LSTM for Classifying Speech with a Sparse TIDIGITS Dataset

Gabriel Carrizo, Carl-Johan Larsson
carrizo,cjlars@kth.se

Abstract

A sparse dataset of TIDIGTs single digit utterances has been used to investigate the hyperparameters relative to performance using a LSTM-RNN. The dataset contains speakers uttering the digits 0-9. The models have been trained using a K420 GPU and no model is trained for more than one hour. The hyperparameters with the most impact on the test set performance was the amount of hidden states and learning rate, the best performance achieved was 98.63%. The main objective of the paper is to present results for future researchers on speech recognition using an LSTM-RNN on a sparse dataset to give understanding of the hyperparameter impact.

1 Introduction

1.1 Neural Networks and Sequential Data

Classifying sequential data, such as speech, written text or any time dependent signal has recently made progress with deep neural networks (DNNs). The traditional DNN is a feed forward neural net which uses a weight matrix to construct a hyper plane which it then uses to separate data into specific classes. Regular feed forward DNNs have no way of processing contextual or sequential data but classify the data in its input frame as if it were independent in time. A work around method to this is to train the network with data in chunks (for example: word for word or sentence by sentence) and doing the same when classifying. However, this method would still ignore the temporal dependence within the input.

When reading a sentence the human brain sequentially processes input information while simultaneously taking previous information into account in order to build context and understanding. Standard recurrent neural networks (RNN) have attempted to mimic this process by utilizing a network which builds context by cycling input data and storing its internal state for all previous inputs. However, memorizing all previous input information can be costly in terms of memory and effort which is why the human brain will latch on to only the information that is most necessary to build context. In contrast to RNNs, long short term memory (LSTM) select which information to keep and also judges when this information is no longer relevant for present context.

1.2 Long Short Term Memory - LSTM

The core of the LSTM unit is a cell that has three gates that regulate the flow and influence of certain data throughout the cell. The three gates are as follows:

- **The forget gate** is the output of a sigmoid function (essentially a differentiable indicator function which can output the values 0 or 1) which decides how much of previous data is still relevant to cell state or if the data is relevant at all. If the forget gate is 0 the cell state is reset.
- **The input gate** layer is also a sigmoid function and it decides if information should be added to the cell state and if so, how much of it.

- **The output gate** (also a sigmoid) decides how much of the cell state should be passed to the cells hidden layer which is then passed on to the next iteration of the LSTM cell. The hidden layer can also be passed on to an output layer or other LSTM cells.

See Figure 1 below for a visualisation of the LSTM cell.

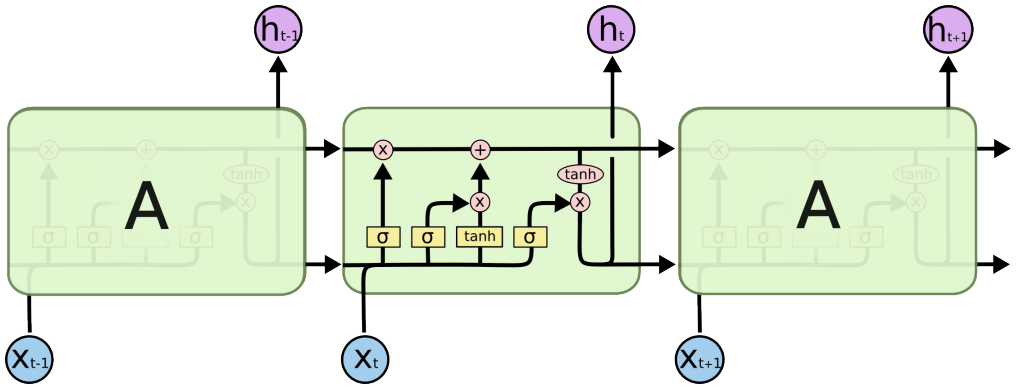


Figure 1: Generic schematic of an LSTM cell with the three gates annotated as σ. Left to right: Forget, input and output gates. The pink circles and ellipse represent point wise operations and joined arrows represent concatenations.

Rather than sequentially updating the cell state with all input and as such avoids *vanishing and exploding gradients*, the LSTM RNN selectively decides when to update its weights. Since RNNs use the networks' gradients to learn tasks, this is a problem that has lead to basic RNNs not being favorable in classification of sequential data.[1]

For more in depth information on LSTMs, consult Cristopher Olah's blog post regarding LSTMs [2] or Hasim Sak et al's paper about LSTMs [3]

1.3 Why LSTM?

In the recent past Hidden Markov Models (HMMs) in combination with DNNs have dominated speech recognition. Recently however, Graves et al have mananged to show that LSTM networks outperform HMM-DNNs with large vocabulary datasets [3].

1.4 Previous Work

Graves et al [4] with Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Switzerland, have previously performed a simillar study with great success on the TIDIGITS dataset. Furthermore, Wang et al [5] at National Institute of Informatics, Japan, has done a thourough evaluation comparing HMM, DNN and LSTM-RNN. LSTM-RNN is shown to achieve the same performance as DNN in a 10th of the time, and the same result as a HMM in one percent of the time. The LSTM-RNN was trained for ten hours.

1.5 Project Aim

The aim of this project has primarily been an extended reproduction of Graves et al's approach [4]. In addition, it has also been to evaluate the effects of hyperparameters on a LSTM network by analyzing the accuracy, cost and confusion matrix generated by different network architectures.

2 Method

2.1 Pre-Processing of the Data

In general, LSTMs have been utilized for large vocabularies. For example, Hasim Sak et al have trained LSTMs on 3 million utterances for Google English Voice Search Task [3]. For the purpose

of this project an LSTM network will be trained on a smaller dataset, TIDIGITS [6]. TIDIGITS is a dataset recorded by Texas Instruments which consists of approximately 300 speakers each pronouncing 77 digit sequences consisting of digits in the range 0-9 (note that the pronunciations 'oh' and 'zero' for 0 both appear in the dataset). For this project the dataset will be restricted to single digits from adult male and female speakers.

The pure form of the data is in .wav-format, which is a linear pulse code modulation format, sampled with a sample rate of 20,000 Hz. The first step of pre-processing the data was converting the .wav format to mel-scale frequency cepstral coefficients (MFCC) containing an energy coefficient, delta and delta-delta coefficients. Deltas are the trajectories of the MFCC coefficients while delta-deltas are the derivatives of deltas. In this report this format may be denoted as MFCC_0_D_A. See Practical Cryptography's blog post [7] for more information on MFCC. Since the MFCCs were computed using 12 cepstral coefficients and one energy coefficient together with the first and second order derivatives of the coefficients, this results in a total of 39 coefficients. The pre-processing was performed with the Hidden Markov Model Toolkit (HTK)[8].

2.2 Neural Network Model

Google's machine intelligence library *Tensorflow* (version 1.0 for Python 2.7) was utilized for the neural network RNN-LSTM model. The model used was a *multi RNN cell* with three *basic LSTM cells* with the following network-related Tensorflow methods:

- tensorflow.contrib.rnn.DropoutWrapper(...)
- tensorflow.contrib.rnn.BasicLSTMCell(...)
- tensorflow.contrib.rnn.MultiRNNCell(...)
- tensorflow.nn.dynamic_rnn(...)

The network was created following a Tensorflow LSTM example GitHub repository by user *aymeric-damien*[9]

2.3 Hyperparameter Optimization

According to Klaus Greff et al's aptly named paper *LSTM - A Search Space Odyssey* [10], learning rate and the hidden layer size are the most influential hyperparameters of an LSTM network, so this project focuses on evaluating the effects of altering the two of them. All networks have been trained using a drop-out rate of 0.75, which means that for every iteration 25% of the weights are ignored. Drop-out is a powerful tool for regularization and will avoid over-fitting by ignoring local patterns in the data. Furthermore, the network is trained for 400,000 iterations where one iteration constitutes one sample being fed through the network one time. The dynamic hyperparameters, on which a grid search was performed, can be found in Table 1.

Table 1: Training paramters for grid search

Learning rate	1e-1	1e-2	1e-3	1e-4	1e-5
Hidden layers	100	150	200	250	300

After running the grid search the best step size was chosen for further evaluation of number of hidden states. Networks were then trained and evaluated for the following number of hidden states:

Table 2: Hidden states evaluated using 1e-3 stepsize

Hidden states	10	50	500

2.4 Confusion Matrix

To create the confusion matrices each of the digits in the test set were fed to a trained LSTM classifier sequentially and for each instance of a digit classification an entry in an 11x11 (0,0,1,2...,9 x 0,0,1,2,...,9) matrix was incremented. For example, if the network was fed data where a person

pronounces the number '1' and classifies it as number '3', the entry [3,5] (if the matrix is indexed from 0) is incremented by 1. This procedure is repeated for the whole dataset.

Well-trained LSTMs perform too well and generate confusion matrices that are nearly impossible to draw any conclusions from. So in order to evaluate whether there were any digits that were more difficult to classify than others, all the matrices generated for step size 1e-3 and 1e-4 were super positioned. To improve the visualization of the matrix, the natural logarithm of the plots were observed.

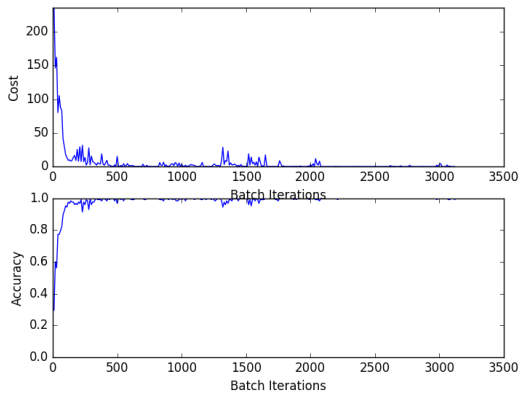
3 Results

3.1 Hyperparameter Tuning

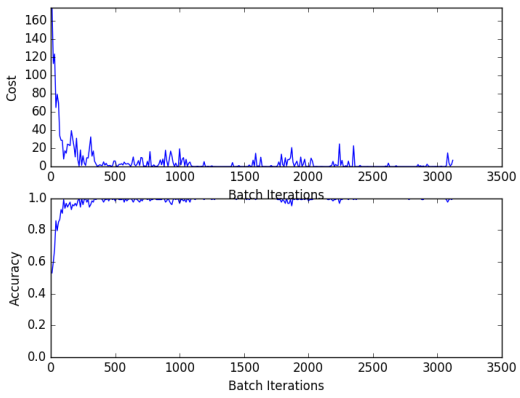
The networks mentioned in section 2.3 performed with the following accuracies:

Table 3: Hyperparameters

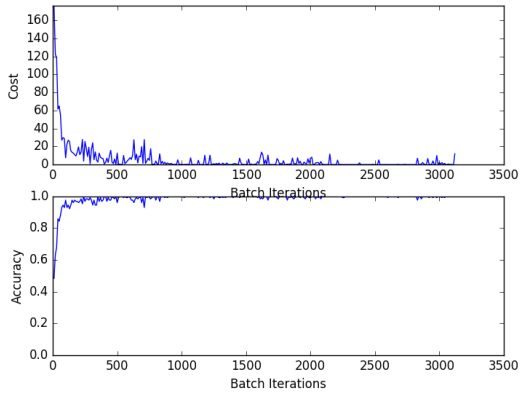
Learning rate	Hidden states	Accuracy
1e-3	100	98.23%
	150	97.05%
	200	96.01%
	250	98.47%
	300	98.63%



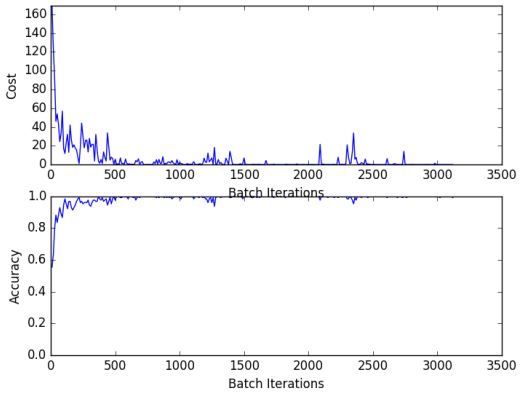
a) Ċost and Accuracy with 100 hidden states



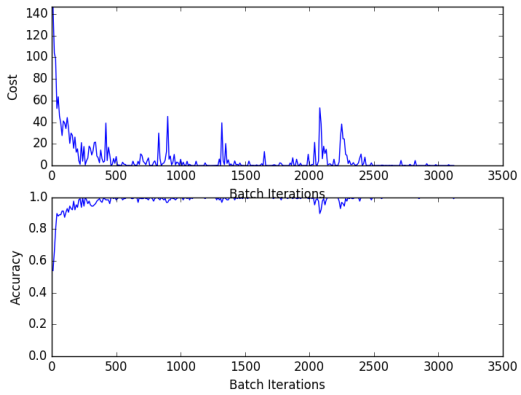
b) Ċost and Accuracy with 150 hidden states



c) Ċost and Accuracy with 200 hidden states



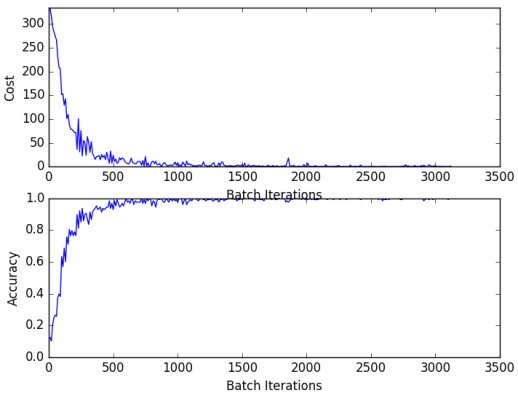
d) Ċost and Accuracy with 250 hidden states



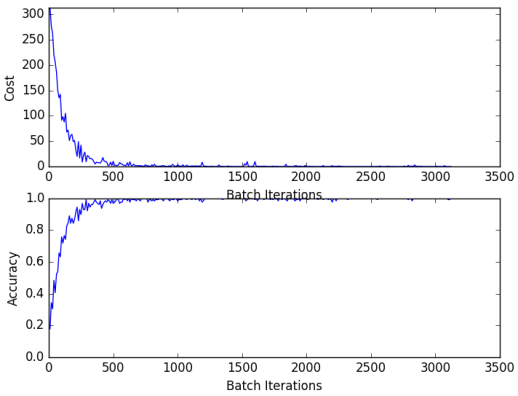
e) Ċost and Accuracy with 300 hidden states

Table 4: Hyperparameters

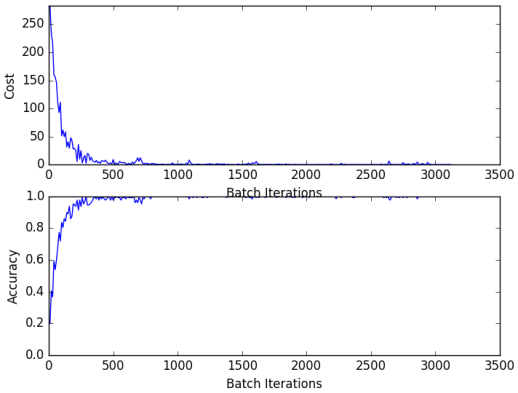
Learning rate	Hidden states	Accuracy
1e-4	100	95.89%
	150	96.74%
	200	96.82%
	250	97.06%
	300	96.78%



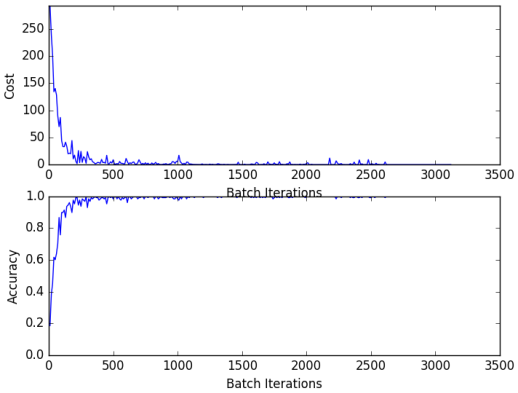
a) Ċost and Accuracy with 100 hidden states



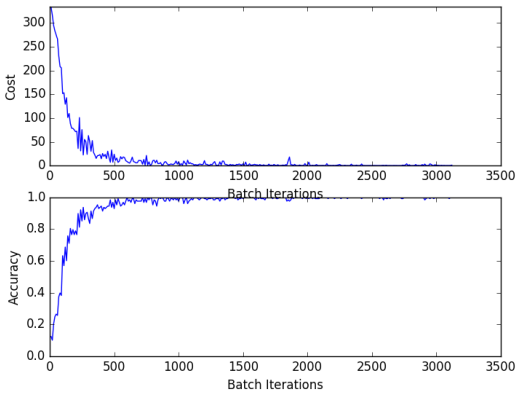
b) Ċost and Accuracy with 150 hidden states



c) Ċost and Accuracy with 200 hidden states



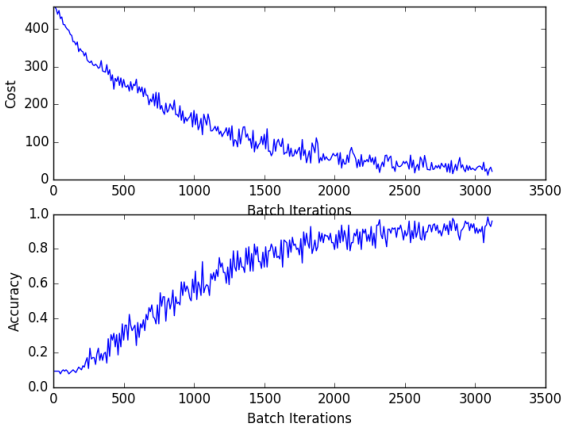
d) Ċost and Accuracy with 250 hidden states



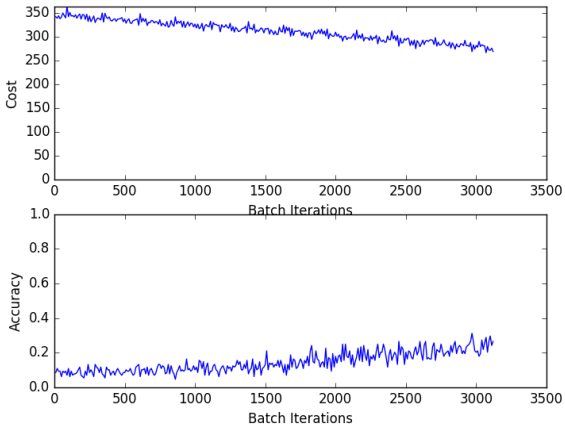
e) Ċost and Accuracy with 300 hidden states

Table 5: Hyperparametrs

Learning rate	Hidden states	Accuracy
1e-5	100	89.17%
1e-6	100	23.69



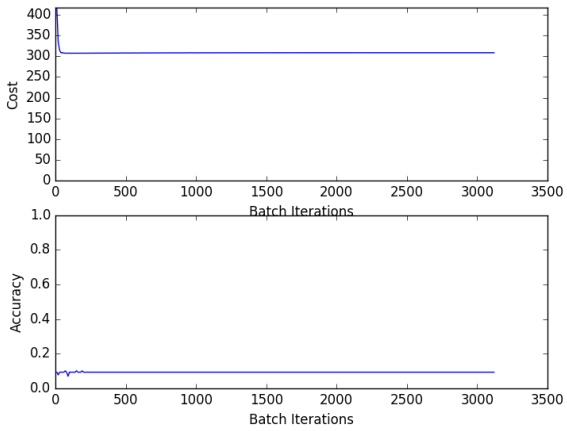
a) Ċost and Accuracy with step size = 1e-5



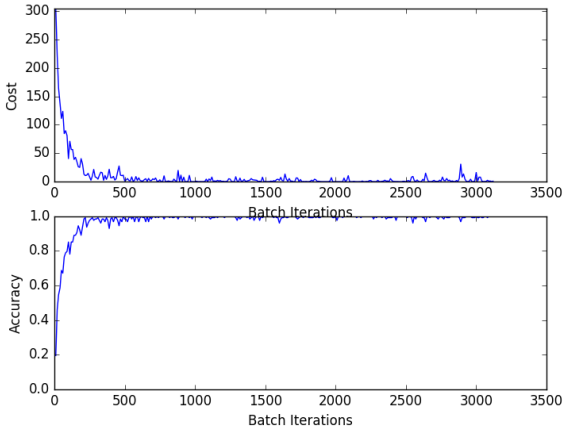
b) Ċost and Accuracy with step size = 1e-6

Table 6: Hyperparameters and Accuracy

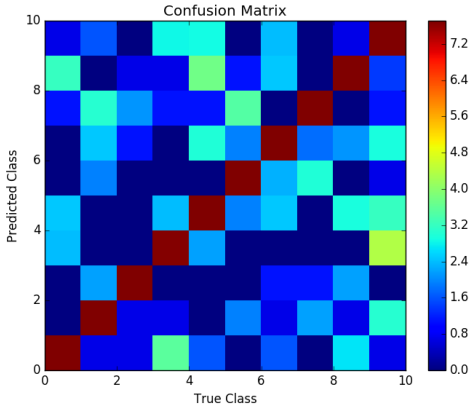
Learning rate	Hidden states	Accuracy
1e-1	100	9.09%
1e-2	100	96.86%



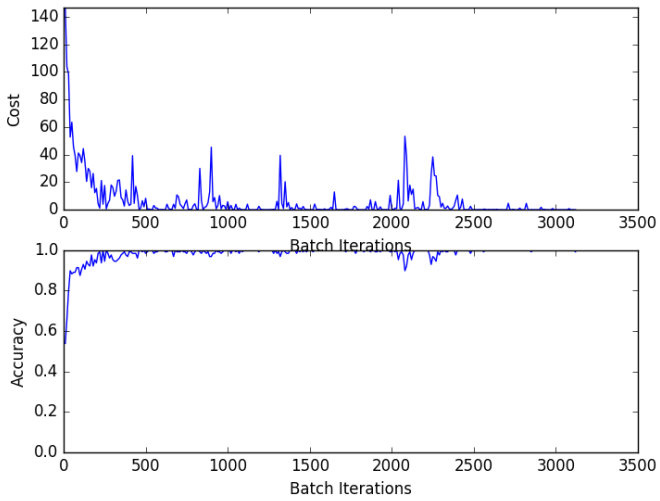
a) Ċost and Accuracy with step size = 1e-1



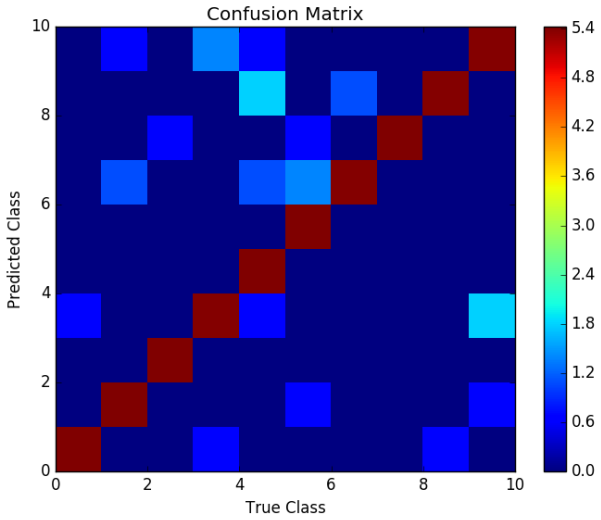
b) Ċost and Accuracy with step size = 1e-2



a) Ċombined natural logarithmic confusion matrix of all networks with step size 1e-3 and 1e-4 presented in section 2.3



a) Ċost and accuracy over batch iterations



b) Ṅatural logarithmic confusion matrix

Figure: The best performing network uses a step size of 1e-3 with 300 hidden states and has an accuracy of 98.63% on the test set

4 Discussion and Conclusions

4.1 Dataset

Since we only used the single digit sequences in the dataset we limited our dataset greatly. A future investigation that would be interesting is to pre-process the data in a more general way which would allow for sequence training. In theory, training a complex network with sparse amount of data will make it prone to overfitting. As mentioned below this influenced our choice of heavy regularization.

4.2 Grid Search

For the grid search it was clear that for this case a learning rate of 1e-5 to 1e-4 all performed acceptably well. We sadly did not have enough time with a GPU to run more iterations on a few of the step sizes which is why some of the worse performing step sizes were not included in the full grid search. for example, 1e-1 (3.1) did not even perform better than random and 1e-6 (3.1 never converged.

For better results we would also have wished to perform a random search but did not feel we had enough time to run enough search pairings to gamble on a random search. In our minds there was a scenario where we would draw bad pairings and reach no conclusion or a false one. At least with our approach we could do the search in smaller steps and narrow down the search fields manually. Another thing we would have liked to try is an adaptive step size but we leave that to future attempts.

We would also have used a lower number of iterations because the models converged fast enough and it would have saved us time. We also propose using a validation error to cancel the training when the network has converged.

4.3 Confusion Matrix

In the confusion matrix 3.1 we can see that although the models perform very well (>96%) it is most prone to confuse the digits two and nine and in general performs the worst on classifying the digit nine. Since the dataset is sparse this can be a result of the dataset containing very significant differences in the way the speakers are pronouncing nine. Therefore, the network is not able to generalize and learn the pattern. Another approach we would like to have tried is training a few networks over fewer iterations in order to generate networks that perform worse than our networks. Super positioned confusion matrices from these attempts could give insight as to which digits are more difficult for the networks to learn.

4.4 Learning Influence of Hyperparameters

As the background research suggested the hyperparameters that will have the biggest influence on the performance of the network are the amount of hidden states and the step size. The best results were achieved using hidden states in the magnitude of 10^2 and step size 10^{-3}, the final parameter setup displayed in 3.1 ensured a fast convergence rate with a test set accuracy of 98.63%. While learning performance measured in time and computations is vital apart from accuracy, it is of course essential not to overfit the network. Different dropout rates were tried and with a probability of 1 to consider all weights in the forward pass the network was overfit within 200,000 iterations. For lower drop-out rates; 0.25 and 0.5, the network performed worse and thus 0.75 was used for all experiments. Not using any regularization at all when training an LSTM is not recommended.

The amount of hidden states for each LSTM layer is as mentioned one of the most vital features together with the learning rate. An experiment was made, increasing the size from 10 to 500 with steps at 10, 50, 100, 150, 200, 250, 300 and 500. When analyzing the edge cases one can see that using very few hidden states will make the model too simple and it will not be able to capture even very general patterns. Increasing the size to 500 will instead make to model too complex, and even when dropping 25% of all weights the network is overfit and the network performance worse than with e.g. 300 hidden states.

References

[1] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1109/72.279181

[2] Olah. Understanding lstm networks. Accessed: May 2017. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *CoRR*, vol. abs/1402.1128, 2014. [Online]. Available: http://arxiv.org/abs/1402.1128

[4] A. Graves, D. Eck, N. Beringer, and J. Schmidhuber, *Biologically Plausible Speech Recognition with LSTM Neural Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 127–136. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-27835-1_10

[5] X. Wang, S. Takaki, and J. Yamagishi, *A Comparative Study of the Performance of HMM, DNN, and RNN based Speech Synthesis Systems Trained on Very Large Speaker-Dependent Corpora*, 9 2016.

[6] G. D. R. Gary Leonard, "Tidigits ldc93s10," 1993. [Online]. Available: https://catalog.ldc.upenn.edu/ldc93s10

[7] P. Cryptography. Mel frequency cepstral coefficient (mfcc) tutorial. Accessed: May 2017. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#what-is-the-mel-scale

[8] HTK. Hidden markov model toolkit. Accessed: May 2017. [Online]. Available: http://htk.eng.cam.ac.uk/

[9] aymericdamien, "recurrent_network.py," *GitHub repository*, 2017.

[10] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015. [Online]. Available: http://arxiv.org/abs/1503.04069