**COLLEGE OF ENGINEERING**

Preliminary Exam Report

**EEG Classification Using Long Short-Term Memory**

**Recurrent Neural Networks**

Meysam Golmohammadi

meysam@temple.edu

Institute for Signal and Information Processing,

Department of Electrical Engineering,

Temple University

February 2016



**Executive Summary**

In this report, I review two papers on deep recurrent neural networks for speech recognition tasks and one paper concerning application of deep belief networks for the seizure detection task. I also use several other sources to support the material discussed in this report.

The three primary references used in this exam are "Training and Analyzing Deep Recurrent Neural Networks", "Speech Recognition with Deep Recurrent Neural Networks" and "Deep Belief Networks used on High Resolution Multichannel Electroencephalography Data for Seizure Detection". The basic materials in the first papers involve training of deep recurrent neural networks for sequential tasks like speech recognition. The second paper discusses the training of deep long short-term memory networks for speech recognition. The last paper explores the potentials of deep belief network for seizure detection task. As we want to focus just on deep learning in this paper, I did not discuss that how authors of the last paper compared other machine learning algorithms with deep learning approach.

My main goal from this review is planning an approach for seizure detection task using deep long short-term memory networks. Additionally in writing this review, one of my primary intentions is to produce a self-sufficient document on deep recurrent neural networks. To reach this goal, I have added a lot of materials to the main three papers. I believe this document can walk every one with no knowledge in RNN through the basic concepts of RNNs and LSTMs.

To the best of my knowledge RNNs have not been used on any EEG analyzing task including seizure detection. In writing this review, one of my primary intentions is to produce a self-sufficient document that can be used as a reference to implement a seizure detection system using deep LSTMs; so I prefer explaining the methods rather than discussing the results of speech recognition tasks. Moreover, an interested reader can easily start from these and derive more general or application specific algorithms.

Table of Contents

[1 Introduction 1](#_Toc443340216)

[2 Recurrent Neural Networks (RNN) 1](#_Toc443340217)

[2.1 The RNN structure 1](#_Toc443340218)

[2.2 RNN training 2](#_Toc443340219)

[2.3 The Vanishing Gradient Problem 3](#_Toc443340220)

[3 Long Short-Term Memory (LSTM) 4](#_Toc443340221)

[4 Deep Recurrent Neural Networks (DRNN) 5](#_Toc443340222)

[4.1 DRNN structure 5](#_Toc443340223)

[4.2 DRNN training 7](#_Toc443340224)

[4.3 Performance of DRNNs 7](#_Toc443340225)

[5 Deep Bidirectional Long Short-Term Memory (DBLSTM) 8](#_Toc443340226)

[5.1 DBLSTM structure 8](#_Toc443340227)

[5.2 DBLSTM training 8](#_Toc443340228)

[5.3 DBLSTM performance 9](#_Toc443340229)

[6 Seizure detection using deep learning 10](#_Toc443340230)

[7 Conclusion 12](#_Toc443340231)

[8 Reference 13](#_Toc443340232)

# Introduction

In recent years, neural networks composed of multiple layers, which are often termed as Deep Learning, have emerged as a powerful machine learning approach. One type of DL that is popular for sequential data is the recurrent neural network (RNN). When folded out in time, an RNN can be considered as a DL with indefinitely many layers. Unlike feed forward neural networks, the primary function of the layers in an RNN is to use their internal memory to process arbitrary sequences of inputs. However, vanishing gradients has been a great challenge in the learning process. This issue formed the core motivation for the so-called Long Short-Term Memory (LSTM) architecture.

LSTM introduces a new structure called a memory cell which is composed of four main elements: an input gate, a self-recurrent connection, a forget gate and an output gate. The effect of the gates is to allow the cells to store and access information over long periods of time. LSTM RNNs have shown to outperform other state-of-the-art approaches in tasks such as speech and handwriting recognition. In this study, we explore an approach to use LSTMs for automatic classification of electroencephalograms (EEGs). In particular, we investigate an approach to apply LSTMs to the task of seizure detection in a system that we have developed, known as AutoEEG that automatically interprets EEGs.

# Recurrent Neural Networks (RNN)

##  The RNN structure

Generally there are two kinds of neural networks which are feedforward neural networks and recurrent neural networks. A feedforward neural network is an artificial neural network where connections between the units do not form a cycle like Figure 1 In other word in feedforward networks processing of information is piped through the network from input layers to output layers.



Figure 1. An example of feedforward neural network

In contrast, a recurrent neural network (RNN) is an artificial neural network where connections between units form cyclic paths. RNNs are called recurrent because they receive inputs, update the hidden states depended on the previous computations, and make predictions for every element of a sequence. By unrolling an RNN in time, it can be considered as a deep neural network (DNN) with indefinitely many layers (Figure 2).



Figure 2. An RNN unrolled in time

We can consider RNNs as neural networks with memory to keep information of what has been processed so far. RNNs are very powerful dynamic systems for sequence tasks, such as speech recognition or handwritten recognition. They are powerful because they can maintain a state vector that implicitly contains information about the history of all the past elements of the sequence [4].

The RNN depicted in Figure 2 network makes predictions by matrix multiplications as follows:

 (2.1)

 (2.2)

In these equations is the input at time step t. is the hidden state at time step t which is in fact the memory of the network and it is calculated based on the input at the current step and the previous hidden state. is a activation function which transforms the inputs of the layer into its outputs and allows us to fit nonlinear hypotheses. Common choices for are tanh and ReLUs. which is required to initialize the first hidden state is typically set to all zeroes. The output of the network is which is calculated by a nonlinear function of matrix multiplication of V and . In fact this nonlinear function, g, is the activation function for the output layer and usually it is the softmax function. It is simply a way to convert raw scores to probabilities. Unlike feed forward neural networks, which have different parameters at each layer, a RNN shares the same parameters (U, V, W in Eq. 2.1 and Eq. 2.2) across all steps.

## RNN training

There are different approaches for training of RNNs including back-propagation through time (BPTT), real-time recurrent learning (RTRL), and extended Kalman filtering approaches (EKF). In this study we mostly focus on BPTT.

The feedforward neural networks can be trained by backpropagation algorithm. In RNNs, a slightly modified version of this algorithm called Backpropagation through Time (BPTT) is used to train the network. The backpropagation algorithm can be extended to BPTT by unfolding RNN in time and stacking identical copies of the RNN. As the parameters that are supposed to be learnt (U, V and W) are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps.

In RNNs, a common choice for the loss function is the cross-entropy loss which is given by:

In this formula, is the number of training examples, is the prediction of the network and is true label. The parameters U, V and W can be calculated during training by minimizing the total loss on the training data. One popular approach to do this is Stochastic Gradient Descent (SGD). The idea behind SGD is iterate over all our training examples and during each iteration, we update the parameters into a direction that reduces the error. These directions are calculated by the gradients on the loss function respect to U, V and W: . In fact BPTT can be considered as a black box that gets training data as input and returns these gradients.

## The Vanishing Gradient Problem

While RNNs are powerful structures, practically they are hard to train. One of the main reason is “vanishing gradient problem” which explored in depth by Bengio[5][6]. They found that in theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps. This means in practice the range of contextual information that standard RNNs can access are limited. The vanishing gradient problem is illustrated schematically in Figure 3. It is proved that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it pipes through RNN. In fact, it is hard for an RNN to bridge gaps of more than about 10 time steps between relevant input and target events [7].



Figure 3. Vanishing gradient problem in RNNs

Fortunately there are a few approaches to overcome this shortcoming of RNN. For example, W matrix can be initialized properly to combat the vanishing gradient problem or using ReLU instead of tanh or sigmoid activation functionscan reduces the effect of vanishing gradients. However, the most successful solution is to use Long Short-Term Memory (LSTM) which will be introduced in next chapter.

# Long Short-Term Memory (LSTM)

Long Short Term Memory networks are a special kind of RNN architecture introduced by Hochreiter & Schmidhuber [8] that are capable of learning long-term dependencies. LSTM can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities [8]. This is achieved by multiplicative gate units learn to open and close access to the constant error flow. LSTM networks can outperforms alternative RNNs and Hidden Markov Models (HMM) and other sequence learning methods in numerous applications such as speech recognition and handwriting recognition. For example, deep bidirectional LSTM achieved the best known results in automatic speech recognition which is 17.7% phoneme error rate on the classic TIMIT natural speech dataset. Also it won the ICDAR handwriting competition for the best known results in unsegmented connected handwriting recognition.

LSTM network introduces a new structure called a memory cell (Figure 4). Each memory cell contains four main elements: the input gate, forget gate, output gate and a neuron with a self-recurrent. These gates allow the cells to keep and access information over long periods of time.



Figure 4. LSTM memory cell

LSTM calculate the hidden states by a set of equation as follows:

In these equations are related to the input gate, forget gate, output gate and self-recurrent respectively. As they look complicated, the LSTM equations will be explained more in detail. indicates how much of the new information will be let through the memory cell. is responsible for information should be thrown away from memory cell. Every one in means keeping information, while every zero means get rid of this information.decides how much of the information will be passed to expose to the next time step and also to output. is related to a neuron with a self-recurrent which is equal to Eq. 2.1 that we have in the traditional RNN. can be called as the internal memory of the memory cell which is the sum of elementwise multiplication of previous internal memory state by the forget gate, and elementwise multiplication of self-recurrent state with input gate. Finally, is related to the hidden state which can be calculated by elementwise multiplication of the internal memory with the output gate. Additionally the final output can be calculated by Eq. 3.7 which is equal with Eq. 3.2.

Figure 5 presents a schematic of an LSTM unrolled in time to show how LSTM can preserve the gradient information. The input, forget, and output gate activations are respectively displayed below, to the left and above the memory block. For simplicity, the gates are either entirely open (‘O’) or entirely closed (‘—’).



Figure 5. Preserving the gradient information in LSTM

Note that traditional RNNs can be considered a special case of LSTMs. If we set the input gate all ones (passing all of the new information), the forget gate all zeros (forgetting all of the previous memory) and the output gate to all ones (exposing the whole memory), we almost get standard RNN with just a small difference which is the tanh term that squeezes the output a little bit. In fact by training the parameters of the gates, an LSTM learns to handle the long-term dependencies. Also note that there are several variants of the LSTMs architecture and equation which we pick the popular one among them in this study [9].

# Deep Recurrent Neural Networks (DRNN)

## DRNN structure

One potential disadvantage of traditional RNN which explained in part 1 is that, the information only passes through one layer of processing before going to the output. In sequence tasks we usually need to process the information at several time scales. In this part a structure called deep recurrent neural network (DRNN) will be explained which is basically a combination of the concepts of deep neural networks (DNN) with RNNs [1]. In DRNN by stacking RNNs, every layer is an RNN in the hierarchy that receives the hidden state of the previous layer as input. In this way different time scales at different levels, and therefore a temporal hierarchy will be created.



Figure 6. Different variations of the structures of DRNNs

Different variations of the structures of DRNNs are depicted in Figure 6. In this picture connection matrices are shown by arrows. Also input frames, hidden states, and output frames are represented by white, black and grey circles respectively. In left a traditional RNN unrolled in time is shown. In middle we can see a DRNN with three layers folded out in time. In right two variants of DRNN structures are shown. In DRNN-1O the output will be computed just based on the hidden state of the last layer, while in DRNN-AO the output will be the combination of the hidden states of the all layers. In both of these structures the looped arrows represent the recurrent weights.

DRNN-AO has two great advantageous in comparison with DRNN-1O. Firstly, when we are using BPTT for DRNN-AO, the error will propagate from the top layer down the hierarchy without attenuating of magnitude and as a result the system will be trained effectively. Secondly, it gives us the ability to evaluate the impact of each layer in solving the task by leaving out an individual layer’s contribution. In this part character-based language modelling will be explored using both of DRNN-1O and DRNN-AO.

As the distribution of characters in this task is quit nonlinear and covers both short and long term dependencies, it is a well-suited task to study temporal hierarchy of DRNNs. Here the experiments ran on Wikipedia-based text corpus using only stochastic gradient descent (SGD).

We assume that our DRNN has L layers and N neurons per layer and input is a time series x(t) with dimensionality of N­in and y(t) is the output of DRNN. Then we have:

 (4.1)

 (4.2)

As it is mentioned before, two different structures for generating outputs are candidate. The output for DRNN-1O is:

And the output for DRNN-AO is:

Note that how these equation are similar to Eq 2.1 and Eq 2.2.

## DRNN training

In order to train a DRNN, BPTT approach can be applied using stochastic gradient decent for optimizing the parameters as follows:

In Eq. 4.5, is the set of all trainable parameters after j updates, and is the gradient of a cost function with respect to this parameter set, as computed on a randomly sampled part of the training set. T is the number of batches and the learning rate is set at an initial value which decreases linearly with each subsequent parameter update.

For DRNN-1O, an incremental layer-wise method should be used which means training the full network with BPTT and linearly reducing the learning rate to zero before a new layer is added. Then the layers will be added one by one, and after adding a layer the previous output weights will be discarded, and new output weights are initialized connecting from the new top layer. For DRNN-AO, we can test the influence of each layer by setting it to zero. This can assure us that model is efficiently trained.

## Performance of DRNNs

Using the methods explained in 4.2., two DRNNs including both of DRNN-1O and DRNN-AO is trained for Wikipedia character prediction task. The results of these experiments are shown in table 4.1.



Table 4.1. Results of DRNN on Wikipedia Corpus [1]

These results show that DRNN can improve the performance of the network for the task of language modeling significantly and in fact it can gives us a state of the art performance using SGD. It is proved that DRNN have a hierarchy of time-scales in their layer. Additionally, it is demonstrated that in certain cases the DRNNs can have extensive memory of several hundred characters long.

# Deep Bidirectional Long Short-Term Memory (DBLSTM)

## DBLSTM structure

In the previous part the structure of DRNN which was based on the stacking of traditional RNN layers was explored. In this part the structure of deep bidirectional long short-term memory RNNs will be introduced. As it was mentioned before, RNNs are inherently deep in time, since their hidden state is a function of all previous hidden states. By stacking multiple LSTM layers on top of each other, we show that DBLSTMs could also benefit from depth in space.

One of the greatest disadvantageous of RNN and LSTM structures that were introduced in part 1 and 2, is that they are processing information just based on the previous context. In a lot of application including speech recognition, we prefer another structure called bidirectional LSTM (BLSTM) which can process information in both directions with two separate hidden layers, which are then fed forwards to the same output layer. BLSTMs contain two separate hidden layers, one of them processes the input sequence forwards, while the other processes it backwards. Both hidden layers are connected to the same output layer, providing it with access to the past and future context of every point in the sequence. BLSTM outperform unidirectional LSTMs and standard RNNs and it is also much faster and more accurate [10]. DBLSTMs can be implemented by repeating equations of 3.1 to 3.7 for every layer, and replacing each hidden state, , in every layer with the forward and backward states, and in a way that every hidden layer receives input from both the forward and backward layers at the level below.

## DBLSTM training

One approach to train DBLSTM can be an end-to-end training method like Connectionist Temporal Classification (CTC) and RNN Transducer.

CTC is an RNN output layer specifically designed for sequence labeling tasks without requiring the data to be presegmented, and it directly outputs a probability distribution over label sequences [11]. A CTC output layer contains as many units as there are labels in the task, plus an additional ‘blank’ or ‘no label’ unit. Given a length input sequence , the output vectors are normalised with the softmax function, then interpreted as the probability of emitting the label (or blank) with index at time :

Where is element k of .

A CTC alignment is a length T sequence of blank and label indices. The probability is the product of the emission probabilities at every time-step:

For a given transcription sequence, there are as many possible alignments as there different ways of separating the labels with blanks. For example (using ‘−’ to denote blanks) the alignments (a, −, b, c, −, −) and (−, −, a, −, b, c) both correspond to the transcription (a, b, c). When the same label appears on successive time-steps in an alignment, the repeats are removed: therefore (a, b, b, b, c, c) and (a, −, b, −, c, c) also correspond to (a, b, c). Denoting by B an operator that removes first the repeated labels, then the blanks from alignments, and observing that the total probability of an output transcription y is equal to the sum of the probabilities of the alignments corresponding to it, we can write

This ‘integrating out’ over possible alignments is what allows the network to be trained with unsegmented data. The intuition is that, because we don’t know where the labels within a particular transcription will occur, we sum over all the places where they could occur. Eq. (5.3) can be efficiently evaluated and differentiated using a dynamic programming algorithm. Given a target transcription , the network can then be trained to minimize the CTC objective function:

Another approach beside CTC for training of DBLSTM is RNN transducer which is a combination of a CTC-like network with a seperate DBLSTM that predicts each phoneme given the previous one [12]. Unlike CTC which is a an acoustic-only model, RNN Transducer can integrate acoustic and linguistic information during a speech recognition task and yields a jointly trained acoustic and language model. While CTC predicts an output distribution at each input time step, an RNN transducer determines a separate distribution for every combination of input timestep and output timestep . To do this, RNN transducer is using two networks. The first one called transcription network scans the input sequence and outputs the sequence of transcription vectors. The second one called as the prediction network, scans the output sequence and outputs the prediction vector sequence. RNN transducers can be decoded with beam search to yield an n-best list of candidate transcriptions.

DBLSTM can be regularized using two approaches which are early stopping and weight noise to avoid overfitting. Early-stopping works by monitoring the model’s performance on a validation set. Weight noise combats overfitting by adding Gaussian noise to the network weights during training. Weight noise was added once per training sequence, rather than at every time step. Weight noise tends to ‘simplify’ neural networks, in the sense of reducing the amount of information required to transmit the parameters, which improves generalization.

## DBLSTM performance

DBLSTM performance is explored by running phoneme recognition experiments on the TIMIT corpus. As shown in Table 5.1, nine RNNs were evaluated, varying along three main dimensions: the training method used (CTC, Transducer or pretrained Transducer), the number of hidden levels (1–5), and the number of LSTM cells in each hidden layer. Bidirectional LSTM was used for all networks except CTC-3l-500h-tanh, which had tanh units instead of LSTM cells, and CTC-3l-421h-uni where the LSTM layers were unidirectional.



Table 5.1

The last raw shows that using pretrained Transducer RNN for training of DBLSTM, we can achieve a test set error of 17.7% on the TIMIT phoneme recognition benchmark which is the best recorded score. The three main conclusions can be drawn from table 5.1 are (a) LSTM works much better than tanh for this task, (b) bidirectional LSTM has a slight advantage over unidirectional LSTM and (c) depth is more important than layer size. Although the advantage of the transducer is slight when the weights are randomly initialized, it becomes more substantial when pretraining is used.

# Seizure detection using deep learning

Electroencephalograms (EEGs) are used in a wide range of clinical settings to record electrical activity along the scalp. EEGs are the primary means by which neurologists diagnose brain-related illnesses such as epilepsy and seizures. Manual review of an EEG by a neurologist is time-consuming and tedious. Interrater agreement is low for annotation of low-level events such as spikes and sharp waves. A clinical decision support tool that automatically interprets EEGs can reduce time to diagnosis, reduce error and enhance real-time applications such as ICU monitoring. In this part a classification system for seizure detection based on principles of deep belief network will be discussed.

The first step for seizure detection task using a deep learning network is feature extraction. As usually neurologists look at the EEG signals and mark the seizures without any advanced signal analyzing method, we can develop feature extraction methods that are visible features of the time series such as area under curve, normalized decay, line length, mean energy, average peak amplitude, average valley amplitude, normalized peak number, peak variation and root mean square which are explained in detail in [3].

One deep learning approach for seizure detection task is using Deep Belief Networks (DBN) which are feedforward neural networks consisting of stacked Restricted Boltzmann Machines (RBMs). DBNs are graphical models which learn to extract a deep hierarchical representation of the training data. They model the joint distribution between observed vector and the hidden layers as follows:

In this equation, is a conditional distribution for the visible units conditioned on the hidden units of the RBM at level k, and is the visible-hidden joint distribution in the top-level RBM. This is illustrated in Figure 7.



Figure 7. Deep Belief Network

DBNs with RBMs can be trained using the greedy layer-wise unsupervised training as the building blocks for each layer [13], [14]. To train DBNs first we train the first layer as an RBM that models the raw input as its visible layer. Then we use that first layer to obtain a representation of the input that will be used as data for the second layer. Then we train the second layer as an RBM, taking the transformed data as training examples. In next step we train the second layer as an RBM, taking the transformed data as training examples. Then we iterate the last two steps for the desired number of layers, each time propagating upward either samples or mean values. Finally all the parameters of this deep architecture will be fine-tuned with respect to a proxy for the DBN log- likelihood, or with respect to a supervised training criterion.

The parameters for a seizure detection system using DBNs are as follows [3]. The EEG that collected the readings sampled from 23 channels at 256 hertz. The number of input nodes to the DBN is 207, with 2 output nodes to classify a second of EEG data as a seizure or non-seizure. The best parameter set was found to be two hidden layers of 500 nodes each. Learning rate is 001. After the pretraining process of abstraction was completed (without the usage of class labels), the logistic regression layer was trained in the finetuning process. 16 iterations of finetuning were completed, with a learning rate of 0.1.

The performance DBNs for seizure detection task is compared with other machine learning algorithms like KNN and SVM in [3]. While I find the approach of the authors useful, I do not find the results can give us impotent information, since the experiments ran on a small dataset including related information for six patients. By the way they showed that deep belief networks often outperformed other machine learning algorithm generally.

# Conclusion

In this report, we have investigated deep recurrent neural networks and deep bidirectional long short-term memory and its application in sequential tasks. Additionally a seizure detection system using deep belief network is introduced.

LSTMs seems to be a good candidate for analyzing of EEG signals as it has memory. Currently we are developing a system called AutoEEG which is a predictive analytic tool for EEGs. In the last version of AutoEEG, an HMM is used in the first pass of processing, then a stacked denoising autoencoder used in the second pass of processing and the third pass of processing is composed of a language model that is developed in collaboration with neurologists. As an example, in our language model we have a rule that GPED and PLED cannot be in EEG of one patient. One problem with this method is that if we detect GPED in the beginning of the file by mistake, all of our analysis will be false. However, I believe by implementing deep bidirectional LSTMs, we can train our system to have an excellent performance on detecting events on EEG signals. Additionally, I think we can develop a high performance seizure detection system using BDLSTM.

# Reference

[1] M. Hermans and B. Schrauwen, “Training and Analyzing Deep Recurrent Neural Networks,” *Adv. Neural Inf. Process. Syst.*, pp. 190–198, 2013.

[2] A. Graves, A. Mohamed, and G. Hinton, “Speech Recognition With Deep Recurrent Neural Networks,” *Icassp*, no. 3, pp. 6645–6649, 2013.

[3] A. J. Yepes, A. Mackinlay, J. Bedő, R. Garnavi, and Q. Chen, “Deep Belief Networks and Biomedical Text Categorisation,” in *ALTA*, 2014.

[4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] Y. Bengio, P. Simard, and P. Frasconi, “Learning Long-Term Dependencies with Gradient Descent is Difficult,” *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[6] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International Conference on Machine Learning*, 2013, no. 2, pp. 1310–1318.

[7] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,” *A F. Guid. to Dyn. Recurr. Networks*, pp. 237–243, 2001.

[8] S. Hochreiter, S. Hochreiter, J. Schmidhuber, and J. Schmidhuber, “Long short-term memory.,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–80, 1997.

[9] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *arXiv*, p. 10, 2015.

[10] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM networks,” in *Proceedings of the International Joint Conference on Neural Networks*, 2005, vol. 4, pp. 2047–2052.

[11] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, “Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks,” *Proc. 23rd Int. Conf. Mach. Learn.*, pp. 369–376, 2006.

[12] A. Graves, “Sequence transduction with recurrent neural networks,” in ICML Representation Learning Workshop, 2012

[13] G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science (80-. ).*, vol. 313, no. 5786, pp. 504–507, 2006.

[14] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy Layer-Wise Training of Deep Networks,” *Adv. Neural Inf. Process. Syst.*, vol. 19, no. 1, p. 153, 2007.