, unlike in the standard approach, in which the speech signal frequency coefficients are first translated to phonemes which are modeled with a standard approach like Hidden Markov Modes for complete words, CW-RNN attempts to model and recognize the complete words directly, where it benefits from the modules running at multiple speeds.

Future work will start by conducting a detailed analysis of the internal dynamics taking place in the CW-RNN to understand how the network is allocating resources for a given type of input sequence. Further testing on other classes of problems, such as reinforcement learning, and comparison to the larger set of connectionist models for sequential data processing are also planned.

Appendix

CW-RNN has fewer total parameters and even fewer operations per time step than a standard RNN with the same number of neurons. Assume CW-RNN consists of g modules of size k for a total of $n = kg$ neurons. Because a neuron is only connected to other neurons with the same or larger period, the number of parameters N_H for the recurrent matrix is:

$$N_H = \sum_{i=1}^{g}\sum_{j=1}^{k} k(g-i+1) = k^2 \sum_{i=0}^{g-1}(g-i) = \frac{n^2}{2} + \frac{nk}{2}.$$

Compared to the n^2 parameters in the recurrent matrix $\mathbf{W}_H$ of RNN this results in roughly half as many parameters:

$$\frac{N_H}{n^2} = \frac{\frac{n^2}{2}+\frac{nk}{2}}{n^2} = \frac{n^2+nk}{2n^2} = \frac{n+k}{2n} = \frac{g+1}{2g} \approx \frac{1}{2}.$$

Each module i is evaluated only every T_i-th time step, therefore the number of operations at a time step is:

$$O_H = k^2 \sum_{i=0}^{g-1} \frac{g-i}{T_i}.$$

For exponentially scaled periods, $T_i = 2^i$, the upper bound for number of operations, O_H, needed for $\mathbf{W}_H$ per time step is:

$$O_h = k^2 \sum_{i=0}^{g-1} \frac{g-i}{2^i} = k^2 \left(g \underbrace{\sum_{i=0}^{g-1} \frac{1}{2^i}}_{\leq 2} + \underbrace{\sum_{i=0}^{g-1} \frac{i}{2^i}}_{\leq 2} \right) \leq$$
$$\leq k^2(2g-2) \leq 2nk,$$

because $g \geq 2$ this is less than or equal to n^2. Recurrent operations in CW-RNN are faster than in an RNN with the same number of neurons by a factor of at least $g/2$, which, for typical CW-RNN sizes ends up being between 2 and 5. Similarly, upper bound for the number of input weight evaluations, E_I, is:

$$O_I \quad = \quad \sum_{i=0}^{g-1} \frac{km}{T_i} \quad = \quad km \sum_{i=0}^{g-1} \frac{1}{T_i} \quad \leq \quad 2km$$

Therefore, the overall CW-RNN speed-up w.r.t RNN is:

$$\frac{n^2+nm+n}{O_R+O_I+2n} = \frac{k^2g^2+kgm+kg}{k^2(2g-2)+2km+2kg} =$$
$$= \frac{g(kg+m+1)}{2(k(g-1)+m+g)} = \frac{g}{2} \underbrace{\frac{(kg+m+1)}{k(g-1)+m+g}}_{\geq \frac{1}{2}} \geq \frac{g}{4}$$

Note that this is a conservative lower bound.

Acknowledgments

This research was supported by Swiss National Science Foundation grant #138219: "Theory and Practice of Reinforcement Learning 2", and the EU FP7 project "NanoBioTouch", grant #228844.

References

Elman, J. L. Finding structure in time. CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

Fernandez, Santiago, Graves, Alex, and Schmidhuber, Jürgen. Sequence labelling in structured domains with hierarchical recurrent neural networks. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., and Dahlgren, N. L. DARPA TIMIT acoustic phonetic continuous speech corpus CD-ROM, 1993.

Graves, A., Beringer, N., and Schmidhuber, J. Rapid retraining on speech data with LSTM recurrent networks. Technical Report IDSIA-09-05, IDSIA, 2005. URL `http://www.idsia.ch/idsiareport/IDSIA-09-05.pdf`.

Graves, A., Fernandez, S., Gomez, F. J., and Schmidhuber, J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural nets. In *ICML'06: Proceedings of the International Conference on Machine Learning*, 2006.

Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., and Schmidhuber, J. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), 2009.

Graves, Alex and Schmidhuber, Jürgen. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems 21*, pp. 545–552. MIT Press, Cambridge, MA, 2009.

Graves, Alex, rahman Mohamed, Abdel, and Hinton, Geoffrey E. Speech recognition with deep recurrent neural networks. In *ICASSP*, pp. 6645–6649. IEEE, 2013.

Hochreiter, S. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991. Advisor: J. Schmidhuber.

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

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer, S. C. and Kolen, J. F. (eds.), *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

Jaeger, H. Short term memory in echo state networks. GMD-Report 152, GMD - German National Research Institute for Computer Science, 2002. URL `http://www.faculty.jacobs-university.de/hjaeger/pubs/STMEchoStatesTechRep.pdf`.

Lin, T., Horne, B.G., Tino, P., and Giles, C.L. Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

Martens, J. and Sutskever, I. Learning recurrent neural networks with Hessian-free optimization. In Getoor, L. and Scheffer, T. (eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Mayer, Hermann, Gomez, Faustino J., Wierstra, Daan, Nagy, Istvan, Knoll, Alois, and Schmidhuber, Juergen. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *Proceedings of the International Conference on Intelligent Robotics and Systems (IROS-06, Beijing)*, 2006.

Mermelstein, P. Distance measures for speech recognition: Psychological and instrumental. In Chen, C. H. (ed.), *Pattern Recognition and Artificial Intelligence*, pp. 374–388. Academic Press, New York, 1976.

Mozer, M. C. Induction of multiscale temporal structure. In Lippman, D. S., Moody, J. E., and Touretzky, D. S. (eds.), *Advances in Neural Information Processing Systems 4*, pp. 275–282. Morgan Kaufmann, 1992.

Mozer, M. C. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3): 247–280, 1994.

Ring, Mark. Learning sequential tasks by incrementally adding higher orders. In S. J. Hanson, J. D. Cowan and Giles, C. L. (eds.), *Advances in Neural Information Processing Systems 5*, pp. 115–122. Morgan Kaufmann, 1993.

Ring, Mark. Recurrent transition hierarchies for continual learning: A general overview. In *Lifelong Learning*, volume WS-11-15 of *AAAI Workshops*. AAAI, 2011.

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

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L. (eds.), *Parallel Distributed Processing*, volume 1, pp. 318–362. MIT Press, 1986.

Sak, Haşim, Senior, Andrew, and Beaufays, Françoise. Long Short-Term Memory based recurrent neural network architectures for large vocabulary speech recognition. Technical report, Google, February 2014. arXiv:1402.1128 [cs.NE].

Schmidhuber, J. Neural sequence chunkers. Technical Report FKI-148-91, Institut für Informatik, Technische Universität München, April 1991.

Schmidhuber, J. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

Schmidhuber, J., Wierstra, D., and Gomez, F. J. Evolino: Hybrid neuroevolution / optimal linear search for sequence prediction. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 853–858, 2005.

Schmidhuber, J., Wierstra, D., Gagliolo, M., and Gomez, F. J. Training recurrent networks by Evolino. *Neural Computation*, 19(3):757–779, 2007.

Sutskever, I., Martens, J., and Hinton, G. Generating text with recurrent neural networks. In Getoor, L. and Scheffer, T. (eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Sutskever, Ilya and Hinton, Geoffrey. Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243, March 2010.

Sutskever, Ilya, Martens, James, Dahl, George E., and Hinton, Geoffrey E. On the importance of initialization and momentum in deep learning. In Dasgupta, Sanjoy and Mcallester, David (eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pp. 1139–1147. JMLR Workshop and Conference Proceedings, May 2013. URL http://jmlr.org/proceedings/papers/v28/sutskever13.pdf.

Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1, 1988.

Williams, R. J. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

Williams, Ronald J. Training recurrent networks using the extended kalman filter. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 4, pp. 241–246. IEEE, 1992.