Biologically Plausible Speech Recognition with LSTM Neural Nets

Alex Graves, Douglas Eck, Nicole Beringer, Juergen Schmidhuber

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Galleria 2, 6928 Manno-Lugano, Switzerland,
alex,doug,nicole,juergen@idsia.ch

Abstract. Long Short-Term Memory (LSTM) recurrent neural networks (RNNs) are local in space and time and closely related to a biological model of memory in the prefrontal cortex. Not only are they more biologically plausible than previous artificial RNNs, they also outperformed them on many artificially generated sequential processing tasks. This encouraged us to apply LSTM to more realistic problems, such as the recognition of spoken digits. Without any modification of the underlying algorithm, we achieved results comparable to state-of-the-art Hidden Markov Model (HMM) based recognisers on both the TIDIGITS and TI46 speech corpora. We conclude that LSTM should be further investigated as a biologically plausible basis for a bottom-up, neural net-based approach to speech recognition.

1 Introduction

Identifying and understanding speech is an inherently temporal task. Not only the waveforms of individual phones, but also their duration, their ordering, and the delays between them all convey vital information to the human ear. While neural networks have dealt very successfully with certain temporal problems, they have so far been unable to fully accommodate the range and precision of time scales required for continuous speech recognition. This failing has left them a peripheral role in current speech technology.

The aim of this paper is to re-examine the neural network approach to speech recognition (SR). In particular, we are interested in providing a more robust, and biologically plausible alternative to statistical learning methods such as HMMs. In Section 2, a summary is given of the problems that the approach has suffered from in the past. In Section 3, the network architecture we will use (Long Short Term Memory, or LSTM) is introduced, and its suitability for SR is discussed. In Section 4, experimental results for LSTM on a digit recognition task are provided. Concluding remarks and future directions are presented in Section 5, and pseudocode for the LSTM training algorithm is given in Section 6.1.

2 Neural Nets in Speech Recognition

For neural nets, dealing with time dependent inputs (such as those present in speech) means one of two things: either collecting the inputs into time windows,

and treating the task as spatial; or using recurrent connections and an internal state to model time directly.

There are two drawbacks to the application of time-windowed networks to speech recognition. Firstly, the window size is fixed by the topology of the network (and usually limited by speed and memory considerations). This means that either the net has a huge number of inputs (and therefore a huge number of parameters and a very long training time), or else that important long time dependencies, such as the position of a word in a sentence, are simply ignored. Secondly, such nets are inflexible with regard to temporal displacements or changes in the rate of input (non-linear time warping), leaving them easily confused by variations in speech rate.

With recurrent neural nets (RNNs), on the other hand, temporal patterns are not transformed into spatial ones. Instead, a time series is presented one frame at a time, with the flow of activations around the connections creating a memory of previous inputs. Recurrent training algorithms such as Backpropagation Through Time (BPTT)[1, 2] and Real Time Recurrent Learning (RTRL)[3] can perform weight updates based on the entire history of the network's states. Therefore it seems feasible that they could process any length of time series. But in practice, these algorithms share a common weakness: their backpropagated errors either explode or decay in time, preventing them from learning dependencies of more than a few timesteps in length: [4].

The difficulties indicated above help to explain why Hidden Markov Models (HMMs), rather than neural nets, have become the core technology in speech recognition [5]. At first sight this is surprising, since their central premise is that the future behaviour of the system depends only on its current state (for example, that the probability of a phoneme depends only on which phoneme was before it). Moreover, they assume that observations (e.g. frames of speech) are statistically independent, which makes it difficult to model such effects as coarticulation (the blurring together of adjacent speech sounds). In fact, handling coarticulation and other contextual effects has been the most effective use of RNNs in the HMM framework. Work by Bourlard [6] and Robinson [7] showed that using RNNs to estimate output probabilities for HMMs (the *hybrid approach*) gave substantially improved performance.

However, the hybrid approach represents an ad hoc combination of top-down linguistic modelling and bottom-up acoustic modelling. We feel that a consistent, bottom-up approach could be made to speech recognition by an RNN that could overcome such problems as long time dependencies and temporal distortions. In the following, we aim to test this claim with the Long Short Term Memory (LSTM) architecture first described in [8] and later extended in [9].

3 The LSTM Architecture

LSTM is an RNN that uses self-connected unbounded internal *memory cells* protected by nonlinear multiplicative gates. Error is back-propagated through the network in such a way that exponential decay is avoided. The unbounded (i.e.

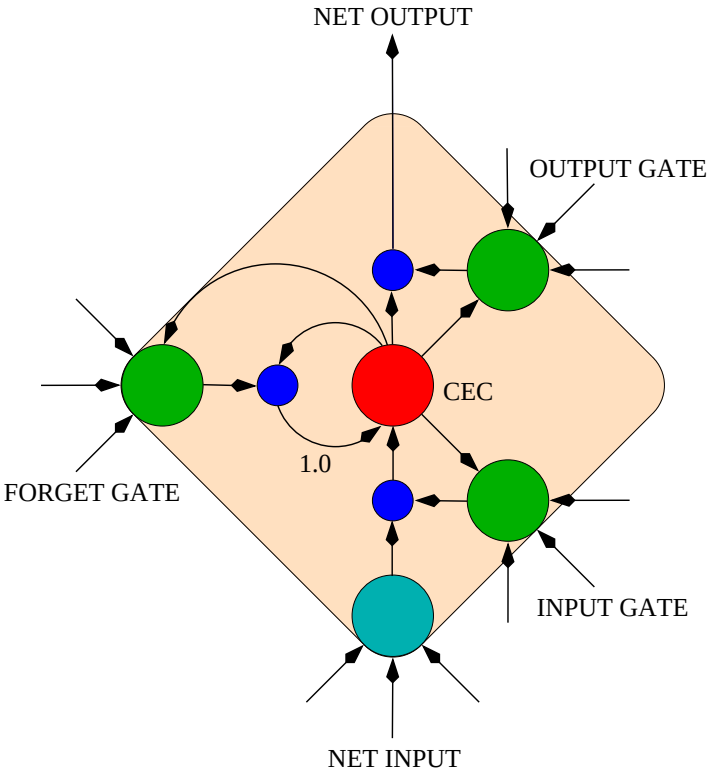

Fig. 1. LSTM memory block with one cell. The internal state of the cell is maintained with a recurrent connection of weight 1.0. The three gates collect activations from both inside and outside the block, and control the cell via multiplicative units (small circles). The input and output gates effectively scale the input and output of the cell while the forget gate scales the internal state—for example by resetting it to 0 (making it forget).

unsquashed) cells are used by the network to store information over long time durations. The gates are used to aid in controlling the flow of information through the internal states. The cells are organized into *memory blocks*, each having an *input gate* that allows a block to selectively ignore incoming activations, an *output gate* that allows a block to selectively take itself offline, shielding it from error, and a *forget gate* that allows cells to selectively empty their memory contents. Note that each memory block can contain several memory cells. See Figure 1. Each gate has its own activation in the range $[0, 1]$. By using gradient descent to optimise weighted connections into gates as well as cells, an LSTM network can learn to control information flow. LSTM's learning algorithm is local in space and time with computational complexity per timestep and weight of $O(1)$ for standard topologies (see Section 6.1 for details). This locality, in contrast with training algorithms such as Real Time Recurrent Learning and Back Propagation Through Time, makes LSTM more biologically plausible than most RNN architectures. Indeed, a recent report by O'Reilly [10] describes a closely related model of working memory in the basal ganglia and prefrontal cortex.

3.1 Why use LSTM for speech recognition?

As mentioned in Section 2, it is essential for an RNN used in speech recognition to be able to bridge long time lags, and adapt to time warped data. These are two areas in which LSTM has outperformed other RNN's. That LSTM can deal with long time lags has been demonstrated in experiments such as [8, 11], while its utility with time-warping is clear from its success in learning context free languages [11], and in generating music [12]. In both cases, its advantage comes from the fact that because its central timing mechanism is not (as for most RNNs) a decaying flow of recurrent activation. Instead, its memory cells act as a set of independent counters. These cells (along with the gates used to open, close and reset them) allow LSTM to extract and store information at a very wide range of timescales.

However, HMMs, rather than RNNs, are the standard tool for speech recognition, and the question we must ask is why use LSTM instead of HMMs? The answer is that statistical models like HMMs tend to be less general and less robust than RNNs, as also less neurologically plausible. For example, the parameters and language models used in HMMs are tuned towards particular datasets, and the choice of acoustic model they use is dependent on the size of the corpus. Also HMMs are very sensitive to channel errors, and coding algorithms are needed to clean up the data before they see it. LSTM on the other hand, is a general purpose algorithm for extracting statistical regularities from noisy time series. Unlike HMMs, it doesn't rely on the manual introduction of linguistic and acoustic models, but can learn its own internal models directly from the data. And although (like all neural nets) it does have free parameters, such as learning rate and layer size, we demonstrate below that these do not need to be adjusted for particular corpora.

4 Experiments

Two datasets were used in the following experiments. The first was a subset (containing only the single digit utterances) of the TIDIGITS corpus, collected by Texas Instruments from more than 300 adults and children. The second was a set of 500 randomly selected spoken digits from the TI46 corpus. The task on the TI46 data was to correctly identify ten digits "zero", "one", "two",...,"nine", while on the TIDIGITS data there was the additional digit "oh". Unlike some experiments found in the literature, we did not separate the adult speakers from the children in TIDIGITS. In both cases the audio files were preprocessed with mel-frequency cepstrum coefficient (MFCC) analysis, using the HTK toolkit [13], with the following parameters: 12 cepstral coefficients, 1 energy coefficient, and 13 first derivatives, giving 26 coefficients in total. The frame size was 15 ms and the input window was 25 ms.

4.1 Experimental Setup

We used a neural net with a mix of LSTM and sigmoidal units (with a range of $[0, 1]$). The net had 26 inputs (one for each MFCC coefficient) and 11 (10) sigmoidal output units for TIDIGITS (TI46) - one for each digit. The classification was based on the most active output layer at the end of an input sequence (i.e. after each spoken digit). A cross-entropy objective function was used, and the output layer had a gain of 3. The network also had two hidden layers. The first of these was an extended LSTM layer with forget gates and peephole connections (see section 3 for details). The layer contained 20 memory blocks, each with two cells, containing 100 cells in total. The squashing function was logistic with range $[-2, 2]$, and the activation functions of the gates were logistic in range $[0, 1]$. The bias weights to the LSTM forget (input and output) gates were initialised blockwise with positive (negative) values of $+0.5$ (-0.5) for the first block, $+1.5$ (-1.5) for the second block and so on. The second hidden layer consisted of 10 sigmoidal units. In total there were 121 units (excluding inputs) and 7791 weights.

Most of these parameters are standard and have been used in all our LSTM experiments. The biasing of the LSTM gates (as used in [9]) ensures that the input and output gates are initially open and the forget gate is initially closed. The staggering in this bias causes the blocks to become active sequence processors one after another, which seems to aid in the subdivision of tasks between separate blocks. A smaller LSTM net, with only 10 memory blocks, was found to perform less well on this task; experiments with even larger nets failed to give any further improvement. The use of a squashing function with range $[-2, 2]$ for the cells is also standard, and is helpful in that it allows the stored cell values to step down as well as up. The inclusion of the extra sigmoidal layer and the gain at the output layer facilitate classification tasks, as they tend to sharpen the network outputs towards one or zero. Cross entropy was used as the objective function because of its known affinity for identifying mutually exclusive classes [14].

The connection scheme was as follows: all units were biased, except for the input units. The input layer was fully connected to the LSTM layer. The LSTM layer was fully connected to itself, the hidden sigmoidal layer, and the output layer (note that the LSTM layer had only outputs from its cells, and not from its gates). The second hidden layer was fully connected to the output layer.

The learning rate was 10^{-6} and online learning was used, with weight updates at every timestep. The momentum algorithm from [15] was used with a value of 0.9, and the network activations were reset to zero after each pattern presentation. These parameters were experimentally determined, although we have not deviated from them significantly in any of our LSTM speech experiments. Errors were fed back on every timestep, encouraging the net to make correct decisions as early as possible (a useful property for real time applications). Gaussian noise was injected into the training data to prevent overfitting.

4.2 Results

With the above setup, we achieved an error rate of 1.1% on the TIDIGITS data. The same network, with unaltered weights, achieved an initial error rate of 42% on the TI46 data, dropping to 0.5% after only six minutes of additional training. Restarting with randomised weights, we achieved an error rate of 2% on the TI46 data. Note that our results were actually better with the net previously trained on a different corpus, suggesting that an LSTM based recogniser could be well suited to incremental learning.

A recent paper [16] gives an error rate of 2.1% on the TIDIGITS corpus with a state-of-the-art HMM based recogniser, when the data was coded to make it robust to channel errors. With two other, less robust, coding schemes, errors of 0.7% and 0.4% were recorded. On the TI46 database, an error rate of 0.5% was recorded with a similar system. Although the best of these results are better than those achieved with LSTM, it should be pointed out that the HMM systems were heavily tuned towards individual databases, that they have to suffer a drop in accuracy to become more robust to noise (to which neural nets are always robust), and that 20 years of research and development, incorporating knowledge from linguists and statisticians, have gone into achieving these figures. For LSTM on the other hand, no specific adjustments have been made to improve its performance on speech. Furthermore, the kind of incremental learning demonstrated above would be impossible with an HMM-based system, which would require complete retraining to switch from one corpus to another.

5 Conclusions and Future Work

The failure of traditional artificial RNNs in speech recognition is at least partly due to their problems with long time lags between relevant input signals. However, these problems that have been overcome by the LSTM, which is also more biologically plausible than traditional RNNs such as BPTT and RTRL, as its

learning algorithm is entirely local in space and time, and even related to a model of the prefrontal cortex.

Previous work on LSTM has focused on artificially generated sequence processing tasks. Here, for the first time, we applied it to the task of spoken digit recognition, using data from the TIDIGITS and TI46 speech corpora. With no tailoring of the basic LSTM setup towards speech recognition, we obtained impressive results, already comparable to those recorded with specially constructed HMM-based systems that have some 20 years of research and development behind them.

We are confident that LSTM has the potential to perform well on more complex continuous speech recognition tasks. We intend to extend our research by applying LSTM to automatic syllable and phone segmentation - one of the most challenging problems in text to speech applications - and by using articulatory features for word and phone level identification.

6 Appendix: The LSTM Algorithm

In the following we give detailed pseudocode for a single training step in the learning algorithm of extended LSTM (LSTM with forget gates and peephole connections). See [9] for more information on extended LSTM.

As with other Backpropagation algorithms, each training step contains two phases: the forward pass and the backward pass. In the case of LSTM the backward pass (where an error signal at the output layer is propagated backwards through the net) is only carried out if error is injected (i.e. if a target is presented). However, the calculation of the partial derivatives required for the weight updates must be carried out on every timestep, regardless of pattern presentation (hence this step is included in the forward pass). Note also that weight updates can be executed at any time - e.g. after every time step (online learning) or after after every complete pass through the training set (batch learning). All the experiments in this paper used online learning.

6.1 Pseudocode

Notation j indexes memory blocks and v indexes memory cells in block j. c identifies a particular cell and S_c identifies the internal state of that cell; so c_j^v is cell v in block j and $S_{c_j^v}$ is its internal state. The memory cell squashing function is denoted g: in our experiments g was a logistic sigmoid with range $[-2, 2]$. The synaptic weight from unit a to unit b is denoted W_{ba} and previous activations (from one timestep ago) are marked $\hat{}$. dS is the set of all partial derivatives used for weight updates. net_a denotes network (unsquashed) inputs to unit a. $\mathrm{in_j}$, $\mathrm{out_j}$, and φ_j are respectively the input gate, output gate and forget gate in block j; likewise f_{in_j}, f_{out_j} and f_{φ_j} are the squashing functions of these gates, all of which (for all blocks) are logistic sigmoids with range $[0, 1]$ for the purposes of this paper. The activation of a generic unit a is denoted y^a. During the weight updates, Δw_{ba} is the change applied to the weight from unit a to unit b. α is the network learning rate.

Initialise network:

states: $s_{c_j^v}=\hat{s}_{c_j^v}=0$**; partial derivatives:** $dS=0$**; activations:** $y=\hat{y}=0$**;**

Forward pass:

input units: $y =$ `current external input;`

begin new timestep: activations: $\hat{y}=y$**; cell states:** $\hat{s}_{c_j^v}=s_{c_j^v}$**;**

loop over memory blocks, indexed j {

input gates

$z_{\text{in}_j} = \sum_m w_{\text{in}_j m}\ \hat{y}_m + \sum_{v=1}^{S_j} w_{\text{in}_j c_j^v}\ \hat{s}_{c_j^v}; \quad y_{\text{in}_j} = f_{\text{in}_j}(z_{\text{in}_j});$

forget gates

$z_{\varphi_j} = \sum_m w_{\varphi_j m}\ \hat{y}_m + \sum_{v=1}^{S_j} w_{\varphi_j c_j^v}\ \hat{s}_{c_j^v}; \quad y_{\varphi_j} = f_{\varphi_j}(z_{\varphi_j});$

cell states

loop over the S_j cells in block j, indexed v {

$z_{c_j^v} = \sum_m w_{c_j^v m}\ \hat{y}_m; \quad s_{c_j^v} = y_{\varphi_j}\ \hat{s}_{c_j^v} + y_{\text{in}_j}\ g(z_{c_j^v});$ }

output gate activation

$z_{\text{out}_j} = \sum_m w_{\text{out}_j m}\ \hat{y}_m + \sum_{v=1}^{S_j} w_{\text{out}_j c_j^v}\ s_{c_j^v}; \quad y_{\text{out}_j} = f_{\text{out}_j}(z_{\text{out}_j});$

cell outputs

loop over the S_j cells in block j, indexed v { $y_{c_j^v} = y_{\text{out}_j}\ s_{c_j^v};$ }

} end loop over memory blocks

output units : $z_k = \sum_m w_{km}\ y_m$**;** $\quad y_k = f_k(z_k)$**;**

partial derivatives:

loop over memory blocks, indexed j {

loop over the S_j cells in block j, indexed v {

cells , $(dS_{cm}^{jv} := \frac{\partial s_{c_j^v}}{\partial w_{c_j^v m}})$**:**

$dS_{cm}^{jv} = dS_{cm}^{jv}\ y_{\varphi_j} + g'(z_{c_j^v})\ y_{\text{in}_j}\ \hat{y}_m;$

input gates , $(dS_{\text{in},m}^{jv} := \frac{\partial s_{c_j^v}}{\partial w_{\text{in}_j m}}$ **,** $dS_{\text{in},c_j^{v'}}^{jv} := \frac{\partial s_{c_j^v}}{\partial w_{\text{in}_j c_j^{v'}}})$**:**

$dS_{\text{in},m}^{jv} = dS_{\text{in},m}^{jv}\ y_{\varphi_j} + g(z_{c_j^v})\ f'_{\text{in}_j}(z_{\text{in}_j})\ \hat{y}_m;$

loop over peephole connections from all cells, indexed v' {

$dS_{\text{in},c_j^{v'}}^{jv} = dS_{\text{in},c_j^{v'}}^{jv}\ y_{\varphi_j} + g(z_{c_j^v})\ f'_{\text{in}_j}(z_{\text{in}_j})\ \hat{s}_c^{v'};$ }

forget gates , $(dS_{\varphi m}^{jv} := \frac{\partial s_{c_j^v}}{\partial w_{\varphi_j m}}$ **,** $dS_{\varphi c_j^{v'}}^{jv} := \frac{\partial s_{c_j^v}}{\partial w_{\varphi_j c_j^{v'}}})$**:**

$dS_{\varphi m}^{jv} = dS_{\varphi m}^{jv}\ y_{\varphi_j} + \hat{s}_{c_j^v}\ f'_{\varphi_j}(z_{\varphi_j})\ \hat{y}_m;$

loop over peephole connections from all cells, indexed v' {

$dS_{\varphi c_j^{v'}}^{jv} = dS_{\varphi c_j^{v'}}^{jv}\ y_{\varphi_j} + \hat{s}_{c_j^v}\ f'_{\varphi_j}(z_{\varphi_j})\ \hat{s}_c^{v'};$ }

} } end loops over cells and memory blocks

Backward pass **(if error injected):**

error injected into ouput units (indexed k): $e_k = t_k - y_k$

δs of output units : $\delta_k = f'_k(z_k)\ e_k$

δs of non LSTM units connected to outputs : $\delta_i = f'_i(z_i)\,(\sum_k w_{ki}\ \delta_k)$
loop over memory blocks, indexed j {

δs of output gates :

$\delta_{\text{out}_j} = f'_{\text{out}_j}(z_{\text{out}_j})\ \left(\sum_{v=1}^{S_j}\ s_{c_j^v} \sum_k w_{kc_j^v}\ \delta_k\right)$;

internal state error :
loop over the S_j cells in block j, indexed v {

$e_{s_{c_j^v}} = y_{\text{out}_j}\ \left(\sum_k w_{kc_j^v}\ \delta_k\right)$; }

} end loop over memory blocks

weight updates:

output units : $\Delta w_{km} = \alpha\ \delta_k\ y_m$**;**

loop over memory blocks, indexed j {

output gates :
$\Delta w_{\text{out},m} = \alpha\ \delta_{\text{out}}\ \hat{y}_m$; $\Delta w_{\text{out},c_j^v} = \alpha\ \delta_{\text{out}}\ s_{c_j^v}$;

input gates :
$\Delta w_{\text{in},m} = \alpha \sum_{v=1}^{S_j}\ e_{s_{c_j^v}}\ dS^{jv}_{\text{in},m}$;

loop over peephole connections from all cells, indexed v' {

$\Delta w_{\text{in},c_j^{v'}} = \alpha \sum_{v=1}^{S_j}\ e_{s_{c_j^v}}\ dS^{jv}_{\text{in},c_j^{v'}}$; }

forget gates :
$\Delta w_{\varphi m} = \alpha \sum_{v=1}^{S_j}\ e_{s_{c_j^v}}\ dS^{jv}_{\varphi m}$;

loop over peephole connections from all cells, indexed v' {

$\Delta w_{\varphi c_j^{v'}} = \alpha \sum_{v=1}^{S_j}\ e_{s_{c_j^v}}\ dS^{jv}_{\varphi c_j^{v'}}$; }

cells :
loop over the S_j cells in block j, indexed v {

$\Delta w_{c_j^v m} = \alpha\ e_{s_{c_j^v}}\ dS^{jv}_{cm}$; };

} end loop over memory blocks

References

1. Williams, R.J., Zipser, D.: Gradient-based learning algorithms for recurrent networks and their computational complexity. In Chauvin, Y., Rumelhart, D.E., eds.: Back-propagation: Theory, Architectures and Applications. Lawrence Erlbaum Publishers, Hillsdale, N.J. (1995) 433–486
2. Werbos, P.J.: Generalization of backpropagation with application to a recurrent gas market model. Neural Networks **1** (1988)

3. Robinson, A.J., Fallside, F.: The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department (1987)
4. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer, S.C., Kolen, J.F., eds.: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
5. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. In: Proc. IEEE. Volume 77., IEEE (1989) 257–286
6. Bourlard, H., Morgan, N.: Connnectionist Speech Recognition: A Hybrid Approach. Kluwer Academic Publishers (1994)
7. Robinson, A.J.: An application of recurrent nets to phone probability estimation. IEEE Transactions on Neural Networks **5** (1994) 298–305
8. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9** (1997) 1735–1780
9. Gers, F.: Long Short-Term Memory in Recurrent Neural Networks. PhD thesis (2001)
10. O'Reilly, R.: Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. Technical Report ICS-03-03, ICS (2003)
11. Gers, F.A., Schmidhuber, J.: LSTM recurrent networks learn simple context free and context sensitive languages. IEEE Transactions on Neural Networks **12** (2001) 1333–1340
12. Eck, D., Schmidhuber, J.: Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In Bourlard, H., ed.: Neural Networks for Signal Processing XII, Proceedings of the 2002 IEEE Workshop, New York, IEEE (2002) 747–756
13. Young, S.: The HTK Book. Cambridge University (1995/1996)
14. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Inc. (1995)
15. Plaut, D.C., Nowlan, S.J., Hinton, G.E.: Experiments on learning back propagation. Technical Report CMU–CS–86–126, Carnegie–Mellon University, Pittsburgh, PA (1986)
16. Zheng, F., Picone, J.: Robust low perplexity voice interfaces. Technical report, MITRE Corporation (2001)