RNNDROP: A NOVEL DROPOUT FOR RNNS IN ASR

Taesup Moon[1,2], Heeyoul Choi[2], Hoshik Lee[2], and Inchul Song[2]

[1]Daegu Gyeongbuk Institute of Science and Technology (DGIST)
333 Techno Jungang-daero , Hyeonpung-myeon, Dalseong-gun, Daegu, 711-873, South Korea

[2]Samsung Advanced Institute of Technology, Samsung Electronics
130 Samsung-ro, Suwon, 443-803, South Korea

ABSTRACT

Recently, recurrent neural networks (RNN) have achieved the state-of-the-art performance in several applications that deal with temporal data, e.g., speech recognition, handwriting recognition and machine translation. While the ability of handling long-term dependency in data is the key for the success of RNN, combating over-fitting in training the models is a critical issue for achieving the cutting-edge performance particularly when the depth and size of the network increase. To that end, there have been some attempts to apply the dropout, a popular regularization scheme for the feed-forward neural networks, to RNNs, but they do not perform as well as other regularization scheme such as weight noise injection. In this paper, we propose *rnnDrop*, a novel variant of the dropout tailored for RNNs. Unlike the existing methods where dropout is applied only to the non-recurrent connections, the proposed method applies dropout to the recurrent connections as well in such a way that RNNs generalize well. Our experiments show that rnnDrop is a better regularization method than others including weight noise injection. Namely, when deep bidirectional long short-term memory (LSTM) RNNs were trained with rnnDrop as acoustic models for phoneme and speech recognition, they significantly outperformed the current state-of-the-arts; we achieved the phoneme error rate of 16.29% on the TIMIT core test set for phoneme recognition and the word error rate of 5.53% on the Wall Street Journal (WSJ) dataset, dev93, for speech recognition, which are the best reported results on both of the datasets.

Index Terms— Recurrent neural networks, LSTM, Dropout, rnnDrop

1. INTRODUCTION

Recurrent neural networks (RNNs) have been successfully applied to many applications including speech recognition [1], handwriting recognition [2] and language understanding [3] for modeling temporal dependencies in data. Recently, as RNNs, deep bidirectional long short-term memory (DBLSTM) networks have been drawing much attention because of their ability to model long-term dependencies and shown state-of-the-art performances in several applications such as image caption generation [4].

To train deep neural networks, regularization is crucial and several methods including pretraining or weight noise injection are typically used to improve generalization performance. Among such regularization methods, dropout was proposed to prevent co-adaptation among hidden nodes of deep feed-forward neural networks by dropping out randomly selected hidden nodes [5,6]. Deep feed-forward neural networks trained with dropout have achieved the state-of-the-art performances in several benchmark datasets [7,8].

However, applying the original dropout scheme to RNNs is not straightforward because of the difficulty in handling recurrent connections. Although there are some attempts to apply dropout to RNNs [9,10], they all apply dropout to only non-recurrent connections since random dropout of recurrent connections makes it hard for RNNs to learn temporal dependencies. As shown in Section 4, a DBLSTM RNN trained with their methods for phoneme recognition did not perform as well as weight noise injection, which is a well-known regularization technique.

In this paper, we propose a new dropout method, *rnnDrop*, which drops out hidden nodes instead of some connections. That is, rnnDrop drops both the non-recurrent and recurrent connections that are connected to the dropped nodes. More importantly, the dropout mask is randomly selected for each input sequence and fixed throughout the sequence. The proposed method is a better way of applying dropout to RNNs, since it can learn temporal dependencies avoiding co-adaptation, which leads to better performances. When we trained DBLSTM RNN acoustic models for phoneme and speech recognition, we obtained the state-of-the-art results: the phoneme error rate (PER) of 16.29% on the TIMIT core test set, and the word error rate (WER) of 5.53% on the Wall Street Journal (WSJ) dataset, dev93.

The rest of this paper is organized as follows. In Section 2, we discuss past research related to this work. In Sections 3 and 4, we describe our dropout method and experiment results, respectively. Finally conclusions follow in Section 5.

2. RELATED WORK

2.1. Long Short-Term Memory (LSTM)

Consider an input sequence (utterance) of length T, $\mathbf{x} = (x_1, \dots, x_T)$, where each $x_t \in R^d$ is a d-dimensional vector. An ordinary RNN with m hidden nodes computes the sequence of hidden node output vectors $\mathbf{h} = (h_1, \dots, h_T)$, $h_t \in R^m$ by the following equations throughout the sequence from $t=1$ to T:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \quad (1)$$

$$y_t = W_{hy}h_t + b_y, \quad (2)$$

where W denotes the weight matrices, b denotes the bias vectors, and y stands for the network output vector. The function $\sigma(\cdot)$ is a nonlinear activation function for hidden nodes, which often takes the form of an element-wise sigmoid function.

The long short-term memory (LSTM) architecture attempts to resolve the exploding and vanishing gradient problems of RNNs [11,12,13]. That is, in order to effectively learn the memory range of the model from the training data, LSTM explicitly designs a memory block inside a hidden node that has the following ingredients: a memory cell which stores information about the past and input, output, and forget gates that control the flow of information within and among the memory blocks. Fig. 1 shows the structure of a single LSTM memory block, which replaces a simple hidden node used in ordinary RNNs.

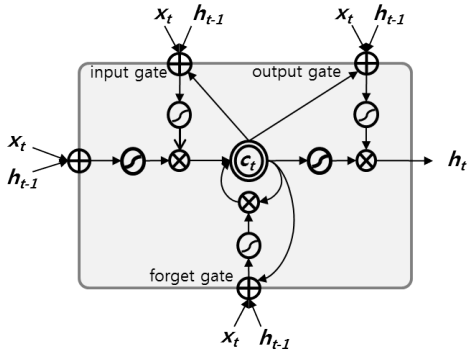

Fig. 1. A long short-term memory block

An LSTM network recurrently applies the following series of equations to obtain the sequence of hidden node outputs, $\mathbf{h} = (h_1, \dots, h_T), h_t \in R^m$:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \quad (3)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \quad (4)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \quad (5)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \quad (6)$$

$$h_t = o_t \tanh(c_t), \quad (7)$$

where the symbols *i, f, o* and *c,* respectively, stand for the input gate, forget gate, output gate, and memory cell state vectors. Note that for the gates, there are not only the recurrent connections from the hidden node outputs from the previous time stamp, but also the peephole connections from the cell states. With explicitly designed cell and gate structures as above, LSTM learns W and b from the training data so that it can determine when to receive input signals to the cell, output the hidden node activations from the memory blocks, and reset the cell states to refresh the memory.

The learning of LSTM parameters can still be done with conventional backpropagation through time (BPTT) algorithm as long as a differentiable loss function on the output layer is used. Optimization with BPTT on such network parameters can be conducted by the stochastic gradient descent (SGD) method.

One can stack multiple LSTM layers to make the network structure deep, and combine two separate LSTM networks that run in forward and backward directions to implement bidirectional architecture. For more detailed coverage on the structure and learning LSTM networks, we refer the readers to [14] and [15] and the references therein. The ability of LSTM network to learn the sequential dependency patterns from the data has led its empirical success in achieving state-of-the-art performances for several sequence recognition tasks such as speech recognition or handwriting recognition [2].

2.2. Dropout

Dropout was first introduced in [5,6] as a training method for preventing co-adaptation among hidden nodes of deep feed-forward neural networks, e.g., DNNs. The method randomly omits hidden nodes with probability p (usually p=0.5) during each iteration of training process, and only the model weights that are connected to surviving nodes are updated by backpropagation. As a result, each hidden node becomes more robust, and the co-adaptation among hidden nodes is mitigated.

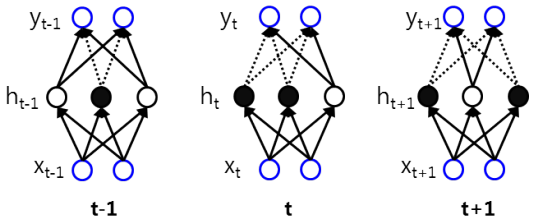

Fig. 2. An example of applying the dropout method proposed in [7] to DNNs. Note that input data for the time stamp *t* typically includes several consecutive frames in speech recognition.

Once the weights are learned with dropout training, during test time, the weights of full model are rescaled by

multiplying (*1- p*) as a mean of effectively averaging the different models learned with random omission during training. In [5,6], it is shown that the neural networks trained with dropout have excellent generalization capabilities and achieve the state-of-the-art performances in several benchmark datasets. Furthermore, dropout played an indispensable role in the neural network systems that won recent learning competitions such as ImageNet classification, the Merck molecular activity challenge, and the Street View House Numbers recognition [5,6,16].

In the context of speech recognition, applying dropout for training a DNN-based acoustic model can be depicted with a simplified example in Fig. 2. Basically, the way to apply dropout is the same as in image recognition, as long as the same network structure is used. The figure shows a two-layer neural network being trained with dropout for three time stamps, *t-1, t,* and *t+1*. Circles stand for nodes in the neural network and arrows are the model weights that connect nodes. The black circles denote the randomly omitted hidden nodes during dropout training, and the weights connected to those omitted nodes are shown with dotted arrows. From the figure, we can see that a different dropout mask (a random omission pattern) is chosen for each time stamp; hence, different sets of model weights, the surviving ones, are updated at each time stamp.

Recently, there have been attempts to apply dropout to recurrent neural networks [9,10]. They all keep the recurrent connections among the hidden nodes and only randomly drop the non-recurrent connections. Their methods have been shown to improve performance in some applications such as handwriting recognition. As presented in Section 4, however, their dropout methods do not perform as well as a well-known regularization method for deep neural networks, called weight noise injection [14].

3. RNNDROP: DROPOUT FOR RNNS

We propose a new dropout method to train recurrent neural networks motivated from the success of dropout for deep feed-forward neural networks. When applying dropout to RNNs, the difficulty arises in dealing with recurrent connections. In contrast to previous work [9,10], we consider dropping both the non-recurrent and recurrent connections. If we apply different random omission patterns, i.e., dropout masks, for different time stamps as in Fig. 2, each hidden node would be omitted very frequently (every 2 time stamps on average if *p=0.5* is used); hence the memory on the past may be easily lost. This frequent memory reset will hurt the main power of RNNs, and LSTM networks in particular, which learns long-term dependencies in data through memory cells. In fact, we observed that when we trained a deep bidirectional LSTM network as just described, the performance of the resulting network dramatically deteriorated.

In order to resolve the above problem and still realize the effectiveness of dropout, instead of applying different dropout masks for different time stamps, our proposed dropout method, *rnnDrop*, generates the dropout mask only at the beginning of each training sequence and fixes it through the sequence. A simple figure to explain the idea is given in Fig. 3. The figure shows an RNN being trained with rnnDrop for three frames (*t-1, t, t+1*) on two different training sequences in the data (denoted as 'sequence1' and 'sequence2'). The black circles denote the randomly omitted hidden nodes during training, and the dotted arrows stand for the model weights connected to those omitted nodes. The recurrent connections to the omitted nodes are deleted and not shown in the figure for clear presentation. The difference from Fig. 2 is that a random omission pattern is fixed throughout each training sequence.

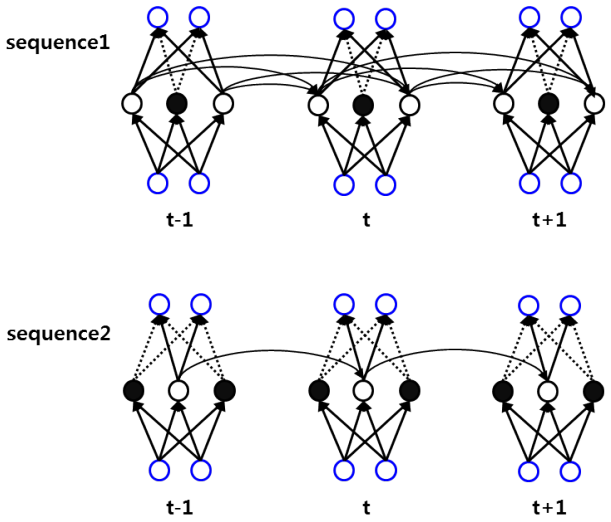

Fig. 3. A new dropout method, *rnnDrop*, for RNNs. The dropout mask is randomly selected for each sequence and fixed throughout the sequence.

We can think of rnnDrop as applying dropout to hidden nodes at the sequence level. We believe this is a more appropriate way of doing dropout for RNNs, since nodes are on or off once for a single training sequence, as they are in DNNs for a single training image. In other words, considering BPTT on unfolded recurrent layers, the network connections remain consistent through time. Moreover, as different random sets of hidden nodes are omitted for different training sequences, rnnDrop still combats co-adaptation among the hidden nodes as in the original dropout.

Mathematically, the proposed rnnDrop can be simply described by adding following one update formula between Eqs. (5) and (6):

$$c_t \leftarrow m_u \odot c_t, \tag{8}$$

where m_u stands for the dropout mask vector with each element drawn independently from Bernoulli(p). The symbol $\odot$ stands for element-wise multiplication. Note from

the subscript *u* that the dropout mask is generated once for each training example (e.g., utterance) and does not depend on *t*. The BPTT algorithm for the model update can be also done by setting the gradients of the weights connecting to the omitted nodes to zero. After training, analogously as in the original dropout, we rescale the learned weights by *1-p* so that the model averaging can be done effectively during test time.

In our experiments, we used deep bidirectional LSTM (DBLSTM) networks to build acoustic models for phoneme and speech recognition. DBLSTM networks not only take the past, but also the future for making a prediction at the current time. The rnnDrop method can be easily applied to DBLSTM networks by generating independent dropout mask vectors for each direction.

4. EXPERIMENTS

In experiments on benchmark datasets, we evaluated the effectiveness of rnnDrop compared to other regularization methods by training DBLSTM-based acoustic models using rnnDrop for TIMIT phoneme recognition [17] and Wall Street Journal (WSJ) large vocabulary continuous speech recognition [18]. We implemented our proposed method on a GPU using NVIDIA CUDA. To speed up the training of large DBLSTM networks, we implemented a data-parallel training algorithm similarly as in [19]. On each iteration of the training process, the SGD method is done separately with randomized disjoint subsets of the training data on different GPUs, which can be in different machines. After that, we average the parameters of the resulting models and redistribute the result to the GPUs. We repeat this process for a specified number of epochs or until convergence.

4.1 TIMIT experiments

We followed the data preparation procedure outlined in [1]. The standard 462 speaker set with all SA records removed was used for training, and a separate development set of 50 speakers with 300 utterances was used for early stopping. The results are for the 24-speaker core test set with 192 utterances. We used the Fourier-transform-based log filterbank with 40 coefficients (plus energy) distributed on mel-scale, together with their first and second derivatives. Thus, the input vector size for each frame was 123. The data were normalized so that every element of the input vectors had zero mean and unit variance over the training set. All 61 phoneme labels were used for training and decoding and then mapped to 39 classes for scoring.

We used exactly the same model configuration as in [1] in order to demonstrate the effect of rnnDrop compared to the regularization method used in [1], i.e., weight noise injection. In weight noise injection, zero mean Gaussian noise is added to the weights when computing the gradient. The DBLSTM networks we used had 5 layers, each of which contains 250 LSTM memory blocks for each direction. We used three sub-phone states for each of 61 phonemes; hence the dimension of target states was 183. A simple GMM-HMM system was used to generate a forced alignment for cross-entropy training. The posterior state probabilities provided by the networks were not divided by the state occupancy priors as in [1]. We constructed a bi-phone language model from the training data and used it for decoding. See [1] for more details on the training procedure such as learning rates, momentum and initialization of weights.

Table 1. Phoneme error rate (PER) results on the TIMIT core test set.

Acoustic Model	Test PER
GMM (2011) [20]	25.6%
DNN (2012) [20]	20.7%
CNN (2012) [20]	20.0%
HP-CNN-DNN [21]	18.7%
DBLSTM + weight noise [1]	18.0%
DBLSTM + dropout as in [9]	18.2%

We first summarize the recent progresses made on the TIMIT core test set in terms of PER in Table 1. The first row in the table shows the PER of a conventional GMM-HMM based system, and the next three the PERs obtained by DNN-HMM based systems. DBLSTM networks trained with weight noise injection performed the best as shown in the fifth row. The last row shows the PER obtained when we applied the dropout method proposed in [9] to DBLSTM networks. Their dropout method was not as effective as weight noise injection in phoneme recognition.

Table 2 shows the PERs of our DBLSTM acoustic models trained with rnnDrop. We have run five experiments, each with different random weight initialization. For each experiment, we picked the model that minimized the PER on the development set and tested it on the core test set. We indicated the mean and standard deviation of PERs obtained from those five experiments.

Table 2. PER results of our DBLSTM-HMM systems with the same complexity as in [1] trained with rnnDrop on the TIMIT validation set and core test set.

Acoustic Model	Dev PER	Test PER
DBLSTM + rnnDrop	**15.93±0.14%**	**16.92±0.19%**

Our DBLSTM acoustic models trained with rnnDrop achieved the test PER of 16.92%, which is a 6% relative improvement over the state-of-the-art. We also have trained a larger 5-layer DBLSTM network with 500 LSTM memory blocks in each direction using rnnDrop and obtained an even lower PER of 16.29%, which is a 10% relative PER reduction compared to the state-of-the-art. Due to time and

resource constraints, we were not able to conduct repeated experiments as before. We believe that, by enlarging the model size and applying rnnDrop, we can further lower the PER.

4.2 Wall Street Journal (WSJ) experiments

We conducted large vocabulary automatic speech recognition experiments using WSJ corpus (available as LDC corpus, LDC93S6B and LDC94S13B). We followed the standard Kaldi recipe s5 [18,22] for preparing speech data.

We used 5-layer DBLSTM networks, in which each direction has 1000 memory blocks. For training, we followed the standard approach. That is, we first trained a baseline GMM-HMM system on the full 81 hours training set by Kaldi recipe tri4b. With the baseline GMM-HMM system, we generated a forced alignment, in which there were 3469 triphone states. It is used for training DBLSTM networks with cross entropy loss. After that, we switched to sequence training. Among many variations of sequence training criteria, we used state-level minimum Bayes risk [23]. We applied rnnDrop to both cross entropy training and sequence training. DBLSTM networks were trained in a data-parallel fashion on 8 GPUs as described above. A pruned trigram language model was used in decoding. We compared the performance of our DBLSTM acoustic models to the state-of-the-art result from Kaldi open source toolkit [18], which used a DNN-based acoustic model.

Table 3. Test results on the WSJ eval92 and dev93 datasets. The relative WER reductions are indicated inside parentheses.

Acoustic Model	eval92 WER	dev93 WER
DNN [18]	3.56%	6.15%
DBLSTM + rnnDrop	**3.53% (0.8%)**	**5.53% (10.1%)**

Table 3 summarizes the word error rate (WER) results on the eval92 and dev93 datasets. We can see that our DBLSTM acoustic models outperformed DNN-based ones used in the Kaldi recipe. Particularly, for the dev93 dataset, which is a harder test set, we achieved 10.1% relative WER reduction compared to the state-of-the-art. Furthermore, when rnnDrop was not used, the performance of DBLSTM acoustic model was worse than that of the DNN-based ones, which shows the critical importance of applying rnnDrop for attaining our results.

5. CONCLUSIONS

In this paper, we have proposed a novel dropout method called *rnnDrop* for RNNs. Compared to other dropout methods, rnnDrop is a better way of applying dropout to RNNs, fixing the random dropout mask for each sequence. With experiment results on benchmark datasets, we showed that DBLSTM acoustic models trained with rnnDrop outperformed the current state-of-the-arts for both phoneme recognition and large vocabulary automatic speech recognition. Obviously, the rnnDrop technique can play a significant role in other applications beyond speech recognition, as long as RNNs are adopted in the application.

Although DBLSTM networks trained with rnnDrop showed superior performances, it took much longer to train them. As future work, we plan to devise a fast rnnDrop method similarly as in [24].

6. REFERENCES

[1] A. Graves, N. Jaitly, and A. Mohamed, Hybrid speech recognition with deep bidirectional LSTM, *ASRU* (2013)

[2] A. Grave, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, **31**(5):855-868 (2009)

[3] Tomas Mikolov, Martin Karafiat, Jan Cernocky, and Sanjeev Khudanpur, Recurrent neural network based language model, *Interspeech* (2010)

[4] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention, arXiv:1502.03044v2 (2015)

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, *Technical Report*, arXiv:1207.0580 (2012)

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, 15:1929-1959 (2014)

[7] G. E. Dahl, T. N. Sainath, and G. E. Hinton, Improving deep neural networks for LVCSR using rectified linear units and dropout, *ICASSP* (2013)

[8] T. N. Sainath, B. Kingsbury, A. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran, Improvements to deep convolutional neural networks for LVCSR, *ASRU* (2013)

[9] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, Dropout improves recurrent neural networks for handwriting recognition, *ICFHR* (2014)

[10] W. Zaremba, I. Sutskever, and O. Vinyals, Recurrent Neural Network Regularization, http://arxiv.org/abs/1409.2329v5

[11] Y. Bengio, P. Simard, and P. Frasconi, Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neural Networks*, **5**(2):157-166 (1994)

[12] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation*, **9**(8):1735-1780 (1997)

[13] F. A. Gers, N. Schraudolph, and J. Schmidhuber, Learning precise timing with LSTM recurrent networks, *Journal of Machine Learning Research*, 3:115-143 (2003)

[14] A. Graves, A. Mohamed, and G. Hinton, Speech recognition with deep recurrent neural networks, *INTERSPEECH* (2013)

[15] A. Graves, Supervised sequence labeling with recurrent neural networks, *Studies in Computational Intelligence*, Springer (2012)

[16] J. Schmidhuber, Deep learning in neural networks: An overview. *Neural Networks*, vol. 61, pp. 85-117 (2015)

[17] C. Lopes and F. Perdigao, Phoneme recognition on the TIMIT database, *Speech Technologies*, pp.285-302 (2011)

[18] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, The kaldi speech recognition toolkit, *IEEE Workshop on Automatic Speech Recognition and Understanding* (2011)

[19] D. Povey, X. Zhang, and S. Khudanpur, Parallel training of DNNs with natural gradient and parameter averaging, ICLR, (2015)

[20] G. E. Hinton, L. Deng, D. Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbur, Deep neural networks for acoustic modeling in speech recognition, *IEEE Signal Processing Magazine*, **29**(6):82-97, (2012)

[21] L. Deng, O. Abdel-Hamid, and D. Yu, A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion, *ICASSP* (2013)

[22] Kaldi open-source toolkit, http://kaldi.sf.net/

[23] B. Kingsbury, Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling, *ICASSP* (2012)

[24] S. I. Wang and C. D. Manning, Fast dropout training, ICML, (2013)