Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures

Alex Graves* and Jürgen Schmidhuber*†
IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland*
TU Munich, Boltzmannstr. 3, 85748 Garching, Munich, Germany†
{alex,juergen}@idsia.ch

Abstract— **In this paper, we present bidirectional Long Short Term Memory (LSTM) networks, and a modified, full gradient version of the LSTM learning algorithm. We evaluate bidirectional LSTM (BLSTM) and several other network architectures on the benchmark task of framewise phoneme classification, using the TIMIT database. Our main findings are that bidirectional networks outperform unidirectional ones, and that LSTM is much faster and also more accurate than both standard Recurrent Neural Nets (RNNs) and time-windowed Multilayer Perceptrons (MLPs). Our results support the view that contextual information is crucial to speech processing, and suggest that BLSTM is an effective architecture with which to exploit it.** [1]

I. Introduction

For neural networks, there are two main ways of incorporating context into sequence processing tasks: collect the inputs into overlapping time-windows, and treat the task as spatial; or use recurrent connections to model the flow of time directly. Using time-windows has two major drawbacks: firstly the optimal window size is task dependent (too small and the net will neglect important information, too large and it will overfit on the training data), and secondly the network is unable to adapt to shifted or timewarped sequences. However, standard RNNs (by which we mean RNNs containing hidden layers of recurrently connected neurons) have limitations of their own. Firstly, since they process inputs in temporal order, their outputs tend to be mostly based on *previous* context (there are ways to introduce future context, such as adding a delay between the outputs and the targets; but these do not usually make full use of backwards dependencies). Secondly they are known to have difficulty learning time-dependencies more than a few timesteps long (Hochreiter et al., 2001). An elegant solution to the first problem is provided by bidirectional networks (Section II). For the second problem, an alternative RNN architecture, LSTM, has been shown to be capable of learning long time-dependencies (Section III).

Our experiments concentrate on framewise phoneme classification (i.e. mapping a sequence of speech frames to a sequence of phoneme labels associated with those frames). This task is both a first step towards full speech recognition (Robinson, 1994; Bourlard and Morgan, 1994), and a challenging benchmark in sequence processing. In particular, it requires the effective use of contextual information.

The contents of the rest of this paper are as follows: in Section II we discuss bidirectional networks, and answer a possible objection to their use in causal tasks; in Section III we describe the Long Short Term Memory (LSTM) network architecture, and our modification to its error gradient calculation; in Section IV we describe the experimental data and how we used it in our experiments; in Section V we give an overview of the various network architectures; in Section VI we describe how we trained (and retrained) them; in Section VII we present and discuss the experimental results, and in Section VIII we make concluding remarks. Appendix A contains the pseudocode for training LSTM networks with a full gradient calculation, and Appendix B is an outline of bidirectional training with RNNs.

II. Bidirectional Recurrent Neural Nets

The basic idea of bidirectional recurrent neural nets (BRNNs) (Schuster and Paliwal, 1997; Baldi et al., 1999) is to present each training sequence forwards and backwards to two separate recurrent nets, both of which are connected to the same output layer. (In some cases a third network is used in place of the output layer, but here we have used the simpler model). This means that for every point in a given sequence, the BRNN has complete, sequential information about all points before and after it. Also, because the net is free to use as much or as little of this context as necessary, there is no need to find a (task-dependent) time-window or target delay size. In Appendix B we give an outline of the bidirectional algorithm, and Figure 1 illustrates how the forwards and reverse subnets combine to classify phonemes. BRNNs have given improved results in sequence learning tasks, notably protein structure prediction (PSP) (Baldi et al., 2001; Chen and Chaudhari, 2004) and speech processing (Schuster, 1999; Fukada et al., 1999).

A. Bidirectional Networks and Online Causal Tasks

In a spatial task like PSP, it is clear that any distinction between input directions should be discarded. But for temporal problems like speech recognition, relying on knowledge of the

[1] An abbreviated version of some portions of this article appeared in (Graves and Schmidhuber, 2005), as part of the IJCNN 2005 conference proceedings, published under the IEEE copyright.

future seems at first sight to violate causality — at least if the task is online. How can we base our understanding of what we've heard on something that hasn't been said yet? However, human listeners do exactly that. Sounds, words, and even whole sentences that at first mean nothing are found to make sense in the light of future context. What we must remember is the distinction between tasks that are truly online — requiring an output after every input — and those where outputs are only needed at the end of some input segment. For the first class of problems BRNNs are useless, since meaningful outputs are only available after the net has run backwards. But the point is that speech recognition, along with most other 'online' causal tasks, is in the second class: an output at the end of every segment (e.g. sentence) is fine. Therefore, we see no objection to using BRNNs to gain improved performance on speech recognition tasks. On a more practical note, given the relative speed of activating neural nets, the delay incurred by running an already trained net backwards as well as forwards is small.

In general, the BRNNs examined here make the following assumptions about their input data: that it can be divided into finitely long segments, and that the effect of each of these on the others is negligible. For speech corpora like TIMIT, made up of separately recorded utterances, this is clearly the case. For real speech, the worst it can do is neglect contextual effects that extend across segment boundaries — e.g. the ends of sentences or dialogue turns. Moreover, such long term effects are routinely neglected by current speech recognition systems.

III. LSTM

The Long Short Term Memory architecture (Hochreiter and Schmidhuber, 1997; Gers et al., 2002) was motivated by an analysis of error flow in existing RNNs (Hochreiter et al., 2001), which found that long time lags were inaccessible to existing architectures, because backpropagated error either blows up or decays exponentially.

An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units — the input, output and forget gates — that provide continuous analogues of write, read and reset operations for the cells. More precisely, the input to the cells is multiplied by the activation of the input gate, the output to the net is multiplied by that of the output gate, and the previous cell values are multiplied by the forget gate. The net can only interact with the cells via the gates.

Recently, we have concentrated on applying LSTM to real world sequence processing problems. In particular, we have studied isolated word recognition (Graves et al., 2004b; Graves et al., 2004a) and continuous speech recognition (Eck et al., 2003; Beringer, 2004b).

A. LSTM Gradient Calculation

The original LSTM training algorithm (Gers et al., 2002) used an error gradient calculated with a combination of Real Time Recurrent Learning (RTRL)(Robinson and Fallside, 1987) and Back Propagation Through Time (BPTT)(Williams and Zipser, 1995). The backpropagation was truncated after one timestep, because it was felt that long time dependencies would be dealt with by the memory blocks, and not by the (vanishing) flow of backpropagated error gradient. Partly to check this assumption, and partly to ease the implementation of Bidirectional LSTM, we calculated the full error gradient for the LSTM architecture. See Appendix A for the revised pseudocode. For both bidirectional and unidirectional nets, we found that using the full gradient gave slightly higher perfor-

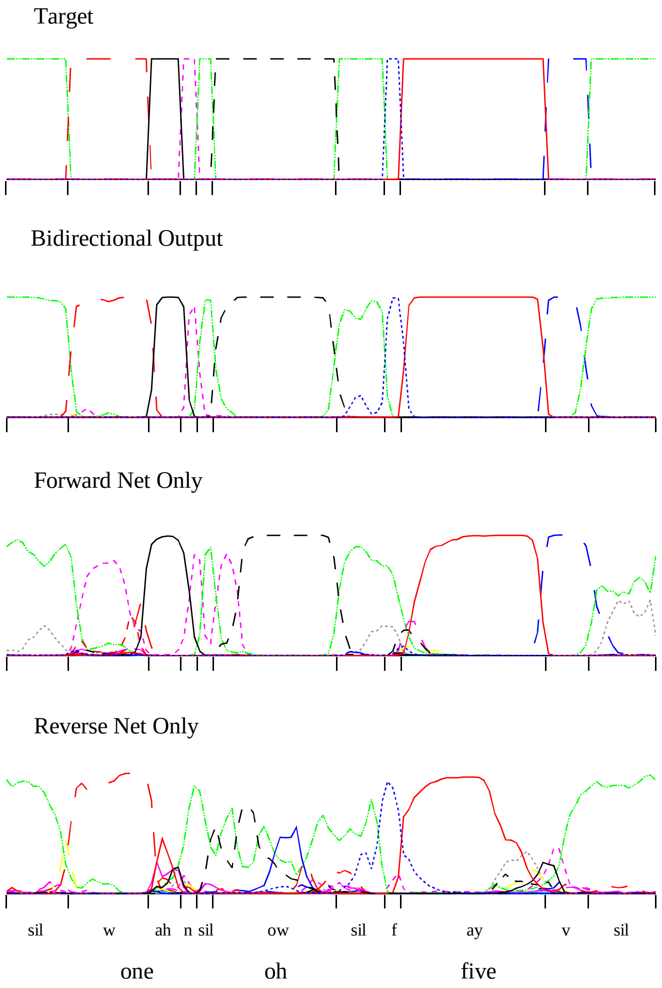

Fig. 1. A bidirectional LSTM net classifying the utterance "one oh five" from the Numbers95 corpus. The different lines represent the activations (or targets) of different output nodes. The bidirectional output combines the predictions of the forward and reverse subnets; it closely matches the target, indicating accurate classification. To see how the subnets work together, their contributions to the output are plotted separately ("Forward Net Only" and "Reverse Net Only"). As we might expect, the forward net is more accurate. However there are places where its substitutions ('w'), insertions (at the start of 'ow') and deletions ('f') are corrected by the reverse net. In addition, both are needed to accurately locate phoneme boundaries, with the reverse net tending to find the starts and the forward net tending to find the ends ('ay' is a good example of this).

mance than the original algorithm. It had the added benefit of making LSTM directly comparable to other RNNs, since it could now be trained with standard BPTT. Also, since the full gradient can be checked numerically, its implementation was easier to debug.

IV. Experimental Data

The data for our experiments came from the TIMIT corpus (Garofolo et al., 1993) of prompted utterances, collected by Texas Instruments. The utterances were chosen to be phonetically rich, and the speakers represent a wide variety of American dialects. The audio data is divided into sentences, each of which is accompanied by a complete phonetic transcript.

We preprocessed the audio data into 12 Mel-Frequency Cepstrum Coefficients (MFCC's) from 26 filter-bank channels. We also extracted the log-energy and the first order derivatives of it and the other coefficients, giving a vector of 26 coefficients per frame. The frame size was 10 ms and the input window was 25 ms.

For consistency with the literature, we used the complete set of 61 phonemes provided in the transcriptions for classification. In full speech recognition, it is common practice to use a reduced set of phonemes (Robinson, 1991), by merging those with similar sounds, and not separating closures from stops.

A. Training and Testing Sets

The standard TIMIT corpus comes partitioned into training and test sets, containing 3696 and 1344 utterances respectively. In total there were 1,124,823 frames in the training set, and 410,920 in the test set. No speakers or sentences exist in both the training and test sets. We used 184 of the training set utterances (chosen randomly, but kept constant for all experiments) as a validation set and trained on the rest. All results for the training and test sets were recorded at the point of lowest cross-entropy error on the validation set.

V. Network Architectures

We used the following five neural network architectures in our experiments (henceforth referred to by the abbreviations in brackets):

- Bidirectional LSTM, with two hidden LSTM layers (forwards and backwards), both containing 93 one-cell memory blocks of one cell each (BLSTM)
- Unidirectional LSTM, with one hidden LSTM layer, containing 140 one cell memory blocks, trained backwards with no target delay, and forwards with delays from 0 to 10 frames (LSTM)
- Bidirectional RNN with two hidden layers containing 185 sigmoidal units each (BRNN)
- Unidirectional RNN with one hidden layers containing 275 sigmoidal units, trained with target delays from 0 to 10 frames (RNN)
- MLP with one hidden layer containing 250 sigmoidal units, and symmetrical time-windows from 0 to 10 frames (MLP)

All nets contained an input layer of size 26 (one for each MFCC coefficient), and an output layer of size 61 (one for each phoneme). The input layers were fully connected to the hidden layers and the hidden layers were fully connected to the output layers. For the recurrent nets, the hidden layers were also fully connected to themselves. The LSTM blocks had the following activation functions: logistic sigmoids in the range $[-2, 2]$ for the input and output squashing functions of the cell, and in the range $[0, 1]$ for the gates. The non-LSTM nets had logistic sigmoid activations in the range $[0, 1]$ in the hidden layers. All units were biased.

None of our experiments with more complex network topologies (e.g. multiple hidden layers, several LSTM cells per block, direct connections between input and output layers) led to improved results.

A. Computational Complexity

The hidden layer sizes were chosen to ensure that all networks had roughly the same number of weights W ($\approx 100,000$). However, for the MLPs the network grew with the time-window size, and W varied between 22,061 and 152,061. For all networks, the computational complexity was dominated by the $O(W)$ feedforward and feedback operations. This means that the bidirectional nets and the LSTM nets did not take significantly more time to train per epoch than the unidirectional or RNN or (equivalently sized) MLP nets.

B. Range of Context

Only the bidirectional nets had access to the complete context of the frame being classified (i.e. the whole input sequence). For MLPs, the amount of context depended on the size of the time-window. The results for the MLP with no time-window (presented only with the current frame) give a baseline for performance without context information. However, some context is implicitly present in the window averaging and first-derivatives of the preprocessor.

Similarly, for unidirectional LSTM and RNN, the amount of future context depended on the size of target delay. The results with no target delay (trained forwards or backwards) give a baseline for performance with context in one direction only.

C. Output Layers

For the output layers, we used the cross entropy error function and the softmax activation function, as is standard for 1 of K classification (Bishop, 1995). The softmax function ensures that the network outputs are all between zero and one, and that they sum to one on every timestep. This means they can be interpreted as the posterior probabilities of the phonemes at a given frame, given all the inputs up to the current one (with unidirectional nets) or all the inputs in the whole sequence (with bidirectional nets).

Several alternative error functions have been studied for this task (Chen and Jamieson, 1996). One modification in particular has been shown to have a positive effect on full speech recognition. This is to weight the error according to the

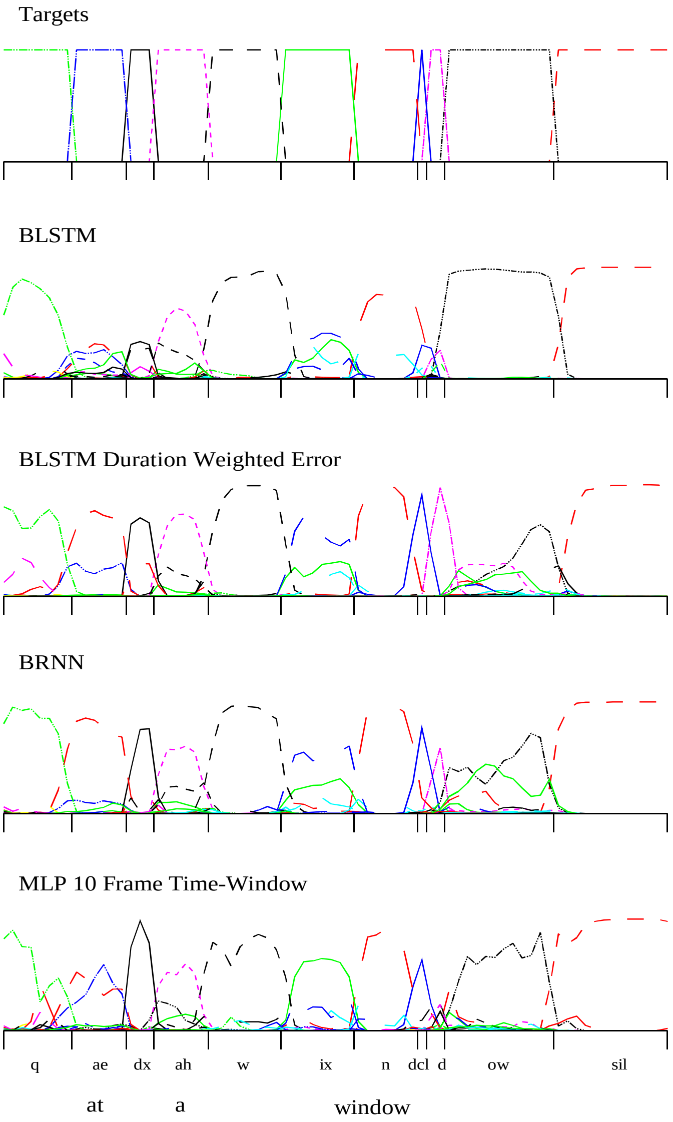

Fig. 2. The best exemplars of each architecture classifying the excerpt ”at a window” from an utterance in the TIMIT database. In general, the networks found the vowels more difficult (here, ”ix” is confused with ”ih”, ”ah” with ”ax” and ”axr”, and ”ae” with ”eh”), than the consonants (e.g. ”w” and ”n”), which in English are more distinct . For BLSTM, the net with duration weighted error tends to do better on short phones, (e.g. the closure and stop ”dcl” and ”d”), and worse on longer ones (”ow”), as expected. Note the more jagged trajectories for the MLP net (e.g. for ”q” and ”ow”); this is presumably because they have no recurrency to smooth the outputs.

duration of the current phoneme, ensuring that short phonemes are as significant to the training as longer ones. However, we recorded a slightly lower framewise classification score with BLSTM trained with this error function (see Section VII-D).

VI. Network Training

For all architectures, we calculated the full error gradient using online BPTT (BPTT truncated to the lengths of the utterances), and trained the weights using gradient descent with momentum. We kept the same training parameters for all experiments: initial weights randomised in the range $[-0.1, 0.1]$, a learning rate of 10^{-5} and a momentum of 0.9. At the end of each utterance, weight updates were carried out and network activations were reset to 0.

Keeping the training algorithm and parameters constant allowed us to concentrate on the effect of varying the architecture. However it is possible that different training methods would be better suited to different networks.

A. *Retraining*

For the experiments with varied time-windows or target delays, we iteratively retrained the networks instead of starting again from scratch. For example, for LSTM with a target delay of 2, we first trained with delay 0, then took the best net and retrained it (without resetting the weights) with delay 1, then retrained again with delay 2. To find the best networks, we retrained the LSTM nets for 5 epochs at each iteration, the RNN nets for 10, and the MLPs for 20. It is possible that longer retraining times would have given improved results. For the retrained MLPs, we had to add extra (randomised) weights from the input layers, since the input size grew with the time-window.

Although primarily a means to reduce training time, we have also found that retraining improves final performance (Graves et al., 2005; Beringer, 2004a). Indeed, the best result in this paper was achieved by retraining (on the BLSTM net trained with a weighted error function, then retrained with normal cross-entropy error). The benefits presumably come from escaping the local minima that gradient descent algorithms tend to get caught in.

TABLE I

Framewise Phoneme Classification on the TIMIT Database: Bidirectional LSTM

Network	Training Set Score	Test Set Score	Epochs
BLSTM (1)	77.0%	69.7%	20
BLSTM (2)	77.9%	70.1%	21
BLSTM (3)	77.3%	69.9%	20
BLSTM (4)	77.8%	69.8%	22
BLSTM (5)	77.1%	69.4%	19
BLSTM (6)	77.8%	69.8%	21
BLSTM (7)	76.7%	69.9%	18
mean	77.4%	69.8%	20.1
standard deviation	0.5%	0.2%	1.3

VII. Results

Table I contains the outcomes of 7, randomly initialised, training runs with BLSTM. For the rest of the paper, we use their mean as the result for BLSTM. The standard deviation in the test set scores (0.2%) gives an indication of significant difference in network performance.

The last three entries in Table II come from the papers indicated (note that Robinson did not quote framewise classification scores; the result for his network was recorded by Schuster, using the original software). The rest are from our own experiments. For the MLP, RNN and LSTM nets we give the best results, and those achieved with least contextual

TABLE II
FRAMEWISE PHONEME CLASSIFICATION ON THE TIMIT DATABASE: MAIN RESULTS

Network	Training Set	Test Set	Epochs
BLSTM (retrained)	78.6%	70.2%	17
BLSTM	77.4%	69.8%	20.1
BRNN	76.0%	69.0%	170
BLSTM (Weighted Error)	75.7%	68.9%	15
LSTM (5 frame delay)	77.6%	66.0%	34
RNN (3 frame delay)	71.0%	65.2%	139
LSTM (backwards, 0 frame delay)	71.1%	64.7%	15
LSTM (0 frame delay)	70.9%	64.6%	15
RNN (0 frame delay)	69.9%	64.5%	120
MLP (10 frame time-window)	67.6%	63.1%	990
MLP (no time-window)	53.6%	51.4%	835
RNN (Chen and Jamieson, 1996)	69.9%	74.2%	-
RNN (Robinson, 1994; Schuster, 1999)	70.6%	65.3%	-
BRNN (Schuster, 1999)	72.1%	65.1%	-

information (i.e. with no target delay / time-window). The number of epochs includes both training and retraining.

There are some differences between the results quoted in this paper and in our previous work (Graves and Schmidhuber, 2005). The most significant of these is the improved score we achieved here with the bidirectional RNN (69.0% instead of 64.7%). Previously we had stopped the BRNN after 65 epochs, when it appeared to have converged; here, however, we let it run for 225 epochs (10 times as long as LSTM), and kept the best net on the validation set, after 170 epochs. As can be seen from Figure 4 the learning curves for the non LSTM networks are very slow, and contain several sections where the error temporarily increases, making it difficult to know when training should be stopped.

The results for the unidirectional LSTM and RNN nets are also better here; this is probably due to our use of larger networks, and the fact that we retrained between different target delays. Again it should be noted that at the moment we do not have an optimal method for choosing retraining times.

A. Comparison Between LSTM and Other Architectures

The most obvious difference between LSTM and the RNN and MLP nets was the training time (see Figure 4). In particular, BRNN took more than 8 times as long to converge as BLSTM, despite having more or less equal computational complexity per time-step (see Section V-A). There was a similar time increase between the unidirectional LSTM and RNN nets, and the MLPs were slower still (990 epochs for the best MLP result).

The training time of 17 epochs for our most accurate network (retrained BLSTM) is remarkably fast, needing just a few hours on an ordinary desktop computer. Elsewhere we have seen figures of between 40 and 120 epochs quoted for RNN convergence on this task, usually with more advanced training algorithms than the one used here.

A possible explanation of why RNNs took longer to train than LSTM on this task is that they require more fine-tuning of their weights to make use of the contextual information, since their error signals tend to decay after a few timesteps. A detailed analysis of the evolution of the weights would be required to check this.

As well as being faster, the LSTM nets were also slightly more accurate. Although the final difference in score between BLSTM and BRNN on this task is small (0.8%) the results in Table I suggest that it is significant. The fact that the difference is not larger could mean that long time dependencies (more than 10 timesteps or so) are not very helpful to this task.

It is interesting to note how much more prone to overfitting LSTM was than standard RNNs. For LSTM, after only 15-20 epochs the performance on the validation and test sets would begin to fall, while that on the training set would continue to rise (the highest score we recorded on the training set with BLSTM was 86.4%, and still improving). With the RNNs on the other hand, we never observed a large drop in test set score. This suggests a difference in the way the two architectures learn. Given that in the TIMIT corpus no speakers or sentences are shared by the training and test sets, it is possible that LSTM's overfitting was partly caused by its better adaptation to long range regularities (such as phoneme ordering within words, or speaker specific pronunciations) than normal RNNs. If this is true, we would expect a greater distinction between the two architectures on tasks with more training data.

B. Comparison with Previous Work

Overall BLSTM outperformed any neural network we found in the literature on this task, apart from the RNN used by Chen and Jamieson. Their result (which we were unable to approach with standard RNNs) is surprising as they quote a substantially higher score on the test set than the training set: all other methods reported here were better on the training than the test set, as expected.

In general, it is difficult to compare with previous work on this task, owing to the many variations in training data (different preprocessing, different subsets of the TIMIT corpus, different target representations) and experimental method (different learning algorithms, error functions, network sizes etc). This is why we reimplemented all the architectures ourselves.

C. Effect of Increased Context

As is clear from Figure 3 networks with access to more contextual information tended to get better results. In particular, the bidirectional networks were substantially better than the unidirectional ones. For the unidirectional nets, note that LSTM benefits more from longer target delays than RNNs; this could be due to LSTM's greater facility with long timelags, allowing it to make use of the extra context without suffering as much from having to remember previous inputs.

Interestingly, LSTM with no time delay returns almost identical results whether trained forwards or backwards. This suggests that the context in both directions is equally important. However, with bidirectional nets, the forward subnet usually dominates the outputs (see Figure 1).

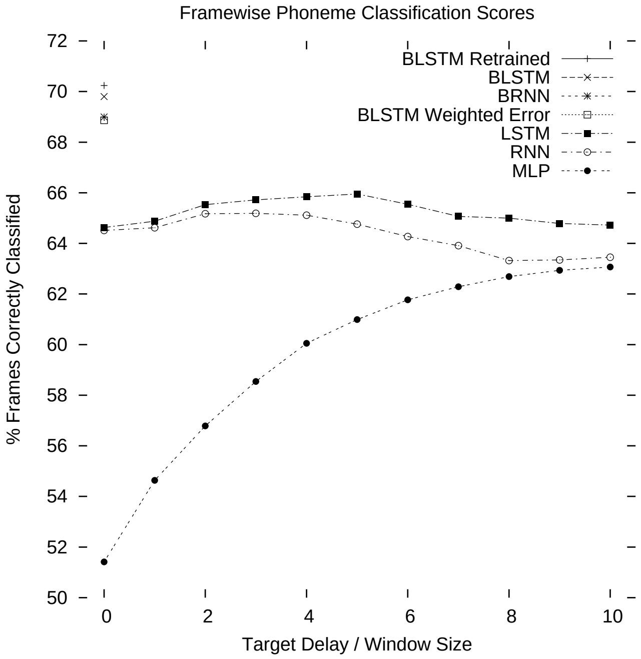

Fig. 3. Framewise phoneme classification results for all networks on the TIMIT test set. The number of frames of introduced context (time-window size for MLPs, target delay size for unidirectional LSTM and RNNs) is plotted along the x axis. Therefore the results for the bidirectional nets (clustered around 70%) are plotted at x=0.

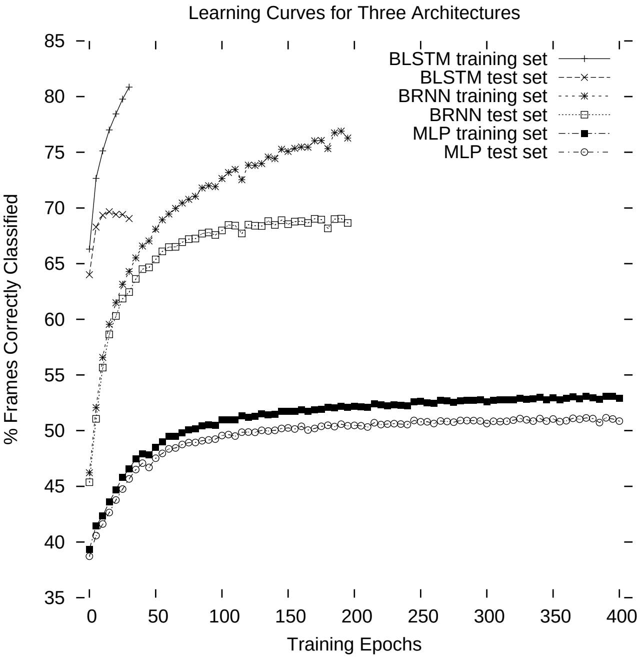

Fig. 4. Learning curves for BLSTM, BRNN and MLP with no time-window. For all experiments, LSTM was much faster to converge than either the RNN or MLP architectures.

For the MLPs, performance increased with time-window size, and it appears that even larger windows would have been desirable. However, with fully connected networks, the number of weights required for such large input layers makes training prohibitively slow.

D. Weighted Error

The experiment with a weighted error function gave slightly inferior framewise performance for BLSTM (68.9%, compared to 69.7%). However, the purpose of this weighting is to improve overall phoneme recognition, rather than framewise classification (see Section V-C). As a measure of its success, if we assume a perfect knowledge of the test set segmentation (which in real-life situations we cannot), and integrate the network outputs over each phoneme, then BLSTM with weighted errors gives a phoneme correctness of 74.4%, compared to 71.2% with normal errors.

VIII. Conclusion and Future Work

In this paper we have compared bidirectional LSTM to other neural network architectures on the task of framewise phoneme classification. We have found that bidirectional networks are significantly more effective than unidirectional ones, and that LSTM is much faster to train than standard RNNs and MLPs, and also slightly more accurate. We conclude that bidirectional LSTM is an architecture well suited to this and other speech processing tasks, where context is vitally important.

In the future we would like to apply BLSTM to full speech recognition, for example as part of a hybrid RNN / Hidden Markov Model system.

Appendix A: Pseudocode for Full Gradient LSTM

The following pseudocode details the forward pass, backward pass, and weight updates of an extended LSTM layer in a multi-layer net. The error gradient is calculated with online BPTT (i.e. BPTT truncated to the lengths of input sequences, with weight updates after every sequence). As is standard with BPTT, the network is unfolded over time, so that connections arriving at layers are viewed as coming from the previous timestep. We have tried to make it clear which equations are LSTM specific, and which are part of the standard BPTT algorithm. Note that for the LSTM equations, the order of execution is important.

Notation

The input sequence over which the training takes place is labelled S and it runs from time τ_0 to τ_1. $x_k(\tau)$ refers to the network input to unit k at time τ, and $y_k(\tau)$ to its activation. Unless stated otherwise, all network inputs, activations and partial derivatives are evaluated at time τ — e.g. $y_c \equiv y_c(\tau)$. $E(\tau)$ refers to the (scalar) output error of the net at time τ. The training target for output unit k at time τ is denoted $t_k(\tau)$. N is the set of all units in the network, including input and bias units, that can be connected to other units. Note that this includes LSTM cell outputs, but *not* LSTM gates or internal states (whose activations are only visible within their own memory blocks). W_{ij} is the weight *from* unit j *to* unit i.

The LSTM equations are given for a single memory block only. The generalisation to multiple blocks is trivial: simply repeat the calculations for each block, in any order. Within each block, we use the suffixes ι, ϕ and ω to refer to the input gate, forget gate and output gate respectively. The suffix c refers to an element of the set of cells C. s_c is the state value of cell c — i.e. its value after the input and forget gates have been applied. f is the squashing function of the gates, and g and h are respectively the cell input and output squashing functions.

Forward Pass

- Reset all activations to 0.
- Running forwards from time τ_0 to time τ_1, feed in the inputs and update the activations. Store all hidden layer and output activations at every timestep.
- For each LSTM block, the activations are updated as follows:

Input Gates:

$$x_\iota = \sum_{j \in N} w_{\iota j} y_j(\tau - 1) + \sum_{c \in C} w_{\iota c} s_c(\tau - 1)$$
$$y_\iota = f(x_\iota)$$

Forget Gates:

$$x_\phi = \sum_{j \in N} w_{\phi j} y_j(\tau - 1) + \sum_{c \in C} w_{\phi c} s_c(\tau - 1)$$
$$y_\phi = f(x_\phi)$$

Cells:

$$\forall c \in C,\ x_c = \sum_{j \in N} w_{cj} y_j(\tau - 1)$$
$$s_c = y_\phi s_c(\tau - 1) + y_\iota g(x_c)$$

Output Gates:

$$x_\omega = \sum_{j \in N} w_{\omega j} y_j(\tau - 1) + \sum_{c \in C} w_{\omega c} s_c(\tau)$$
$$y_\omega = f(x_\omega)$$

Cell Outputs:

$$\forall c \in C,\ y_c = y_\omega h(s_c)$$

Backward Pass

- Reset all partial derivatives to 0.
- Starting at time τ_1, propagate the output errors backwards through the unfolded net, using the standard BPTT equations for a softmax output layer and the cross-entropy error function:

$$define\ \ \delta_k(\tau) = \frac{\partial E(\tau)}{\partial x_k}$$
$$\delta_k(\tau) = y_k(\tau) - t_k(\tau) \qquad k \in output\ units$$

- For each LSTM block the δ's are calculated as follows:

Cell Outputs:

$$\forall c \in C,\ define\ \epsilon_c = \sum_{j \in N} w_{jc} \delta_j(\tau + 1)$$

Output Gates:

$$\delta_\omega = f'(x_\omega) \sum_{c \in C} \epsilon_c h(s_c)$$

States:

$$\frac{\partial E}{\partial s_c}(\tau) = \epsilon_c y_\omega h'(s_c) + \frac{\partial E}{\partial s_c}(\tau + 1) y_\phi(\tau + 1) + \delta_\iota(\tau + 1) w_{\iota c} + \delta_\phi(\tau + 1) w_{\phi c} + \delta_\omega w_{\omega c}$$

Cells:

$$\forall c \in C,\ \delta_c = y_\iota g'(x_c) \frac{\partial E}{\partial s_c}$$

Forget Gates:

$$\delta_\phi = f'(x_\phi) \sum_{c \in C} \frac{\partial E}{\partial s_c} s_c(\tau - 1)$$

Input Gates:

$$\delta_\iota = f'(x_\iota) \sum_{c \in C} \frac{\partial E}{\partial s_c} g(x_c)$$

- Using the standard BPTT equation, accumulate the δ's to get the partial derivatives of the cumulative sequence error:

$$define\ \ E_{total}(S) = \sum_{\tau = \tau_0}^{\tau_1} E(\tau)$$
$$define\ \ \bigtriangledown_{ij}(S) = \frac{\partial E_{total}(S)}{\partial w_{ij}}$$
$$\implies \bigtriangledown_{ij}(S) = \sum_{\tau = \tau_0 + 1}^{\tau_1} \delta_i(\tau) y_j(\tau - 1)$$

Update Weights

- After the presentation of sequence S, with learning rate α and momentum m, update all weights with the standard equation for gradient descent with momentum:

$$\Delta w_{ij}(S) = -\alpha \bigtriangledown_{ij}(S) + m \Delta w_{ij}(S - 1)$$

Appendix B: Algorithm Outline for Bidirectional Recurrent Neural Networks

We quote the following method for training bidirectional recurrent nets with BPTT (Schuster, 1999). As above, training takes place over an input sequence running from time τ_0 to τ_1. All network activations and errors are reset to 0 at τ_0 and τ_1.

Forward Pass Feed all input data for the sequence into the BRNN and determine all predicted outputs.

- Do forward pass just for forward states (from time τ_0 to τ_1) and backward states (from time τ_1 to τ_0).
- Do forward pass for output layer.

Backward Pass Calculate the error function derivative for the sequence used in the forward pass.

- Do backward pass for output neurons.
- Do backward pass just for forward states (from time τ_1 to τ_0) and backward states (from time τ_0 to τ_1).

Update Weights

Acknowledgments

The authors would like to thank Nicole Beringer for her expert advice on linguistics and speech recognition. This work was supported by the SNF under grant number 200020-100249.

References

Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., and Soda, G. (2001). Bidirectional dynamics for protein secondary structure prediction. *Lecture Notes in Computer Science*, 1828:80–104.

Baldi, P., Brunak, S., Frasconi, P., Soda, G., and Pollastri, G. (1999). Exploiting the past and the future in protein secondary structure prediction. *BIOINF: Bioinformatics*, 15.

Beringer, N. (2004a). Human language acquisition in a machine learning task. *Proc. ICSLP*.

Beringer, N. (2004b). Human language acquisition methods in a machine learning task. In *Proceedings of the 8th International Conference on Spoken Language Processing*, pages 2233–2236.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc.

Bourlard, H. and Morgan, N. (1994). *Connnectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers.

Chen, J. and Chaudhari, N. S. (2004). Capturing long-term dependencies for protein secondary structure prediction. In Yin, F., Wang, J., and Guo, C., editors, *Advances in Neural Networks - ISNN 2004, International Symposiumon Neural Networks, Part II*, volume 3174 of *Lecture Notes in Computer Science*, pages 494–500, Dalian, China. Springer.

Chen, R. and Jamieson, L. (1996). Experiments on the implementation of recurrent neural networks for speech phone recognition. In *Proceedings of the Thirtieth Annual Asilomar Conference on Signals, Systems and Computers*, pages 779–782.

Eck, D., Graves, A., and Schmidhuber, J. (2003). A new approach to continuous speech recognition using LSTM recurrent neural networks. Technical Report IDSIA-14-03, IDSIA, www.idsia.ch/techrep.html.

Fukada, T., Schuster, M., and Sagisaka, Y. (1999). Phoneme boundary estimation using bidirectional recurrent neural networks and its applications. *Systems and Computers in Japan*, 30(4):20–30.

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., , and Dahlgren, N. L. (1993). Darpa timit acoustic phonetic continuous speech corpus cdrom.

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

Graves, A., Beringer, N., and Schmidhuber, J. (2004a). A comparison between spiking and differentiable recurrent neural networks on spoken digit recognition. In *The 23rd IASTED International Conference on modelling, identification, and control*, Grindelwald.

Graves, A., Beringer, N., and Schmidhuber, J. (2005). Rapid retraining on speech data with lstm recurrent networks. Technical Report IDSIA-09-05, IDSIA, www.idsia.ch/-techrep.html.

Graves, A., Eck, D., Beringer, N., and Schmidhuber, J. (2004b). Biologically plausible speech recognition with lstm neural nets. In *First International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, Lausanne.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. In *Proceedings of the 2005 International Joint Conference on Neural Networks*, Montreal, Canada.

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

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Robinson, A. J. (1991). Several improvements to a recurrent error propagation network phone recognition system. Technical Report CUED/F-INFENG/TR82, University of Cambridge.

Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5(2):298–305.

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

Schuster, M. (1999). *On supervised learning from sequential data with applications for speech recognition*. PhD thesis, Nara Institute of Science and Technolog, Kyoto, Japan.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681.

Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Chauvin, Y. and Rumelhart, D. E., editors, *Back-propagation: Theory, Architectures and Applications*, pages 433–486. Lawrence Erlbaum Publishers, Hillsdale, N.J.