MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Time Series Prediction Using Neural Networks

BACHELOR THESIS

Karol Kuna

Brno, Spring 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Karol Kuna

Advisor: doc. RNDr. Tomáš Brázdil, Ph.D.

Acknowledgement

I would like to thank my advisor for the guidance he has provided.

Abstract

This thesis compares existing methods for predicting time series in real time using neural networks. Focus is put on recurrent neural networks (RNNs) and online learning algorithms, such as Real-Time Recurrent Learning and truncated Backpropagation Through Time. In addition to the standard Elman's RNN architecture, Clockwork-RNN is examined. Methods are compared in terms of prediction accuracy and computational time, which is critical in real-time applications. Part of the work is experimental implementation of the tested models and working applications in robotics and network traffic monitoring.

Keywords

time series, prediction, neural network, recurrent network, backpropagation, backpropagation through time, real-time recurrent learning, clockwork recurrent network

Contents

1 **Introduction** . . . 1
2 **Prediction** . . . 3
2.1 *Time Series* . . . 3
2.2 *Prediction* . . . 3
2.3 *Prediction Horizon* . . . 4
2.4 *Prediction Chain* . . . 4
3 **Artificial Neural Networks** . . . 5
3.1 *Artificial Neuron* . . . 5
3.2 *Feedforward Neural Networks* . . . 6
3.2.1 Backpropagation . . . 6
3.3 *Recurrent Neural Networks* . . . 8
3.3.1 Elman's Simple Recurrent Network . . . 9
3.3.2 Backpropagation Through Time . . . 10
3.3.3 Truncated Backpropagation Through Time . . . 11
3.3.4 Real-Time Recurrent Learning . . . 12
3.4 *Clockwork Recurrent Network* . . . 13
4 **Implementation** . . . 15
4.1 *Neural Prediction Framework* . . . 15
4.1.1 Memory Block . . . 15
4.1.2 Neural Layer . . . 15
4.1.3 Neural Network . . . 15
4.2 *Network Monitor* . . . 15
4.3 *Robotic Simulator* . . . 15
5 **Experiments** . . . 16
5.1 *Sine Wave* . . . 16
5.1.1 Role of Learning and Momentum Rate . . . 16
5.1.2 Impact of Network Size on Performance . . . 16
5.2 *Network Usage* . . . 16
5.3 *Robotic Manipulator* . . . 16
6 **Results** . . . 17
6.1 *Trade-offs* . . . 17
7 **Conclusion** . . . 18
A **Appendix** . . . 19

List of Tables

List of Figures

3.1 On the left, MLP consisting of input layer with two units, two hidden layers with four and three units respectively, and output layer with two untis. Schematic diagram of the MLP's layers on the right. *7*
3.2 On the left, Elman's recurrent network with input, hidden, context, and output layer, each containing two units. On the right, schema of layers in Elman network. Dotted link signifies copying activity of source's units to target units. *9*
3.3 Elman's recurrent network unfolded in time. *11*
3.4 Clockwork-RNN. *13*

1 Introduction

Each year more and more data is collected from high velocity data streams. Machine learning algorithms are used to analyse and create models of the data. Patterns found in the data structure are then exploited to make predictions about the future which can be used to guide decision making. In this thesis, I compare various artifical neural network (ANN) architectures and learning algorithms for online prediction of time series.

One of the most popular machine learning algorithms are ANNs which are capable of approximating unknown functions and thus are good candidates for use in prediction. Inspired by biological neural networks, ANNs consist of artificial neurons wired together to form a network.

Common architectures, such as feedforward networks (FFN) and simple recurrent networks (SRN) trained by backpropagation through time (BPTT) and Real-Time Recurrent Learning (RTRL), as well as latest Clockwork-RNN are evaluated in this thesis. Algorithms are compared in terms of prediction accuracy as well as computational time and the trade-off between them.

Time series are sequences of data points in time, usually created by measuring output of some process in discrete time intervals. The goal of prediction is to successfully estimate output of the process in next time step or several steps. It is assumed that the process is at least partially observable and to some extent, future values can be determined by observing past values. Prediction then reduces to problem of process approximation.

Since value of stored data decreases over time, effort is put into gaining insight from data as they come. Many applications don't require storing data at all, or storing it would be impractical, therefore it is advantageous to create models and update them with every data point. Continuous online learning is well suited for such tasks and can adapt to changing environments without human intervention.

Quality and speed of prediction is tested in two application scenarios. The first is robotic simulator implemented in 3D physics library BEPUphysics. Manipulator with five actuated joints is created, the goal is to predict future state of the robot's body. The prediction

network should act as a virtual model of the body, which is useful in control systems, e.g. internal model control.

The second scenario is monitoring utilisation of computer resources, namely processor, memory, disk, and network usage. Network should constantly predict future values of these variables and detect anomalies. Should an unexpected event occur, it needs to be logged for an administrator to examine.

2 Prediction

2.1 Time Series

Time series are sequences of data points measured over time. In this thesis, data points are real-valued vectors implemented as arrays of floating point numbers. Data sequence is created by measuring output of a process at discrete, regular time intervals. Process may also receive input in every time step, which affects its future behaviour, or it may be purely generative receiving no input at all. For generalisation's sake, let's assume every process receives input which is a real-valued vector just like output. Generative processes then simply receive input vector of zero length.

TODO: $Input_t$ -> [Process] -> $Output_{t+1}$

In every time step, process receives input, updates its internal state and produces output. Internal state of the process is usually hidden and can be observed only partially from output of the process.

2.2 Prediction

The goal of prediction is to approximate the process as closely as possible and hence minimise forecast error, i.e. difference between actual and forecasted value. Naive approaches include methods like averaging past data points, returning previous data point or linear extrapolation. While these methods may suffice for some very simple time series, more sophisticated methods are required to cope with real-world time series. Sophisticated methods use historical data to estimate future value by finding a model of the process that empirically fits past data.

The problem of prediction can be alternatively viewed as problem of function approximation.

$$(State_{t+1}, Output_{t+1}) = Process(State_t, Input_t)$$

Process is a function from current internal state and input to next internal state and output. Unfortunately, internal state is unknown, so

next output can only be estimated from history of inputs and outputs. History can be defined as an ordered set of past inputs and outputs.

$$History_t = ((Input_t, Output_t), (Input_{t-1}, Output_{t-1}), \dots (Input_0, Output_0))$$

Prediction is then a function from current history to next output.

$$Output_{t+1} \approx Prediction(History_t)$$

2.3 Prediction Horizon

Number of steps the prediction is made into the future is called prediction horizon. Some applications require estimating output of the process more than one step into the future. This can be achieved by chaining one step predictions or by approximating function:

$$Output_{t+h} \approx Prediction(History_t, FutureInputs)$$

2.4 Prediction Chain

As an alternative for predicting multiple steps of the future, one step prediction can be applied recursively multiple times, i.e. predict next step, from the prediction predict next step etc. Intermediary results are then available for use. In case of processes that receive input, chaining predictions is suitable only if the future inputs are known up until prediction horizon.

$$Output_{t+h} \approx Prediction(Prediction(\dots Prediction(State_t, Input_t), Input_{t+1}) \dots Input_{t+h})$$

3 Artificial Neural Networks

Inspired by biological neural networks, ANNs are groups of elementary processing units called artificial neurons connected together to form a directed graph. Nodes of the graph represent biological neurons and connections between them represent synapses. Unlike in biological neural networks, connections between artificial neurons cannot be added or removed after the network was created. Instead, connections are weighted and the weights are adapted by learning algorithm.

Input signal propagates through the network in the direction of connections until it reaches output of the network. In supervised learning, learning algorithm adapts the weights in order to minimize the difference between output of the network and desired output provided by teacher.

3.1 Artificial Neuron

The complex behavior of biological neurons was simplified to create a mathematical model of artificial neurons, also called units. Each unit consists of a real valued activation representing some property of the unit. Unit receives activations of other units via input connections, computes its output activation and sends it to other units.

Connections between units are stored in a matrix w, where w_{ij} denotes weight of the connection from unit i to unit j. Every unit j has a potential p_j which is calculated as weighted sum of all of its N input units and bias.

$$p_j = \sum_{i=1}^{N+1} w_{ij} a_i$$

Bias term, also known as threshold unit, is usually represented as an extra input unit whose activation always equals one. Presence of bias term enables shifting the activation function along x-axis by changing the weight of connection from threshold unit.

Activation of the unit a_j is then computed from the potential p_j transformed by a non-linear activation function act.

$$a_j = act\left(p_j\right)$$

Commonly used non-linear activation function ranging from 0 to 1 is sigmoid function thanks to its easily computable derivative which is used by learning algorithms.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)\left(1-\sigma(x)\right)$$

3.2 Feedforward Neural Networks

Feedforward neural network is an ANNs where information moves in one direction, from input to output, i.e. without any backward or recurrent connections. Multilayer perceptron (MLP) is a class of feedforward networks consisting of three or more layers of units. Layer is a group of units receiving connections from the same units. Units inside a layer are not connected to each other.

MLP consists of three types of layers: input layer, one or more hidden layers and output layer. Input layer is the first layer of network and it receives no connections from other units, but instead holds network's input vector as activation of its units. Input layer is fully connected to first hidden layer. Hidden layer i is then fully connected to hidden layer $i+1$. Last hidden layer is fully connected to output layer. Activation of output units is considered to be output of the network.

MLPs are often used to approximate unknown functions from their inputs to outputs. MLP's capability of approximating any continuous function with support in the unit hypercube with only single hidden layer and sigmoid activation function was first proved by George Cybenko [1].

3.2.1 Backpropagation

Backpropagation, or backward propagation of errors, is the most used supervised learning algorithm for adapting weights of feedforward

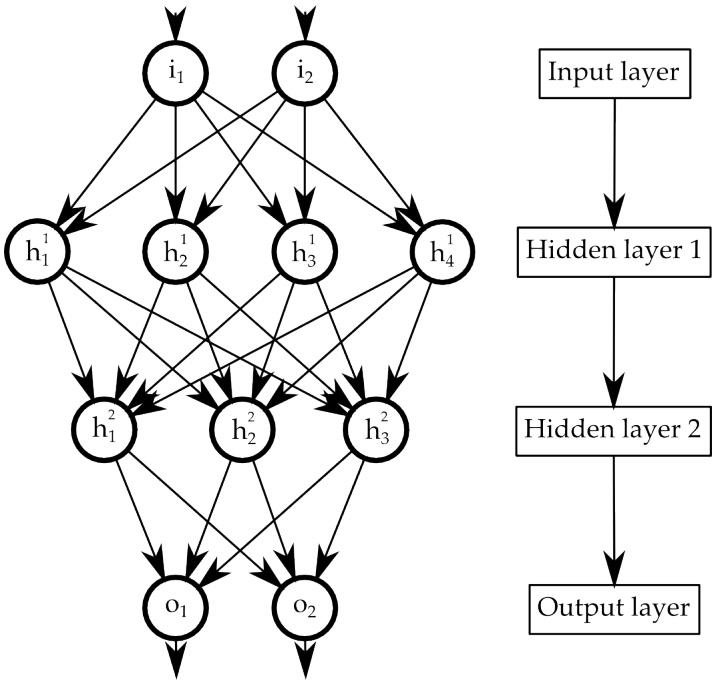

Figure 3.1: On the left, MLP consisting of input layer with two units, two hidden layers with four and three units respectively, and output layer with two untis. Schematic diagram of the MLP's layers on the right.

ANNs. Weights of the network are tuned so as to minimize summed squared error

$$E = \frac{1}{2}(t - o)^2$$

where t denotes target output provided by teacher and o is network's prediction of the output for the corresponding input.

Let's assume the error is a function of network's weights, then backpropagation can be seen as optimization problem and standard gradient descent method can be applied. Local minimum is approached by changing weights along the direction of negative error gradient

$$-\frac{\partial E}{\partial w}$$

proportionally to α, which is constant positive value called learning

rate.

$$new\ w_{ij} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}}$$

The central part of the algorithm is finding the error gradient. Let's assume there is a MLP with L layers, first being input and last being output layer. Layer k has U_k units and holds a matrix of weights w_{ij}^k representing weights of connections from unit i in layer $k-1$ to unit j in layer k. The computation can be then divided into three steps:

1. Forward propagation. Input vector is copied to activation of input layer. Layer by layer, from first hidden to output layer, activation of units is calculated. Activation of the output layer a^L is considered output of the network.

2. Backward propagation. Compute error gradient Δ_i^L w.r.t. unit i for each output layer unit i as

$$\Delta_i^L = \left(target_i - a_i^L\right) \frac{\partial act\left(p_i^L\right)}{\partial p_i^L}$$

   For units i of hidden layers $h = L-1,\ L-2,\ \ldots 2$, error term is

$$\Delta_i^h = \sum_{j=1}^{U_{h+1}} \Delta_j^{h+1} w_{ji}^h \frac{\partial act\left(p_i^h\right)}{\partial p_i^h}$$

3. Weights update. Change weights in layer k according to

$$new\ w_{ij}^k = w_{ij}^k + \alpha \Delta_i^{k+1} a_j^k$$

3.3 Recurrent Neural Networks

Recurrent network is a class of ANNs which allows units to form a directed graph with cycles. This allows the network to store an internal state and consequently process sequences of inputs and thus perform temporal tasks.

3.3.1 Elman's Simple Recurrent Network

One of the simplest and most popular RNN architectures is Elman's simple recurrent network (SRN). SRN resembles a three-layer feed-forward network due to its structure composed of input, hidden and output layer, with addition of a context layer. Input and context layer connect to hidden layer, which connects to output layer. Context layer is a copy of hidden layer's activation in previous time step. Therefore, context layer acts as network's memory of previous activity.

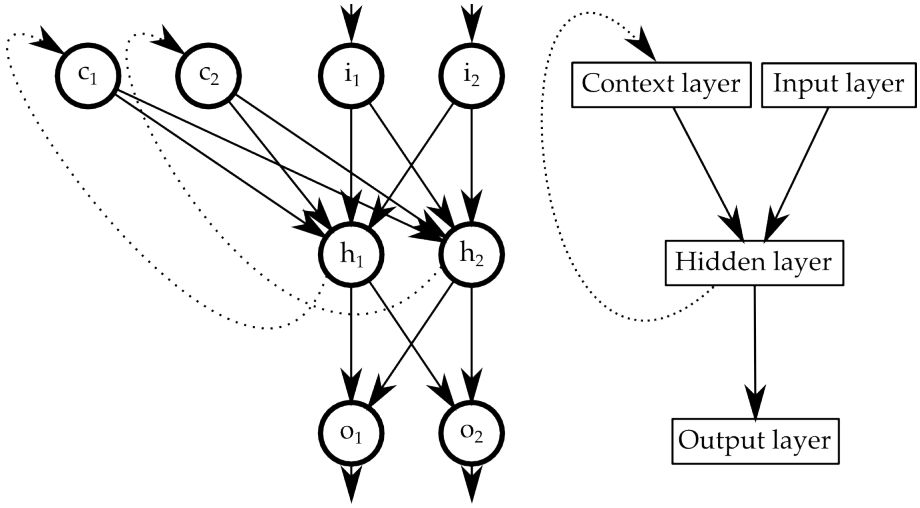

Figure 3.2: On the left, Elman's recurrent network with input, hidden, context, and output layer, each containing two units. On the right, schema of layers in Elman network. Dotted link signifies copying activity of source's units to target units.

Recurrent network's dynamics can be formulated by two equations:

$$a^{hid}(t) = act\left(W^{in}a^{in}(t) + W^{hid}a^{hid}(t-1)\right)$$

$$a^{out}(t) = act\left(W^{out}a^{hid}(t)\right)$$

where $a^{in}(t)$, $a^{hid}(t)$ and $a^{out}(t)$ are column vectors of activations of units in input, hidden and output layer respectively in time step t.

W^{in}, W^{hid} and W^{out} are matrices [1] of weights of connections from input to hidden layer, hidden to hidden layer and hidden to output layer respectively. *act* is an element wise activation function.

3.3.2 Backpropagation Through Time

Standard backpropagation algorithm is not suited for networks with cycles in them. Fortunately, RNN can be modified to look like a feedforward network by unfolding the network in time as shown in figure 3.3 and then trained with Backpropagation Through Time (BPTT) algorithm first laid out by Rumelhart, Hinton and Williams in 1986 [2].

The unfolding process begins with a SRN in current time step t, denoted as SRN_t. Since context layer of a SRN is just a copy of hidden layer activation from previous step, cycles in the network can be avoided by replacing context layer with an identical copy of the SRN network from previous step, SRN_{t-1}. Hidden layer of SRN_{t-1} is then connected to hidden layer of SRN_t. This procedure is repeated until time step 0 is reached, in which case the context layer is not replaced, but rather stays set to its initial activity. The number of SRN copies represents depth of the unfolded network and each copy of the network uses exact same set of weights.

Once the SRN has been unfolded into a feedforward network, backpropagation can be used. The algorithm again consists of 3 steps:

1. Forward propagation. Signal is propagated through the unfolded network in the standard fashion, from top to bottom. In this case, from the SRN copy furthest in the past to the most recent copy.

2. Backward propagation. Error gradient $\Delta_i^{out}(t)$ of every output layer of SRN copy in time t is computed w.r.t. output unit i as:

$$\Delta_i^{out}(t) = \left(target_i(t) - a_i^{out}(t)\right) \frac{\partial act\left(a_i^{out}(t)\right)}{\partial a_i^{out}(t)}$$

1. Element at row i and column j of the matrix W holds weight of connection from unit j to unit i

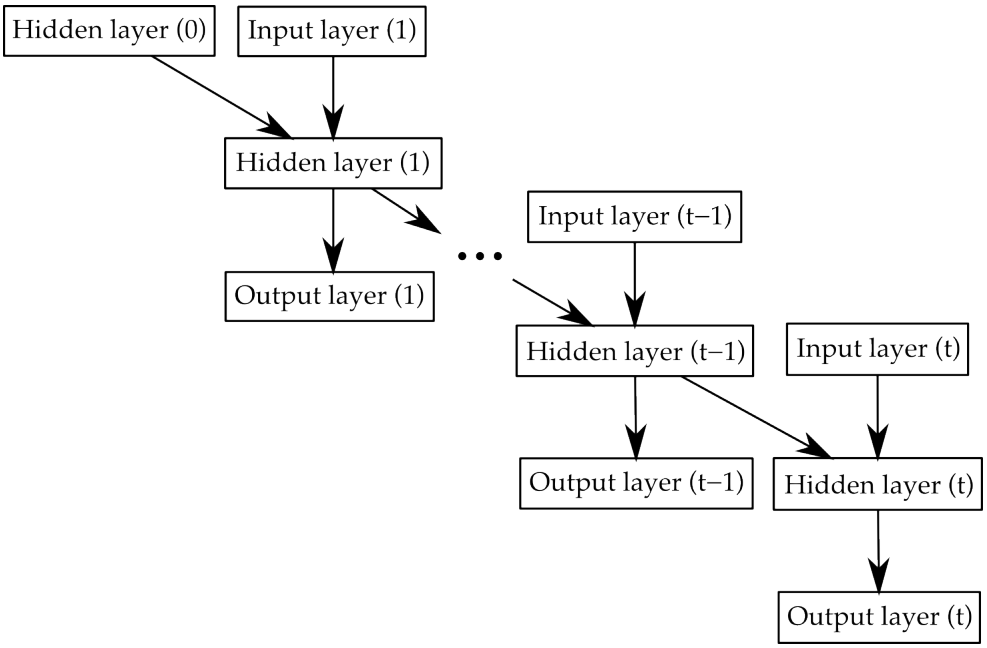

Figure 3.3: Elman's recurrent network unfolded in time.

For every unit i in hidden layer of unfolded SRN in time step t, let $\Delta_1 \dots \Delta_N$ be error terms of units that receive connections from hidden layer in time t. Error term $\Delta_i^{hid}(t)$ w.r.t. hidden unit i is then

$$\Delta_i^{hid}(t) = \left(\sum_{j=1}^{N} \Delta_j w_{ij}^h \right) \frac{\partial act\left(a_i^{hid}(t)\right)}{\partial a_i^{hid}(t)}$$

3. Update weights in the original SRN. For every unit i of the original network k that is connected to units j in layer l, update the weight according to

$$new\ w_{ij}^l = w_{ij}^l + \alpha \sum_{m=1}^{t} \Delta_i^k(m)\, a_j^l(m)$$

3.3.3 Truncated Backpropagation Through Time

Number of SRN copies in the unfolded network is equal to current time step t. Should this algorithm be used in online manner, it would

be impractical, since its memory footprint would grow linearly with time. To overcome this, online version of the BPTT algorithm called Truncated Backpropagation Through Time (TBTT) can be used. TBPTT works analogously to BPTT, except the maximum depth of the unfolded network is limited.

3.3.4 Real-Time Recurrent Learning

Real-Time Recurrent Learning (RTRL) algorithm is a gradient descent method suitable for online learning of recurrent networks.

Let's assume the recurrent network's total number of U units is divided into U_{in} input units, U_{hid} hidden units and U_{out} output units. For convenience, let's denote potential and activation of all units by p_i and a_i, where $i = 1 \dots U_{in}$ represents indices of input units, $i = U_{in}+1 \dots U_{in}+U_{hid}$ represents indices of hidden units and $i = U_{in}+U_{hid}+1 \dots U_{in}+U_{hid}+U_{out}$ represents indices of output units. All weights of connections from unit i to unit j can be then denoted by w_{ij}.

We wish to minimise error E in time step t

$$E(t) = \frac{1}{2}\left(a_i(t) - target_i(t)\right)^2$$

where i enumerates indices of output units and *target* holds teacher given desired activations of output units. We do this by adjusting weights along the negative gradient of error

$$-\frac{\partial E(t)}{\partial w_{ij}} = \sum_{k=U_{in}+U_{hid}+1}^{U} \left(target_i(t) - a_i(t)\right) \frac{\partial a_i(t)}{\partial w_{ij}}$$

$\partial a_i(t)/\partial w_{ij}$ can be computed by differentiating the network dynamics equation, resulting in the derivative v_{ij}^k of hidden or output unit k w.r.t. weight w_{ij}

$$v_{ij}^k(t+1) = \frac{\partial a_k(t+1)}{\partial w_{ij}} = act'(p_k(t)) \left[\left(\sum_{j=U_{in+1}}^{U} w_{kj} \frac{\partial a_j(t)}{\partial w_{ij}} \right) + \delta_{ki} act_j(t) \right]$$

where δ_{ki} is Kronecker's delta

$$\delta_{ki} = \begin{cases} 1, & \text{if } k = i, \\ 0, & \text{if } k \neq i. \end{cases}$$

This creates dynamical system with variables v_{ij}^k for all hidden and output units [3]. Since the initial state of the network is independent from its weights, we can set $v_{ij}^k(0) = 0$. Network's weights are then updated according to negative gradient of the error

$$new\ w_{ij} = w_{ij} - \alpha \sum_{k=U_{in}+U_{hid}+1}^{U} \left[(a_k(t) - target_k(t))\, v_{ij}^k \right]$$

3.4 Clockwork Recurrent Network

SRNs have trouble capturing capturing long-term dependencies in input sequences due to vanishing gradient [**?**]. Clockwork recurrent neural network (CW-RNN) is a modification of Elman's SRN designed to solve this problem by having hidden layer split into M modules running at different clocks [**?**]. Each module i is assigned a clock rate T_i. In time step t only modules with period T_i that satisfies $(t\ mod\ T_i) = 0$ compute its activation, other modules retain their previous activation.

Like in SRN, input layer is connected to hidden layer, context layer stores activation of hidden layer from previous time step and hidden layer is connected to output layer. The difference is that module of hidden layer with clock rate T_i connects to module in context layer with period T_j only if $T_i <= T_j$ as shown in figure 3.4. This

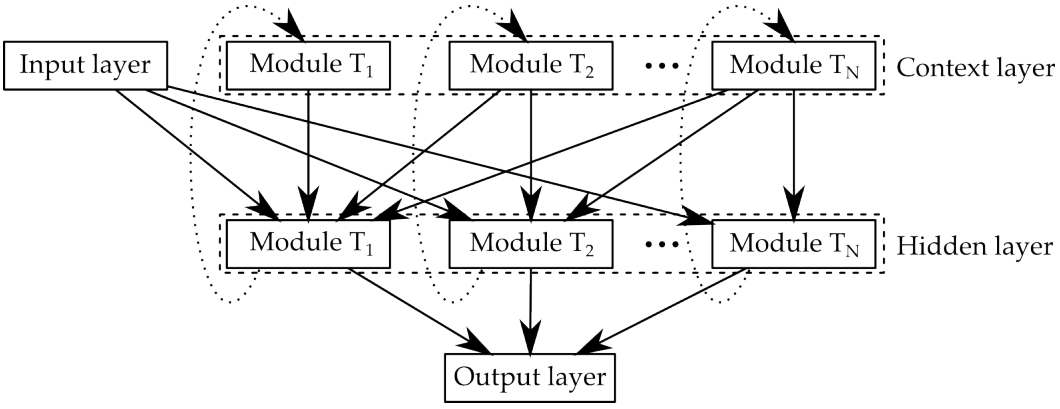

Figure 3.4: Clockwork-RNN.

allows slower modules to focus on long-term information in the input sequence, while faster modules focus on short-term information with context provided by slower modules.

To adapt network weights, BPTT learning algorithm can be used. The algorithm works similarly with the only difference compared to SRN being that error propagates only from active modules executed at time t. Error of inactive modules is retained from previous step.

4 Implementation

4.1 Neural Prediction Framework

Neural Prediction Framework (NPF) is a tool built for experimenting with various neural network models in online time series prediction tasks.

4.1.1 Memory Block

MemoryBlock is an object encapsulating an array of floating point numbers and providing convenience methods. MemoryBlock is used all around NPF to store data points, network weights or activations.

4.1.2 Neural Layer

NeuralLayer is an object storing activation of units in a layer and weights of connections coming from units of other layers.

4.1.3 Neural Network

TODO

4.2 Network Monitor

TODO

4.3 Robotic Simulator

TODO

5 Experiments

5.1 Sine Wave

TODO

5.1.1 Role of Learning and Momentum Rate

TODO

5.1.2 Impact of Network Size on Performance

TODO

5.2 Network Usage

TODO

5.3 Robotic Manipulator

TODO

6 Results

TODO

6.1 Trade-offs

TODO

7 Conclusion

TODO

A Appendix

Source code of all tested models and experiments can be found at https://github.com/karolkuna/Time-Series-Prediction-Using-Neural-Networks

Bibliography

[1] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 12 1989.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Internal Representations by Error Propagation, pages 673–695. MIT Press, Cambridge, MA, USA, 1988.

[3] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, June 1989.