RECURRENT DEEP-STACKING NETWORKS FOR SEQUENCE CLASSIFICATION

Hamid Palangi[1], *Li Deng*[2], *Rabab K Ward*[1]

[1]University of British Columbia, Vancouver, BC, Canada
[2]Microsoft Research, Redmond, WA, USA

ABSTRACT

Deep Stacking Networks (DSNs) are constructed by stacking shallow feed-forward neural networks on top of each other using concatenated features derived from the lower modules of the DSN and the raw input data. DSNs do not have recurrent connections, making them less effective to model and classify input data with temporal dependencies. In this paper, we embed recurrent connections into the DSN, giving rise to Recurrent Deep Stacking Networks (R-DSNs). Each module of the R-DSN consists of a special form of recurrent neural networks. Generalizing from the earlier DSN, the use of linearity in the output units of the R-DSN enables us to derive a closed form for computing the gradient of the cost function with respect to all network matrices without backpropagating errors. Each module in the R-DSN is initialized with an echo state network, where the input and recurrent weights are fixed to have the echo state property. Then all connection weights within the module are fine tuned using batch-mode gradient descent where the gradient takes an analytical form. Experiments are performed on the TIMIT dataset for frame-level phone state classification with 183 classes. The results show that the R-DSN gives higher classification accuracy over a single recurrent neural network without stacking.

Index Terms— Recurrent Neural Network, Stacking, Deep Learning, Sequence Classification

1. INTRODUCTION

Deep Neural Networks (DNNs) have proven to yield excellent performance in many pattern classification and recognition tasks [1, 2, 3]. To fine tune DNNs, the stochastic gradient descent method is often used. This makes it difficult to parallelize the learning across different machines. As one way to overcome this problem, Deep Stacking Networks (DSNs) have been proposed [4, 5]. Motivated by the stacking concept of [6], (where complex functions are learnt by stacking a number of simple functions), a DSN is constructed by stacking feed-forward neural networks with one hidden layer having a non-linear activation function for the hidden units and a linear activation function for the output units[4]. Training each module is performed independently of other modules, and there is no need to back propagate the error from the output to the input layers. There are however no temporal connections in each module of a DSN; therefore, the temporal dependencies in the input data are not learnt effectively in DSNs.

Recurrent Neural Networks (RNNs) are models that are deep in time and are used to model dynamical systems and time sequences by directly using recurrent connections [7, 8]. It is well known that RNNs have difficulty in training using Back Propagation Through Time (BPTT), due to vanishing and exploding gradient problems and slow convergence [9]. In fact, getting good results with BPTT is not trivial. A tremendous amount of engineering effort is required to make BPTT work, as seen in [10, 11, 12].

In this paper, to model input data having temporal dependencies more effectively, we introduce the Recurrent Deep Stacking Network (R-DSN) that combines the strengths of both DSNs and RNNs while overcoming their weaknesses. The proposed R-DSN has recurrent (temporal) connections that are missing in the DSN. In the R-DSN, each module is a single-layer RNN with linear output units. To train each module, we do not use BPTT. Instead, we initialize the weights using a special type of RNN known as the Echo State Network (ESN), and then fine tune them using batch-mode gradient descent based on a closed-form formulation to compute the gradients. After each epoch of fine tuning, the echo state property is forced to be satisfied.

In Section 2 we describe the ESN and the related RNN as a basic module of the R-DSN. Section 3 describes the learning method for the input and recurrent weights in ESNs. The stacking and learning method for the entire R-DSN architecture is presented in Section 4. Experimental results for frame level phone recognition on the TIMIT dataset are presented in Section 5. Section 6 presents the conclusions.

2. THE ECHO STATE NETWORK

Echo State Networks (ESNs) are a specific type of the general RNNs that have the echo state property. Similar to RNNs, ESNs have recurrent connections, in addition to the input-to-hidden and the hidden-to-output connections. A sample ESN architecture is shown in Fig. 1. In this figure, $\mathbf{x}_i$, $\mathbf{h}_i$ and $\mathbf{y}_i$

Fig. 1. An ESN unfolded over time

represent the input, hidden and output vectors at discrete time

$t = i$. The connections between the input ($\mathbf{x}_i$) and hidden ($\mathbf{h}_i$) layer are denoted by $\mathbf{W}$. The connections between the hidden layer and output ($\mathbf{y}_i$) are denoted by $\mathbf{U}$. The temporal connections between $\mathbf{h}_i$ and $\mathbf{h}_{i+1}$ are denoted by matrix $\mathbf{W}_{rec}$. In Fig. 1 the direct connections from the input to the output layers form a part of matrix $\mathbf{U}$; i.e., it is equivalent to concatenating the input layer with the hidden layer.

ESNs are proposed to resolve the difficulty in training RNNs [13, 14]. In ESNs, $\mathbf{W}$ and $\mathbf{W}_{rec}$ are not learned but are carefully predetermined. Only hidden layer to output connections and the direct connections from input to output, i.e., $\mathbf{U}$, are trained.

Since the output units in an ESN have a linear activation function and assuming the hidden units have a sigmoid activation function, the formulations for Fig. 1 are:

$$\mathbf{h}_{i+1} = \sigma(\mathbf{W}^T\mathbf{x}_{i+1} + \mathbf{W}_{rec}\mathbf{h}_i) \tag{1}$$

$$\mathbf{y}_{i+1} = \mathbf{U}^T\mathbf{h}_{i+1} \tag{2}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$. ESN is designed such that it has the echo state property [14, 13, 15]. The echo state property implies that the hidden units' states of the network can be determined uniquely based on the current and previous inputs and outputs provided the network has been running for a sufficiently long time. The formal definition of echo states is described in [14]. Assuming that the maximum eigenvalue of $\mathbf{W}_{rec}$ is λ_{max} and the activation function of the hidden units is the sigmoid, if $\mid \lambda_{max} \mid < 4$ then the network has echo states [14]. This is similar to the sufficient condition presented in [16] to prevent the exploding gradient problem for recurrent weights in general RNNs.

Training ESNs has mainly three steps: forming a network with the echo state property, computing the hidden units' states and finding $\mathbf{U}$ using a closed form formulation. To form a network with the echo state property, the input weights matrix $\mathbf{W}$ is randomly generated and usually scaled based on the type of input. Then the sparse recurrent weights matrix $\mathbf{W}_{rec}$ is randomly generated and normalized as follows:

$$\mathbf{W}_{rec} = \lambda\frac{\mathbf{W}_{rec}}{\lambda_{max}} \tag{3}$$

where $\lambda < 4$ for sigmoid activation function and is predetermined based on the given data. λ_{max} is the maximum eigenvalue of $\mathbf{W}_{rec}$.

To find the hidden units' states, the hidden states are initialized to zero or to another initial state. Then the network runs freely for i_{trans} time steps where the hidden states of each time step are calculated using (1). After i_{trans} time steps, the hidden state vectors are stacked in matrix $\mathbf{H}$, i.e.

$$\mathbf{H} = [\mathbf{h}_{i_{trans}}\mathbf{h}_{i_{trans}+1}\dots\mathbf{h}_N] \tag{4}$$

where N is the number of time steps. To calculate the output weights $\mathbf{U}$, we stack the desired outputs corresponding to input signal $\mathbf{x}_i$ as a matrix $\mathbf{T}$; i.e.,

$$\mathbf{T} = [\mathbf{t}_{i_{trans}}\mathbf{t}_{i_{trans}+1}\dots\mathbf{t}_N] \tag{5}$$

Since $\mathbf{H}$ is known (computed using known quantities), $\mathbf{U}$ can then be obtained by minimizing the following mean-square-error cost function:

$$E = \| \mathbf{U}^T\mathbf{H}_c - \mathbf{T} \|_F^2 = tr[(\mathbf{U}^T\mathbf{H}_c - \mathbf{T})(\mathbf{U}^T\mathbf{H}_c - \mathbf{T})^T] \tag{6}$$

where F is the Frobenius norm, $tr(.)$ is the trace and

$$\begin{aligned}\mathbf{H}_c &= [\mathbf{H}\ \mathbf{X}]\\ \mathbf{X} &= [\mathbf{x}_{i_{trans}}\mathbf{x}_{i_{trans}+1}\dots\mathbf{x}_N]\end{aligned} \tag{7}$$

Minimizing (6) by setting the gradient of E to zero results in:

$$\mathbf{U} = (\mathbf{H}_c\mathbf{H}_c^T)^{-1}\mathbf{H}_c\mathbf{T}^T \tag{8}$$

3. LEARNING ONE MODULE OF THE R-DSN

The ESN described above is a special type of the RNN, where the output units are linear and the parameters of the ESN are not learned except the output weight matrix $\mathbf{U}$. In this section, we describe an effective way of learning all ESN parameters, based on and extending the earlier technique developed for the DSN. After the learning, the resulting RNN performs better than the ESN and it forms one "module" of the many modules via stacking, to be presented in the next section.

Assuming the memory of the network extends back to m time steps, we use the following notation to facilitate the development of the learning method for input weight matrix $\mathbf{W}$:

$$\begin{aligned}&\mathbf{X}_1 = [\mathbf{x}_1\ \mathbf{x}_{m+1}\ \mathbf{x}_{2m+1}\dots],\ \mathbf{X}_2 = [\mathbf{x}_2\ \mathbf{x}_{m+2}\ \mathbf{x}_{2m+2}\dots],\ \dots\\ &\mathbf{H}_1 = [\mathbf{h}_1\ \mathbf{h}_{m+1}\ \mathbf{h}_{2m+1}\dots],\ \mathbf{H}_2 = [\mathbf{h}_2\ \mathbf{h}_{m+2}\ \mathbf{h}_{2m+2}\dots],\ \dots\\ &\mathbf{T}_1 = [\mathbf{t}_1\ \mathbf{t}_{m+1}\ \mathbf{t}_{2m+1}\dots],\ \mathbf{T}_2 = [\mathbf{t}_2\ \mathbf{t}_{m+2}\ \mathbf{t}_{2m+2}\dots],\ \dots\end{aligned} \tag{9}$$

Therefore, equations (1) and (2) can be written as:

$$\mathbf{H}_{i+1} = \sigma(\mathbf{W}^T\mathbf{X}_{i+1} + \mathbf{W}_{rec}\mathbf{H}_i) \tag{10}$$

$$\mathbf{Y}_{i+1} = \mathbf{U}^T\mathbf{H}_{i+1} \tag{11}$$

To find the gradient of the cost function E with respect to $\mathbf{W}$ and learn the input weights $\mathbf{W}$ we take into account the dependency between $\mathbf{U}$ and $\mathbf{W}$, $\mathbf{H}$ and $\mathbf{W}$ and the time dependency of $\mathbf{h}_{i+1}$ on $\mathbf{h}_i$ at every time step i. We briefly describe the gradient formulations and learning methods for the input and recurrent weights of one of the modules of R-DSN in this section. Detailed derivations are presented in [17].

3.1. Learning Input Weights

The gradient of the cost function E w.r.t $\mathbf{W}$ can be written as

$$\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}tr(\mathbf{U}^T\mathbf{H}_2\mathbf{H}_2^T\mathbf{U} - \mathbf{U}^T\mathbf{H}_2\mathbf{T}_2^T - \mathbf{T}_2\mathbf{H}_2^T\mathbf{U} + \mathbf{T}_2\mathbf{T}_2^T) \tag{12}$$

Substituting $\mathbf{U} = (\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T$ and calculating the gradient for one time step dependency we have:

$$\frac{\partial E}{\partial \mathbf{W}} = -[\mathbf{X}_1[\mathbf{H}_1^T \circ (\mathbf{1} - \mathbf{H}_1^T)\mathbf{W}_{rec}^T \circ \mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{A}] + \mathbf{X}_2[\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{A}]] \tag{13}$$

where $\circ$ is element-wise multiplication and

$$\mathbf{A} = 2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1} - 2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1} \tag{14}$$

This gradient formulation can be generalized for an arbitrary number of time steps as follows:

$$\frac{\partial E}{\partial \mathbf{W}} = -[\sum_{i=1}^{n} \mathbf{X}_i\mathbf{C}_i] \tag{15}$$

where n is the number of time steps and

$$\mathbf{C}_i = [\mathbf{H}_i^T \circ (\mathbf{1} - \mathbf{H}_i^T)\mathbf{W}_{rec}^T] \circ \mathbf{C}_{i+1} \text{ , } for\ i = 1, \ldots, n-1$$
$$\mathbf{C}_n = \mathbf{H}_n^T \circ (\mathbf{1} - \mathbf{H}_n^T) \circ \mathbf{A} \tag{16}$$

To calculate $\mathbf{A}$ using (14), $\mathbf{H}_n$ and $\mathbf{T}_n$ are used. After calculating the gradient of the cost function w.r.t $\mathbf{W}$, the input weights $\mathbf{W}$ are updated using the following update equation

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \alpha\frac{\partial E}{\partial \mathbf{W}_i} + \beta(\mathbf{W}_i - \mathbf{W}_{i-1}) \tag{17}$$

$$\beta = \frac{m_{old}}{m_{new}}$$
$$m_{new} = \frac{1 + \sqrt{1 + 4m_{old}^2}}{2} \tag{18}$$

where α is the step size and the initial value for m_{old} and m_{new} is 1. The third term in (17) helps the algorithm to converge faster and is based on the FISTA algorithm proposed in [18] and used in [19] and [17].

3.2. Learning Recurrent Weights

To learn the recurrent weights, the gradient of the cost function w.r.t $\mathbf{W}_{rec}$ should be calculated:

$$\frac{\partial E}{\partial \mathbf{W}_{rec}} = \sum_{i=1}^{n} \mathbf{W}_{rec}^{n-i}\mathbf{H}_{i-1}\mathbf{C}_i \tag{19}$$

where $\mathbf{H}_0$ includes the initial hidden states and

$$\mathbf{C}_n = \mathbf{H}_n^T \circ (\mathbf{1} - \mathbf{H}_n^T) \circ \mathbf{A}$$
$$\mathbf{C}_i = \mathbf{H}_i^T \circ (\mathbf{1} - \mathbf{H}_i^T) \circ \mathbf{C}_{i+1} \tag{20}$$

and $\mathbf{A}$ is calculated using (14) based on $\mathbf{H}_n$ and $\mathbf{T}_n$.

Only the non-zero entries of the sparse matrix $\mathbf{W}_{rec}$ are updated using (17) and the gradient calculated in (19). To make sure that the network has the echo state property after each epoch, the entries of $\mathbf{W}_{rec}$ are renormalized such that the maximum eigenvalue of $\mathbf{W}_{rec}$ is λ. This renormalization also prevents the gradient explosion problem for recurrent weights from happening.

4. LEARNING THE RECURRENT DEEP STACKING NETWORK

The RNN described in the preceding section can be stacked into multiple modules. The architecture of a R-DSN with three modules is shown in Fig. 2. In the R-DSN, the output of

Fig. 2. Illustration of an R-DSN architecture with three modules shown with different colors

each module is part of the input of the upper module. Therefore, the dimensionality of the input of the upper modules is more than that of the lower modules. In this work, we have not used RBM or temporal RBM [20] to initialize the weights of the lowest module. Instead, we have used random initialization. The only constraint is that $\mathbf{W}_{rec}$ be initialized such that echo state property holds; i.e., the maximum eigenvalue of $\mathbf{W}_{rec}$ is kept less than 4.

The training method we have implemented for the two modules of architecture in Fig. 2 is as follows:

- Training the first (i.e the lowest) module:
  - Input and recurrent weights are initialized using an ESN architecture
  - Hidden units' states $\mathbf{H}_i^{(1)}, i = 1, \ldots, m$ are calculated using (10)
  - The method described in section 3 is used to compute gradients of the cost function w.r.t. $\mathbf{W}^{(1)}$ and $\mathbf{W}_{rec}^{(1)}$
  - $\mathbf{W}^{(1)}$ and $\mathbf{W}_{rec}^{(1)}$ are updated using (17) and (18)
  - Entries of $\mathbf{W}_{rec}$ are renormalized such that the network has the echo state property using (3)
  - $\mathbf{U}^{(1)}$ is calculated using (8)
- Training the second module:
  - Input weights corresponding to output of the first module $\mathbf{W}_a^{(2)}$ are initialized randomly ($\mathbf{R}_a$ in Fig. 2) and input weights corresponding to the input training data are initialized to the fine tuned input weights from the previous module ($\mathbf{W}^{(1)}$ in Fig. 2)
  - Recurrent weights $\mathbf{W}_{rec}^{(2)}$ are initialized with a random sparse matrix whose sparsity pattern is different from that of the previous module($\mathbf{R}_{rec}^{(2)}$ in Fig. 2)
  - Entries of $\mathbf{W}_{rec}^{(2)}$ are normalized such that the echo state property holds
  - All weights are fine tuned using the method described for the first module

To resolve the exploding gradient problem for input weights we have used the renormalization method proposed in [16].

All modules higher than two in the R-DSN are trained in a similar way to the second step above. The number of epochs used in training all modules are carefully tuned to provide regularization via the "early stopping" mechanism.

Table 1. Frame level phone classification error rates (Err%) for the ESN on the TIMIT core test set as a function of the hidden layer's size (nHid)

nHid	300	500	1000	2000	4000	10000	20000	40000
Err%	71.9	70.1	66.9	63.8	60.9	57.1	54.8	52.7

5. EXPERIMENTS

The standard 462-speaker training set is used in our experiments. A separate dev set of 50 speakers is used for tuning hyper parameters. Results are reported using the 24-speaker core test set, which has no overlap with the dev set. Signal processing for raw speech waveforms is the standard short-time Fourier transform with a 25-ms Hamming window and with a fixed 10-ms frame rate. the standard Mel Frequency Cepstral Coefficients (MFCCs) are then generated, along with their first and second temporal derivatives. All speech data are normalized so that each vector dimension has a zero mean and unit variance before feeding them to a DNN that extracts higher-level features. In our experiments, 183 target class labels are used with three states for each of 61 phones. To prepare the R-DSN targets during training, a high-quality triphone HMM is trained with the training data set, which is then used to generate state-level labels based on the HMM forced alignment. A context window of 3 frames for all experiments (resulting in the input vectors with $3 \times 39 = 117$ entries). The step size (α in (17)) is tuned to be 0.07.

The task was to classify each frame in the TIMIT core test set into one of 183 phone states. For the ESN described in Section 2, we have evaluated the effects of the size of the hidden layer on the classification accuracy. The results presented in Table 1 illustrate very poor performance (i.e. with a high error rate) of the ESN when the hidden layer size is small. However, as the size increases, the error rate drops considerably.

Our further results show that when the input and recurrent weight matrices of the ESN are learned using the method described in Section 3, substantial error reduction is achieved. Even greater error reduction is obtained when this learned ESN is stacked with multiple modules in the way that is described in Section 4. The set of results are summarized in Table 5 for the various R-DSNs with the hidden layer sizes of 4000 and 10000 and with the stacking modules up to three.

6. DISCUSSION AND CONCLUSION

The main focus of this paper was on a novel deep learning architecture, the R-DSN, which extends the earlier RNN and DSN models. The R-DSN constructs multiple modules of the RNN using stacking, in the same way that the DSN uses stacking to form multiple modules of a simple, non-recurrent feed-forward neural network. Alternatively, the R-DSN can be viewed as a generalization of the DSN, where the generalization lies in embedding recurrent connections in each module that were missing in the earlier DSN. The main technical contribution of the work reported in this paper is the development of closed-from formulas for the gradient computation based on the special structure of the R-DSN, and the batch-mode training method for all parameters in the R-DSN capitalizing on these formulas.

Table 2. Frame level phone classification error rates for the R-DSN on the TIMIT core test set as a function of the number of modules for the fixed 4000 or 10000 neurons in the hidden layer

Hidden Units	Modules in R-DSN	Learning $\mathbf{W}$ and $\mathbf{W}_{rec}$ with $m = 1$	Learning $\mathbf{W}$ and $\mathbf{W}_{rec}$ with $m = 3$
4000	1	50.5%	50.0%
4000	2	49.9%	49.5%
4000	3	49.5%	49.1%
10000	1	48.0%	46.8%
10000	2	47.2%	46.0%
10000	3	47.0%	45.7%

Our experiments are designed to test the capability of the R-DSN for time-series data that have strong temporal dependency, for which we chose the TIMIT speech data for the relatively simple frame-level classification task. The results presented in Section 5 demonstrate the effectiveness of the R-DSN with multiple modules over one single module. Future work is planned on extending the evaluation tasks to more complex continuous phone and word recognition tasks and beyond the speech data.

7. REFERENCES

[1] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504 – 507, 2006.

[2] G. Hinton, Li Deng, Dong Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[3] L. Deng, O. Abdel-Hamid, and D. Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *Proc. IEEE ICASSP*, Vancouver, Canada, May 2013.

[4] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, march 2012, pp. 2133 – 2136.

[5] Li Deng and D. Yu, "Deep convex networks for speech pattern classification," *Proc. Interspeech*, 2011.

[6] David H Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[7] Jeffrey L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[8] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. ICASSP*, Vancouver, Canada, May 2013.

[9] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[10] A. J. Robinson, "An application of recurrent nets to phone probability estimation," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 298–305, August 1994.

[11] Tomas Mikolov, Stefan Kombrink, Lukas Burget, JH Cernocky, and Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *Proc. IEEE ICASSP*, Prague, Czech, May 2011, pp. 5528–5531.

[12] I. Sutskever, *Training Recurrent Neural Networks*, Ph.D. thesis, Ph. D. thesis, University of Toronto, 2013.

[13] Herbert Jaeger and Harald Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[14] Herbert Jaeger, "A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach," Tech. Rep., Fraunhofer Institute for Autonomous Intelligent Systems (AIS) since 2003: International University Bremen, 2005.

[15] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens, "Phoneme recognition with large hierarchical reservoirs," *Advances in Neural Information Processing Systems*, 2009.

[16] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. ICML*, Atlanta, GA, June 2013.

[17] Hamid Palangi, Li Deng, and Rabab K Ward, "Learning input and recurrent weight matrices in echo state networks," *Advances in Neural Information Processing Systems (NIPS) Workshop on Deep Learning, Accepted*, 2013.

[18] Amir Beck and Marc Teboulle., *Gradient-based algorithms with applications to signal-recovery problems*, Cambridge University Press, 2009.

[19] D. Yu and L. Deng, "Efficient and effective algorithms for training single-hidden-layer neural networks," *Pattern Recognition Letters*, vol. 33, no. 5, pp. 554–558, 2012.

[20] Ilya Sutskever, Geoffrey E. Hinton, and Graham W. Taylor, "The recurrent temporal restricted boltzmann machine," in *NIPS*, 2008, pp. 1601–1608.