# Incremental Construction of LSTM Recurrent Neural Network

Sabrine Ribeiro René Alquézar
Universitat Politècnica de Catalunya
Departament de Llenguatges e Sistemes Informàtics
Campus Nord, Jordi Girona Salgado, 1-3, C6-201,
08034, Barcelona, España.
{eslopes,alquezar}@lsi.upc.es

**Abstract**

Long Short–Term Memory (LSTM) is a recurrent neural network (RNN) that uses structures called memory blocks to allow the net remember significant events distant in the past input sequence in order to solve long time lag tasks, where other RNN approaches fail. Throughout this work we have performed experiments using LSTM networks extended with growing abilities, which we call GLSTM. Four methods of training growing LSTM have been compared. These methods include cascade and fully connected hidden layers as well as two different levels of freezing previous weights in the cascade case. GLSTM has been applied to a forecasting problem in a biomedical domain, where the input/output behavior of five controllers of the Central Nervous System control has to be modelled. We have compared growing LSTM results against other neural network approaches, including our previous work where conventional LSTM was applied to the task at hand.

## 1 Introduction

In this work, the application of four growing methods are studied to improve the learning of a recurrent neural network (RNN) called Long Short–Term Memory (LSTM) in signal prediction tasks. LSTM is a recurrent network that uses structures called memory blocks to allow the net remember significant events distant in the past input sequence in order to solve long time lag tasks, where other RNN approaches fail. LSTM have been quite successfully applied to standard benchmarks related to classification problems [GS00b, HS97], and more recently to signal forecasting problems [RA01, GS00a].

A key issue in neural network (NN) design is determining the number of hidden units required to perform input/output mapping with satisfactory performance. In recent years, attempts have been made to build NNs incrementally

in an automatic way. The techniques used to solve this problem are called constructive or growing methods.

Growing methods are designed to automate the process of determining the network topology, by modifying both the weights and connectivity of the network during learning. Thus, these methods eliminate the need to guess the network size what makes the use of NNs more user–friendly.

In the latest years a great deal of effort has been directed towards finding efficient growing algorithms [KY97]. Many RNN growing methods have been proposed. The Dynamic Node Creation [Ash89] is a method that adds a new hidden unit when the average error curve begins to flatten out too quickly. A more sophisticated method in order to train the network that uses sophisticated nonlinear least squares and quasi-Newton optimization techniques can be found in [Bel94]. In [HRM+94] an approach called Projection Pursuit Learning Network is modelled as a one hidden layer Multi-layer perceptron that learns neuron by neuron, and layer by layer cyclically after all the training patters are presented. The most used neural network growing method is the Cascade Correlation (CC) [Fah91], that combines two key ideas. The former is the cascade architecture, in which hidden units are added only one at a time. The latter is the learning algorithm, which create and installs the new hidden units by maximizing the correlation of their outputs with the residual error. In [GAP98] a sequential orthogonal approach to the building and training of single hidden layer neural networks is described. A more recent method called Sequential Approximation with Optimal Coefficients and Interacting Frequencies [RA02] has been reported that combines two key ideas. The first one is the optimization of the coefficients (or output layer weights), which provide the linear part of the approximation. The second one is the flexibility to choose the frequencies (or hidden layer weights), which provide the non-linear part.

Regarding to RNNs the most popular growing method used up to now is the Recurrent Cascade-Correlation (RCC) [Fah91] which builds an architecture that adds recurrent operation to the Cascade-Correlation architecture. Therefore, in RCC each hidden unit is provided with a single weighted self–recurrent link that feeds back its own activation value on the previous time step. As an alternative to the original RCC architecture, [KTA95] introduced the Parallel-modular RCC that is trained with natural connectionist glue, which is a concept for modularity and scaling in large phonemic neural networks [AW90, KL90]. Another more recent effort attempting to learn weights and topology of neural nets is the work of Angeline [ASP94]. The GeNeralized Acquisition of Recurrent Links Algorithm is an evolutionary algorithm that non-monotonically constructs recurrent networks to solve a given task.

Throughout this work we have performed experiments using LSTM networks extended with growing abilities, which we call GLSTM. Four methods of training growing LSTM have been compared. GLSTM has been applied to a forecasting problem in a biomedical domain, where the input/output behavior of five controllers of the Central Nervous System control has to be modelled. We have compared growing LSTM results against those reported in [BVA98] using other NN approaches and our previous work applying conventional LSTM

[RA01] to the task at hand.

In the following section we describe the LSTM architecture. In section 3 we introduce the growing LSTM approaches. In section 4 we show the case of study. In section 5 the experimental methodology used is described. In section 6 the obtained results are presented. Finally, some conclusions are given in section 7.

# 2 LSTM Recurrent Neural Network

LSTM [HS97] belongs to a class of recurrent networks that has time-varying inputs and targets. That is, points in the time series or input sequence are presented to the network one at a time. The network can be asked to predict the next point in the future or by classify the sequence or to perform some dynamic input/output association. Error signals are either generated at each point of the sequence or at the end of the sequence.

## 2.1 LSTM Structure

A fully connected LSTM architecture is a three-layer neural network composed of an input layer, a hidden layer and an output layer. The hidden layer has a feedback loop to itself, i.e., at time step $t$ of a sequence with $n$ time steps, presented to the network, the hidden layer receives as input the activation values of the input layer and the activation values of the hidden layer at time step $t-1$. Fig. 1 illustrates a LSTM with a fully connected hidden layer consisting of two memory blocks, each one consisting of two cells. The LSTM showed has an input dimension of two and an output dimension of one. Only a limited subset of connections are shown.

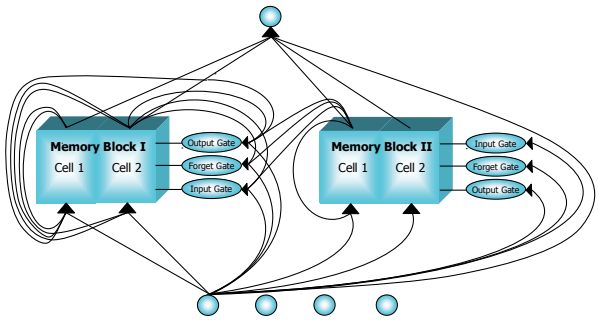


Figure 1: Example of LSTM net consisting of 4 inputs units, 1 output unit and 2 memory blocks of size 2. Only a limited subset of connections are shown.

The basic unit in the hidden layer is known as a memory cell block. A memory cell block (Fig. 2) consists of $S$ memory cells and three multiplicative gates, called the input gate, output gate and forget gate. Each memory cell has at its core a recurrently self-connected linear unit called *Constant Error*

*Carousel* (CEC), whose activation is called the cell state. The CECs solve the vanishing error problem: in the absence of a new input or error signals to the cell, the CEC's local error back flow to remains constant, neither growing nor decaying. Input and output gates regulate write and read access to a cell whose state is denoted $S_c$. The CEC is protected from both flowing activation and backward flowing error by the input and output gates respectively. When gates are closed (activation around zero), irrelevant inputs and noise do not enter the cell, and the cell state does not perturb the remainder of the network. The forget gate feed the self–recurrent connection with its output activation and is responsible for do not allow the internal state values of the cells grow without bound by resetting the internal states $S_c$ as long as it needs. In addition to the self–recurrent connection, the memory cells receive input from input units, other cells and gates.

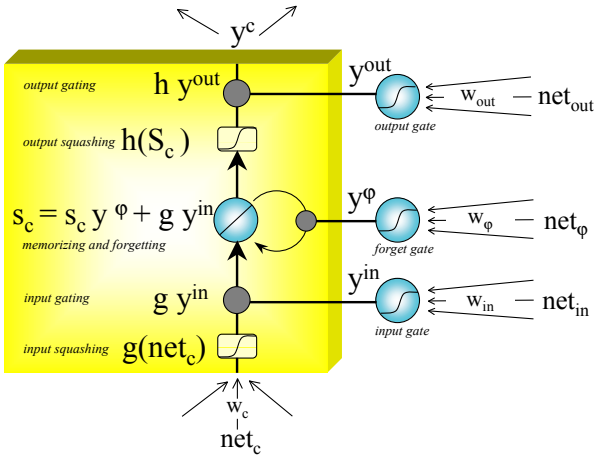


Figure 2: The standard LSTM cell with a recurrent self-connected connection and its respective gates.

While the cells are responsible for maintaining information over long periods of time, the responsibility for deciding what information to store, and when to apply that information lies with the input and output gate units, respectively.

A single step involves the update of all units (forward pass) and the computation of error signals for all weights (backward pass). The equations that describe the dynamics of the LSTM network can be found in [7].

## 2.2 Learning

LSTM's backward pass [HS97] is basically a fusion of slightly modified truncated back-propagation through time (BPTT) [WP90], which is obtained by truncating the backward propagation of error information, and a customized version of RTRL [RF87] which properly takes into account the altered (sigma-pi-like) dynamics caused by input and output gates (see details in [HS97]).

Output units use BPTT; output gates use a truncated version of BPTT. However, weights to the cells and forget gates use a truncated version of RTRL. Truncation means that all errors are cut off once they leak out of a memory cell or gate, although they do serve to change the incoming weights. The effect is that the CECs are the only part of the system through which errors can flow forever. So, the error signals flowing out of the CEC and the multiplicative gates are truncated after they are used to update the incoming, weighted connections.

# 3 Growing LSTM

Aiming to improve the learning abilities of LSTM neural net, this work deals with a version of LSTM where the network topology is incrementally adapted by adding new memory blocks. We call this version Growing LSTM (GLSTM).

In growing algorithms, many heuristics can be used to guide the search in the possible solution space. An important problem is how to set the weight connections of a newly added block. LSTM starts training with just one memory block and grows by inserting blocks, one at time, on the hidden layer in two basic ways: *cascade* and *fully connected*, that are discussed posteriorly.

In both architectures, each new memory block receives a connection from each of the network's original inputs. However the connections that comes from other blocks changes according to the architecture used. Every new block has the same number of cells than the first initial block.

## 3.1 Cascade Growing LSTM

In the recurrent cascade architecture, the new units are added one at a time and each new unit receives inputs from every preexisting units and also from itself. Thus, carrying out this concept on LSTM, each new memory block in addition to have a self-connect connection will also receives a link from each of the network's original inputs and also from every memory cell on preexisting blocks, as can be seen in Fig. 3.

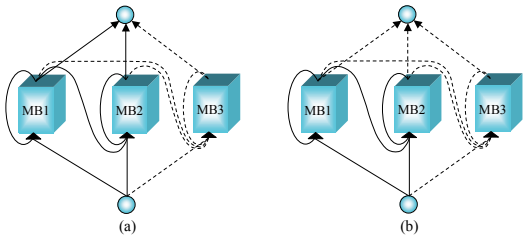


Figure 3: Cascade growing. Whereas in (a) all preexisting weights are frozen, in (b) only weights arriving at already existing blocks are frozen, letting the hidden-output weights free.

In this work, the weights arriving and leaving from the memory blocks were

frozen in two different ways. In the first one, all the preexisting weights in the whole net were frozen, leaving only the new block weights to be trained. In the second way, previous weights were frozen except for those of the output units, that remained trainable.

Fig. 3 illustrates these two configurations. In both, it can be supposed that there exist just two memory blocks (MB1 and MB2) and a new memory block (MB3) will be added. For the first configuration (Fig. 3(a)), the weights arriving at already existing blocks (MB1 and MB2) are kept frozen (solid lines) and those arriving at new memory block (MB3) are trained repeatedly (dashed lines). The output unit weights are configured in a similar way. In the second configuration (Fig. 3(b)) the same procedure is carried out, but now all hidden-output weights are modified during subsequent training.

A third configuration, not shown in the figure, is not to freeze any weight in the net, so the preexisting weight connections still can be trained further after the addition of a new memory block.

### 3.2 Fully Connected Growing LSTM

In addition to the fact that the new memory block receives connections from every preexisting blocks, as occurs in the cascade architecture, in the fully connected architecture the preexisting blocks also receives weight connections from each new added block. As can be seen in Fig. 4 no weight in the net is frozen during training, since every cell of each memory block receives new connections after adding a new block.

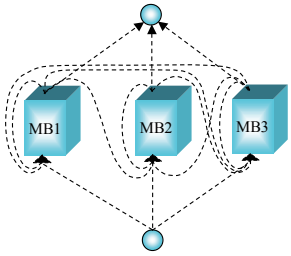


Figure 4: Fully GLSTM.

## 4 Case of Study

The human cardiovascular system is composed of the hemodynamical system and the Central Nervous System (CNS) control. In this work we try to model the latter by capturing its input/output dynamic behavior.

The CNS generates the regulating signals for the blood vessels and the heart, and it is composed of five controllers: the heart rate controller (HRC), the peripheric resistance controller (PRC), the myocardiac contractility controller

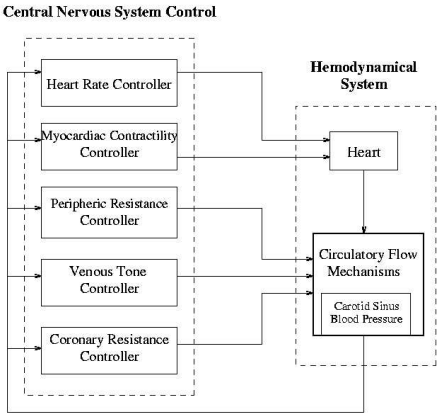


Figure 5: Simplified diagram of the cardiovascular system model, composed of the hemodynamical system and the CNS control.

(MCC), the venous tone controller (VTC), and the coronary resistance controller (CRC). A simplified diagram of the cardiovascular system is shown in Fig. 5. All of these controllers are single–input/single-output (SISO) systems driven by the same input variable, namely the carotid sinus pressure. Although the Carotid Sinus Pressure is not easily measurable, it can be extracted from the differential equation model describing the hemodynamics of the cardiovascular system [Val93]. The five output variables of the controller models are not even amenable to a physiological interpretation, except for the heart rate controller variable, which is the inverse heart rate, measured in seconds between beats.

Whereas the structure and functioning of the hemodynamical system are well known and a number of quantitative models, mostly based on differential equations, have been developed, the functioning of the central nervous system control is of high complexity and still not completely understood. Although some differential equation models for the central nervous system have been postulated [SS74], these models are not accurate enough, and therefore, the use of other modelling approaches like neural networks may offer an interesting alternative for capturing the behavior of the CNS control [LK90].

# 5 Experimental Methodology

## 5.1 Prediction Strategy

Prediction tasks involve the use of currently available input and output points to predict a future output point, i.e., given two finite sequences $x(1), x(2), x(3), \ldots, x(t)$ of input signal values and $y(1), y(2), y(3), \ldots, y(t-1)$ of output signal values, predict the value $y(t-1+T)$ of the output signal.

To prepare the data conveniently, we have replaced the original target output

$y(t)$ by the difference between the $y(t)$ output value and the previous value $y(t-1)$ multiplied by a scaling factor $fs$, so that the target is calculated as $t_g(t) = fs * (y(t) - y(t-1)) = \triangle y(t) * fs$. $fs$ scales $\triangle y(t)$ between $-1$ and 1.

Stepwise and iterated predictions are made. In single-step prediction the network predicts the next output point, $y(t)$, after being fed with the current input $x(t)$ and the last known value of the output, $y(t-1)$. In this case, $T = 1$. It should be noted that, both the inputs and the desired response are provided from the known training points.

During iterated prediction with $T = n$ the output is clamped to the $y-$input and the predicted values are fed back $n$ times, i.e. the $y-$input samples are progressively substituted by the output of the network. This closed loop system is illustrated in Fig. 6.

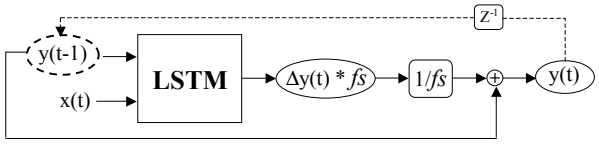


Figure 6: Setup for the output signals.

## 5.2 The Data

The data used in the training and test phases are composed of the five controllers mentioned in §4: HRC, PRC, MCC, VTC and CRC, that were recorded with a sampling rate of 0.12 seconds from simulations of a purely differential equation model. This model had been tuned to represent a specific patient suffering from coronary arterial obstruction, by making the four different physiological variables (right auricular pressure, carotid pressure, coronary blood flow, and heart rate) of the simulation model agree with the measurement data taken from the patient.

The training set consists of $1,500$ data points for each controller. Each trained network was validated by using it to forecast six data sets that had not been employed in the learning process. Each one of these six test sets (for each controller), with a size of 300 points each, contains signals representing specific morphologies, allowing the validation of the model for different system behaviors.

## 5.3 Experimental Setup

The GLSTM network architecture used is made up of an input layer with 2 inputs units, an output layer with 1 output unit and a hidden layer consisting of memory cell blocks of size 1. A new memory block is added to the hidden layer when the previous configuration has been trained at least 1000 epochs and the mean of the error in the last 10 epochs has not improved the previous error

mean up to 15 memory blocks or a maximum number of epochs, which depends on the task.

After preliminary experiments, the bias weights for input and output gates in successive blocks were fixed as: $-0.5$, $-1.0$, $-1.5$, and so forth. The initialization of output gates pushes initial memory cells activations towards zero, whereas that of the input gates prevents memory cells from being modified by incoming inputs. As training progresses, the biases become progressively less negative, allowing the serial activation of cells as active participants in the network computation.

The forget gates were initialized with symmetric positive values: $+0.5$ for the first block, $+1.0$ for the second, $+1.5$ for the third, and so forth. The bias initialization must be positive in this case, since it prevents the cells from forgetting everything [GS00a].

For the cell's input squashing function $g$ different configurations were used, depending on the controller. More specifically, for the HRC and CRC training sets the antisymmetric logarithm function [Alq97] was used, whereas for VTC, MCC and PRC controllers $g$ was the logistic sigmoid in $[-1, 1]$. As regard to the output squashing function $h$ and the activation function of the output unit, they were fixed as the linear identity function.

After some preliminary experiments, the number of epochs chosen to stop training was 5000 for HRC, MCC and VTC controllers, 2500 for PRC and 2000 for CRC control. The learning rate has been changed according to the dataset to be learned, but, in essence, it took on values from 0.01 to 0.025. The momentum parameter for each controller was 0.0 for HCR, PRC and CRC training sets and 0.5 for VTC and MCC training sets.

The error measure is given by the normalized mean square error (NMSE), in percent, between the predicted output value and the target value, $y$:

$$NMSE = \frac{E[(y(t) - \hat{y}(t))^2]}{y_{\text{var}}} \cdot 100\% \tag{1}$$

where $y_{var}$ is the variance of $y$. During training, the above NMSE error is used to determine when to finish the learning process, as explained earlier.

# 6 Results

In this section, two kinds of analysis are carried out. In the former, comparisons among different growing architectures are made and in the latter, GLSTM's results obtained throughout the present study are showed and compared with those reported in previous studies on the same task using neural net approaches.

Table 1 displays the outcomes obtained by each controller on the training and test set using the different growing architectures and configurations (see §3). *Fully* stands for fully connected growing without freezing weights. In order to investigate the effects of freezing weights, we tested three different choices for cascade growing LSTM. *Cascade Freezing* refers to the configuration illustrated by Fig. 3(a), *Cascade Not-Freezing* refers to growing cascaded without freezing

| | Fully | | Cascade Freezing | | Cascade Not-Freezing | | Cascade Trainable-Out | |
|---|---|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test | Train | Test |
| HRC | 4.85 | 3.99 | 3.09 | 4.11 | 3.26 | 4.15 | 2.11 | 3.27 |
| PRC | 0.12 | 0.69 | 0.13 | 0.66 | 0.17 | 1.33 | 0.09 | 0.61 |
| MCC | 0.39 | 1.11 | 0.14 | 0.99 | 0.11 | 1.02 | 0.09 | 0.98 |
| VTC | 0.08 | 0.96 | 0.10 | 0.96 | 0.18 | 1.05 | 0.08 | 0.96 |
| CRC | 0.22 | 0.37 | 0.11 | 0.35 | 0.18 | 0.38 | 0.09 | 0.26 |
| Av. | 1.35 | 1.42 | 0.71 | 1.41 | 0.78 | 1.58 | 0.48 | 1.21 |

Table 1: Average of three trials NMSE errors (in percent) for training and test sets of each controller using different growing architectures.

any weights, and *Cascade Trainable-Out* refers to Fig. 3(b) scheme. The results showed are the average of three different training trials using different initial random weights. As it can be seen, the results achieved by *Cascade Trainable-Out* is far better than those obtained by the other configurations. From now on, GLSTM results this configuration.

Next, refer to the outcomes accomplished in [RA01], which applies LSTM to the task at hand, are used as a baseline to illustrate GLSTM improvements on LSTM. Moreover, the results reported in [BVA98] are also recalled for comparison purposes. In [BVA98] four different approaches were performed over the task at hand, where three of them are TDNNs [KL90, HKP91] that differ in the training method used: a standard backpropagation algorithm (TDNN-BP), a hybrid procedure composed by repeated cycles of simulated annealing coupled with conjugate gradient algorithm (TDNN-AC), and a genetic algorithm (TD-HNN). This last network uses indeed a different neuron model based on a similarity computation. The other one is a RNN approach, an ASLRNN net, similar to Elman's SRN net, except that is trained by a true gradient descent algorithm that does not truncate error propagation backwards in time.

Concerning to stepwise prediction results, Table 2 shows the average NMSE errors (in percent) of each of the above mentioned architectures against those yielded by GLSTM. For each controller, three different training trials using different random weight initialization in the range $[-0.1, 0.1]$ were carried out. The net built in each trial was applied to the six test sets associated with the controller.

In order to compare the long-term prediction results, where the whole test set is attempted to be predicted ($T = 300$), Table 3 shows the average NMSE errors for different dynamic input/output architectures of three independent training trials for the six test sets of each controller. As can be seen, GLSTM outperforms in most cases those results achieved by ASLRNN, TDNN-BP and LSTM.

Finally, Fig. 7 shows the error curve of LSTM trained with 1, 2 and 3 memory blocks and LSTM trained with cascade growing method (GLSTM) for the HRC controller. The black points showed in the graph indicate the epoch in which a new memory block was added to the net. As it can be observed,

| | TD-HNN | | TDNN-BP | | TDNN-AC | | ASLRNN | | LSTM | | GLSTM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test | Train | Test | Train | Test | Train | Test |
| HRC | 0.11 | 0.18 | 1.15 | 1.52 | 0.15 | 0.13 | 1.63 | 1.91 | 2.16 | 3.41 | 2.11 | 3.27 |
| PRC | 0.09 | 0.12 | 0.94 | 1.27 | 0.26 | 0.14 | 0.84 | 1.10 | 0.25 | 0.65 | 0.09 | 0.61 |
| MCC | 0.03 | 0.06 | 0.81 | 1.33 | 0.09 | 0.08 | 0.71 | 1.18 | 0.19 | 1.04 | 0.09 | 0.98 |
| VTC | 0.03 | 0.06 | 0.81 | 1.33 | 0.09 | 0.08 | 0.71 | 1.18 | 0.19 | 1.01 | 0.08 | 0.96 |
| CRC | 0.10 | 0.11 | 0.47 | 0.66 | 0.03 | 0.04 | 0.41 | 0.53 | 0.18 | 0.31 | 0.09 | 0.26 |
| Av. | 0.07 | 0.11 | 0.84 | 1.22 | 0.12 | 0.09 | 0.86 | 1.18 | 0.59 | 1.28 | 0.48 | 1.21 |

Table 2: Average NMSE errors (in percent) for step-wise prediction on training and test sets.

| | | TDNN-BP | ASLRNN | LSTM | GLSTM |
|---|---|---|---|---|---|
| HRC | Data Set 1 | 24.31 | 28.25 | 32.81 | 25.52 |
| | Data Set 2 | 7.47 | 8.62 | 28.66 | 21.36 |
| | Data Set 3 | 13.48 | 16.77 | 28.40 | 19.52 |
| | Data Set 4 | 6.87 | 8.16 | 23.25 | 19.23 |
| | Data Set 5 | 32.12 | 38.24 | 31.70 | 24.73 |
| | Data Set 6 | 7.86 | 9.80 | 27.86 | 33.96 |
| Average Error | | 15.35 | 18.31 | 28.78 | 24.05 |
| PRC | Data Set 1 | 58.15 | 50.07 | 12.30 | 6.68 |
| | Data Set 2 | 17.80 | 16.11 | 17.29 | 6.55 |
| | Data Set 3 | 41.56 | 36.89 | 12.15 | 6.53 |
| | Data Set 4 | 29.09 | 26.97 | 7.15 | 35.00 |
| | Data Set 5 | 34.73 | 38.54 | 21.15 | 71.47 |
| | Data Set 6 | 21.22 | 18.40 | 14.22 | 5.48 |
| Average Error | | 33.76 | 31.16 | 14.04 | 21.95 |
| MCC | Data Set 1 | 41.72 | 55.83 | 27.48 | 17.16 |
| | Data Set 2 | 20.92 | 17.18 | 42.88 | 26.26 |
| | Data Set 3 | 40.22 | 35.60 | 26.26 | 17.60 |
| | Data Set 4 | 39.80 | 42.08 | 14.44 | 33.88 |
| | Data Set 5 | 34.32 | 36.87 | 16.15 | 19.27 |
| | Data Set 6 | 27.20 | 23.38 | 30.91 | 20.01 |
| Average Error | | 34.04 | 35.16 | 26.35 | 22.36 |
| VTC | Data Set 1 | 41.68 | 54.25 | 27.01 | 11.36 |
| | Data Set 2 | 20.90 | 16.93 | 35.43 | 9.99 |
| | Data Set 3 | 40.22 | 35.68 | 10.16 | 10.02 |
| | Data Set 4 | 39.80 | 41.86 | 14.58 | 32.37 |
| | Data Set 5 | 34.41 | 36.77 | 15.50 | 40.19 |
| | Data Set 6 | 27.22 | 23.12 | 29.90 | 9.41 |
| Average Error | | 34.04 | 34.77 | 22.02 | 18.89 |
| CRC | Data Set 1 | 147.73 | 148.65 | 3.70 | 6.67 |
| | Data Set 2 | 28.35 | 36.17 | 4.63 | 15.75 |
| | Data Set 3 | 84.35 | 83.75 | 3.00 | 4.29 |
| | Data Set 4 | 4.69 | 4.49 | 5.48 | 6.59 |
| | Data Set 5 | 56.20 | 58.50 | 72.29 | 44.01 |
| | Data Set 6 | 12.32 | 11.16 | 2.99 | 4.64 |
| Average Error | | 55.69 | 57.12 | 14.73 | 13.65 |

Table 3: Average NMSE errors for long-term prediction

GLSTM finds a certain stability by adding memory blocks incrementally and keeping previous hidden-layer weights frozen.

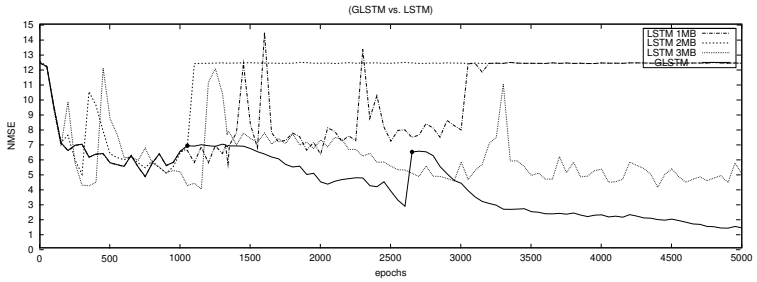


Figure 7: GLSTM vs. LSTM prediction.

# 7 Conclusions

The need to fix the size and topology of a network before its training phase is one of the most important practical problems when using common neural network approaches. Therefore, the use of incremental growing methods that build the network automatically are seen as a practical way of making the use of neural networks less complex. This is specially desirable in the case of recurrent neural networks, where the topology can be extremely complex.

Most of the previous work on NN growing methods has dealt with feedforward NNs and only a few techniques have been proposed for some particular types of recurrent NNs. In this work, we have studied the incremental construction of the LSTM net, which is maybe the more powerful RNN architecture proposed so far. Some different alternatives have been identified and tested on a signal forecasting task concerning the learning of models for the Central Nervous System Control. These include cascade and fully connected hidden layers as well as different levels of freezing previous weights in the cascade case.

It has been shown that, in addition to remove the need to fix the number of hidden units in advance, the growing LSTM can yield a better performance both in the training and test phases. Moreover, the behavior of the error minimization during the training phase appears to be more stable when using Growing LSTM with frozen weights.

Nevertheless, the experiments carried out here need to be complemented with new studies on many other different problems. In addition, the final objective of our work is not only to build LSTM nets incrementally, but rather to develop a methodology for the incremental construction of recurrent NNs for prediction tasks that can combine LSTM units (memory blocks) with other RNN architectures or even with Time-Delay Neural Networks. The underlying idea is to build a RNN as simplest as possible for a given problem and only adding more sophisticated elements (such as LSTM memory blocks) in a

parsimonious way when strictly required to improve the approximation and/or generalization performance.

## References


[Alq97] R. Alquézar. *Symbolic and Connectionist Learning Techniques for Grammatical Inference*. PhD thesis, Technical University of Catalonia, 1997.

[Ash89] T. Ash. Dynamic Node Creation in Backpropagation Networks. *Connection Science*, 1(4):365–375, 1989.

[ASP94] P.J. Angeline, G.M. Saunders, and J.B. Pollack. An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, 5(1):54–64, 1994.

[AW90] K. Shikano A. Waibel, H. Sawai. Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks. In *Proc of the Int. Conf. on Acoust, Speech, and Signal Processing*, volume S3.9, pages 112–115, 1990.

[Bel94] M.G. Bello. Enhanced Training Algorithms, and Integrated Training/Architecture Selection for Multilayer Perceptron Networks. *IEEE Transactions on Neural Networks*, 3(6):864–875, 1994.

[BVA98] L. Belanche, J.J. Valdés, and R. Alquézar. Fuzzy Heterogeneous Neural Networks for Signal Processing. In *Proc. of Int. Conf. of Artificial Neural Network*, 1998.

[Fah91] Fahlman, S.E. The Recurrent Cascade-correlation Architecture. *Advances in Neural Information Processing Systems*, 3:190–196, 1991.

[GAP98] A.H. Gee, S.V.B. Aiyer, and R.W. Prager. A Sequential Learning Approach for Single Hidden Layer Neural Networks. *Neural Computation*, 11:65–80, 1998.

[GS00a] F.A. Gers and J. Schmidhuber. Applying LSTM to Time Series Predictable Through Time-Window Approaches. Technical Report 22–00, IDSIA, 2000.

[GS00b] F.A. Gers and J. Schmidhuber. Recurrent Nets that Time and Count. In *Proc. of the Int. Joint Conf. on Neural Networks*, page 273, Como, Italy, 2000.

[HKP91] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, Redwood City, CA, 1991.

[HRM+94] J.N. Hwang, S.R. Ray, M. Maechler, D. Martin, and J. Schimert. Regression Modelling in Backpropagation and Projection Pursuit Learning. *IEEE Transactions on Neural Networks*, 5(3):54–64, 1994.

[HS97] S. Hochereiter and J. Schmidhuber. LSTM can Solve Hard Long Time Lag Problems. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 473–479, Cambridge, MA, 1997. MIT Press.

[KL90] G. Hinton K. Lang, A. Waibel. A TimeDelay Neural Network Architecture for Isolated Word Recognition. *Neural Computation*, 3:23–34, 1990.

[KTA95] I. Kirschning, H. Tomabechi, and J.I. Aoe. A Parallel Recurrent CascadeCorrelation Neural Network with Natural Connectionist Glue. In *Proc. of the Int. Conf. on Neural Networks*, volume 2, pages 953–956, Perth, Australia, 1995.

[KY97] T.Y. Kwok and D.Y. Yeung. Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. In *IEEE Transactions on Neural Networks*, volume 8, pages 630–645, 1997.

[LK90] A. Law and D. Kelton. *Simulation Modelling and Analysis*. McGraw Hill, New York, 1990.

[RA01] S. Ribeiro and R. Alquézar. A Comparative Study on a Signal Forecasting Task applying Long Short-Term Memory (LSTM) Recurrent Neural Networks. In *VI Simpósio Ibero-Americano de Reconhecimento de Padrões*, pages 487–495, Florianopolis, Brazil, 2001.

[RA02] E. Romero and R. Alquézar. A New Incremental Method for Function Appoximation using Feed-forward Neural Networks. In *Proc. of the Int. Joint Conf. on Neural Networks*, Honolulu, Hawaii, 2002.

[RF87] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, 1987.

[SS74] H. Suga and K. Sagawa. Instantaneous Pressure-volume Relationships and their Ratio in the Excised, Supported Canine Left Ventricle. *Circulation Research*, 53:117–126, 1974.

[Val93] M. Vallverdú. *Modelado y Simulación del Sistema de Control Cardiovascular en Pacientes con Lesiones Coronarias*. PhD thesis, Technical University of Catalonia, 1993.

[WP90] R.J. Williams and J. Peng. An Efficient Gradient-based Algorithm for on-line Training of Recurrent Network Trajectories. *Neural Networks*, 2:491–501, 1990.