UNDERSTANDING THE DROPOUT STRATEGY AND ANALYZING ITS EFFECTIVENESS ON LVCSR

Jie Li, Xiaorui Wang, Bo Xu

Interactive Digital Media Technology Research Center,
Institute of Automation, Chinese Academy of Sciences,
Beijing, P.R.China
{jie.li, xiaorui.wang, xubo}@ia.ac.cn

ABSTRACT

The work by Hinton et al shows that the dropout strategy can greatly improve the performance of neural networks as well as reducing the influence of over-fitting. Nevertheless, there is still not a more detailed study on this strategy. In addition, the effectiveness of dropout on the task of LVCSR has not been analyzed. In this paper, we attempt to make a further discussion on the dropout strategy. The impacts on performance of different dropout probabilities for phone recognition task are experimented on TIMIT. To get an in-depth understanding of dropout, experiments of dropout testing are designed from the perspective of model averaging. The effectiveness of dropout is analyzed on a LVCSR task. Results show that the method of dropout fine-tuning combined with standard back-propagation gives significant performance improvements.

Index Terms— dropout, deep neural networks, LVCSR

1. INTRODUCTION

The structure of Hidden Markov Models (HMMs) with state-dependent Gaussian mixture models (GMMs) has dominated the field of acoustic modeling during the past few decades. Until recently, noteworthy performance improvements have been made by using Deep Neural Networks (DNNs) to classify the acoustic features into pre-defined HMM-states [1, 2, 3]. Based on the work [4], the authors of [5] present a 20% relative reduction in word error rate comparing with a discriminatively trained HMM-GMM model with MPE criteria.

The specific way of training DNNs proposed by Hinton [6] contains two procedures, one is pre-training, and the other is fine-tuning. Pre-training gives a better starting point in weights space for the optimization. When fine-tuning, however, we usually have to use early stopping to prevent training from over-fitting. In his pioneering paper [7], Hinton shows that if we employ a dropout strategy when training a neural network, not only can we reduce the influence of over-fitting greatly, but also improve the performance of the model significantly. Dropout is executed by randomly omitting each unit with certain probabilities on each training case. The role of the dropout strategy can be explained in two ways, one is to prevent co-adaptations of the units, and the other is to average the predictions of many different networks.

This work was supported by 863 program of China (No.2011AA01A207), National Science and Technology Pillar Program of China (2011BAK05B06), and Tsinghua - Tencent Joint Laboratory for Internet Innovation Technology.

In this paper, we attempt to do a more detailed study on the dropout strategy. The influences of different dropout probabilities on performance are first analyzed. We then try to show the validity of dropout from the view of model averaging by several experiments called dropout testing, which employ dropout during the testing process, just as the name implies. At last, the effectiveness of dropout training is investigated in a large vocabulary continuous speech recognition task. To our knowledge, it is the first time the strategy is used on this task.

The rest of this paper is organized as follows. In section 2, two different perspectives of understanding the dropout strategy are discussed. Section 3 presents the implementation method of dropout fine-tuning and dropout testing in detail. We report our experimental results in section 4 and provide our conclusions in section 5. This paper ends at section 6 which describes the relations to prior work.

2. UNDERSTANDING DROPOUT

According to the work by Hinton et al [7], the dropout strategy can be interpreted from two perspectives. The first is that we can mitigate the influence of over-fitting by using dropout to prevent complex co-adaptations of hidden units on the training data. If a hidden unit knows clearly and definitely who its collaborative workers are, it can adapt to them on the training data nicely. But these complex co-adaptations and symbioses are likely to go wrong on new test data. However, if a hidden unit has to work well with many different sets of co-workers, it will be more dependent on itself rather than relying on some specific combinations of hidden units. As a result, it is more likely to do something that is not only individually useful, but also marginally useful.

The other is that we can treat it as a very efficient way to perform model averaging with neural networks. As a frequently-used method to improve generalization, model averaging is most often used with models such as decision trees because the corresponding training and testing are efficient and straightforward. If we apply model averaging for neural networks by the conventional method, it will be computationally expensive during both training and testing since we have to train many different networks and average the predictions of all these networks when testing. Dropout strategy makes it possible to train plenty of different networks in a reasonable time. With dropout, each time we present a training case, we are actually sampling from a huge number of different architectures randomly.

At test time, we use the network that contains all of the hidden units, but with their outgoing weights halved due to the fact that only half of them are used during training (if 50% dropout is carried out when fine-tuning). This network is called "mean network", which is

a pretty good approximation to averaging the predictions of all the dropped out models when the network contains more than one hidden layer. For single hidden layer network with a "soft-max" output layer, using the "mean network" is exactly equivalent to taking the geometric mean of the probability distribution over labels predicted by all 2^N possible networks, where N represents the number of hidden units.

In this paper, several experiments are designed from the perspective of model averaging to obtain a more in-depth understanding of the dropout strategy.

3. IMPLEMENTATION DETAILS

We apply dropout after the generative pre-training is done, that is, the pre-trained network is fine-tuned with dropout back-propagation, as opposed to standard back-propagation. After dropout fine-tuning, dropout method is integrated with testing, which experimentally shows that the reason for the validity of dropout strategy lies in model averaging. In this section, we will introduce the implementation method of dropout fine-tuning and dropout testing in detail.

3.1. Dropout fine-tuning

The dropout strategy can be applied not only for hidden layers but also for the input layer [7], which is already used by "de-noising auto-encoders" developed by Yoshua Bengio's group [8, 9]. In this paper, we only make use of hidden layer dropout.

For a network with $L-1$ hidden layers and a soft-max output layer, the forward pass of dropout fine-tuning can be expressed as

$$\mathbf{E}_l\left[\mathbf{h}_l|\mathbf{v}_l\right]=\sigma\left(\mathbf{W}_l^T\cdot\mathbf{v}_l+\mathbf{b}_l\right)\quad 1\le l\le L-1 \tag{1}$$

$$P_L\left(s|\mathbf{v}_L\right)=softmax\left(\mathbf{W}_L^T\cdot\mathbf{v}_L+\mathbf{b}_L\right) \tag{2}$$

$$\mathbf{v}_l=dropout(\mathbf{v}_l,p_{drop}),\quad 2\le l\le L \tag{3}$$

with weights matrices $\mathbf{W}_l^T$ and bias vectors $\mathbf{b}_l$, where N_l is the number of units in layer l, $\mathbf{E}_l\left[\mathbf{h}_l|\mathbf{v}_l\right]$ means the conditional expectation of hidden binary vectors $\mathbf{h}_l$ given input vectors $\mathbf{v}_l$, σ denotes the element-wise sigmoid, $\sigma\left(x\right)=\left(1+e^{-x}\right)^{-1}$.

Given input feature vector $\mathbf{o}$, we set $\mathbf{v}_1=\mathbf{o}$ and compute $\mathbf{E}_l\left[\mathbf{h}_l|\mathbf{v}_l\right]$ according to formula (1) which will be treated as input $\mathbf{v}_{l+1}$ to the next layer. Before the next calculation using $\mathbf{v}_{l+1}$, however, we should pass it through formula (3) to perform dropout, which implies that the input vector of each hidden layer and the output layer should be dropped out according a probability constant p_{drop} (except for the first hidden layer, since no dropout is performed on the network input).

The implementation method of formula (3) is as follows: for each training case, a vector of random numbers between 0 and 1 which subject to a uniform distribution is generated. The dimensionality of the random vector is equal to that of input vector $\mathbf{v}_l$. If a component in the random vector is less than p_{drop} , the corresponding value of vector $\mathbf{v}_l$ will be set to 0.

The constant p_{drop} represents the probability that one unit would to be omitted. Different constants are tested in this work, and results are given in the next section. During the backward pass of fine-tuning, the weights which associate with units that are dropped out should not participate in gradient computation, that is, the gradient due to those units should be 0.

3.2. Dropout testing

Dropout testing means that instead of using "mean network" when testing, we use the "original network" of which units of each hidden layer are randomly omitted according a probability constant p_{drop} and outgoing weights are not multiplied with the compensation probability $1-p_{drop}$. Two kinds of dropout testing are examined in this work, one is called "network-output averaging" and the other is "layer-wise averaging". The meaning of network-output averaging is that for each utterance in test corpus, we perform dropout feed forward according to formula (1) (2) (3) for certain times and average the class posterior probabilities (output of network) obtained from these testings. Averaging is performed according to:

$$\begin{aligned}\bar{P}_L\left(s|\mathbf{v}_L\right)&=\tfrac{1}{T}\sum_{i=1}^{T}P_L^{(i)}\left(s|\mathbf{v}_L\right)\\&=\tfrac{1}{T}\sum_{i=1}^{T}softmax^{(i)}\left(\mathbf{W}_L^T\cdot\mathbf{v}_L+\mathbf{b}_L\right)\end{aligned} \tag{4}$$

where $P_L^{(i)}\left(s|\mathbf{v}_L\right)$ is the posterior probability of class s obtained from the i-th testing, T denotes how many times of testing we would like to do, and $\bar{P}_L\left(s|\mathbf{v}_L\right)$ is the averaged posterior probability which is to be used for decoding.

To do layer-wise averaging, we replace formula (3) with

$$\mathbf{v}_l{}^{(i)}=dropout^{(i)}(\mathbf{v}_l,p_{drop}),\ 2\le l\le L,\ 1\le i\le T \tag{5}$$

where $\mathbf{v}_l{}^{(i)}$ is the result vector of the i-th dropout processing on $\mathbf{v}_l$.

Two forms of layer-wise averaging are tested namely "layer-input averaging" and "layer-output averaging" respectively. The layer-input averaging is performed according to

$$\bar{\mathbf{v}}_l=\frac{1}{T}\sum_{i=1}^{T}\mathbf{v}_l{}^{(i)},\quad 2\le l\le L \tag{6}$$

which implies that for input vector $\mathbf{v}_l$ of each hidden layer (except for the first hidden layer) and the output layer, dropout method is executed for certain times T and the results are averaged to get $\bar{\mathbf{v}}_l$ which will be sent to the next layer as input vector.

For the form of layer-output averaging, the procedure is as follows. After dropout is performed according to formula (5) for T times, T different input vectors $\mathbf{v}_l{}^{(i)}$ are obtained, each of which will be separately sent to the next layer and be used to compute $\mathbf{E}_l\left[\mathbf{h}_l|\mathbf{v}_l\right]$ (formula (1), for hidden layers except the first one) or $P_L\left(s|\mathbf{v}_L\right)$ (formula (2), for the output layer). Then, for hidden layers (except the first one), these T layer-output vectors are averaged according to formula (7) and sent to the next layer. As for the output layer, averaging is performed according to formula (8).

$$\begin{aligned}\mathbf{E}_l\left[\mathbf{h}_l|\mathbf{v}_l\right]&=\tfrac{1}{T}\sum_{i=1}^{T}\mathbf{E}_l\left[\mathbf{h}_l|\mathbf{v}_l^{(i)}\right]\\&=\tfrac{1}{T}\sum_{i=1}^{T}\sigma\left(\mathbf{W}_l^T\cdot\mathbf{v}_l^{(i)}+\mathbf{b}_l\right)\end{aligned} \tag{7}$$

$$\begin{aligned}P_L\left(s|\mathbf{v}_L\right)&=\tfrac{1}{T}\sum_{i=1}^{T}P_L\left(s|\mathbf{v}_L^{(i)}\right)\\&=\tfrac{1}{T}\sum_{i=1}^{T}soft\max\left(\mathbf{W}_L\cdot\mathbf{v}_L^{(i)}+\mathbf{b}_L\right)\end{aligned} \tag{8}$$

One thing we need to pay attention to is that the dropout probability (p_{drop}) used for testing should be consistent with that used during dropout fine-tuning.

4. EXPERIMENTAL RESULTS

Following the work [7], we first apply dropout fine-tuning on TIMIT dataset to evaluate its performance on the task of phone recognition. The impacts on the performance with different dropout probabilities are analyzed in this experiment. Next, the experiments of dropout testing are conducted to obtain a better understanding of the dropout strategy. In the end, the effectiveness of dropout fine-tuning on large vocabulary continuous speech recognition (LVCSR) is examined.

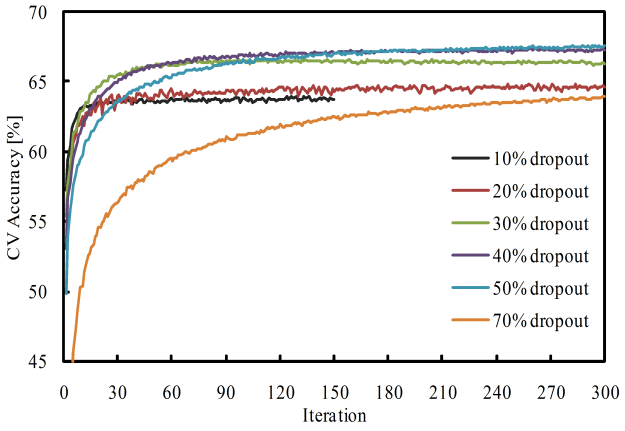

Fig. 1. CV accuracy [%] curves of different dropout probabilities. For 10% dropout, we run the model for 150 epochs. For other values of dropout probability, the model is run for 300 epochs.

Table 1. *Core test PER [%] of different dropout probabilities.*

p_{drop}	10%	20%	30%	40%	50%	70%
PER	22.64	22.51	21.47	**21.10**	21.37	24.82
Baseline	22.52					

4.1. Dropout fine-tuning on TIMIT

For phone recognition experiments on TIMIT corpus, a development set of 50 speakers is selected from the complete test set, and the 24-speaker core test set is used for evaluation. The deep neural network used in these experiments has 4 fully-connected hidden layers, each of which contains 2048 units, and the output layer has 183 softmax units. The conventional 13-dimension MFCC features, along with their first and second derivatives are used. Cepstral mean and variance normalization is performed on per utterance case. For experiments reported below, 11 consecutive frames are used as input features of the network. Mini-batches of size 128 are applied for both pre-training and dropout fine-tuning.

Dropout back-propagation algorithm is used to fine-tune the neural network with weights initialized by pre-trained RBMs. Learning rate is applied to total gradient of a mini-batch with a constant value of 0.008. To get baseline result, the network is fine-tuned using standard back-propagation.

To analyze effects of different dropout probabilities, we employ 10%, 20%, 30%, 40%, 50% and 70% dropout fine-tuning separately. CV accuracy curves are presented in Figure 1 and PER results are given in Table 1.

Figure 1 shows that with the increase of dropout probability, the CV accuracy curve needs more epochs to converge. However, the convergent point of these curves tend to be higher (except for 70% dropout). From Table 1, we can see for 30%, 40% and 50% dropout, the performance improvements are remarkable. Comparing with the baseline result, a relative PER reduction of 6.3% is achieved by 40% dropout. The reason for the poor performance of 70% dropout is that the learning capacity of the model degrades since too many units are omitted during training.

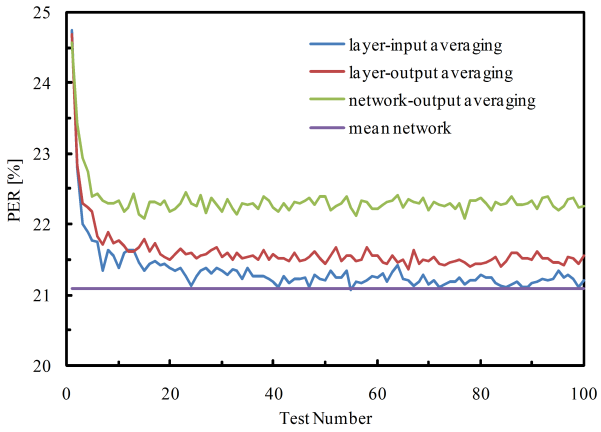

Fig. 2. Core test PER [%] as a function of test number for two kinds of averaging.

4.2. Dropout testing on TIMIT

All the three experiments of dropout testing are conducted following the description in section 3.2 with test number T varying from 1 to 100. According to the results in section 4.1, 40% dropout gives the best performance, thus, p_{drop} is 40% for all experiments reported below. In the experiments of this section, we use the same model which has the lowest test PER (21.10%) in section 4.1.

For the purpose of reducing interference due to randomness, these experiments are conducted for three times separately and the results are averaged to obtain three final test PER curves which are presented in Figure 2.

Comparing with the two curves of layer-wise averaging, the curve obtained from network-output averaging converges to mean network PER (21.10%) much more slowly. This is because network-output averaging is just whole network averaging, while layer-wise averaging can be viewed as a kind of accumulation of single layer model averaging. If we perform layer-wise averaging with a test number of T, we actually have done model averaging for T^l times, where l is the number of layers on which dropout is performed. As for this paper, l is 4 (three hidden layers plus the output layer).

Figure 2 also shows that layer-input averaging converges to mean network PER a lot faster than layer-output averaging. This is due to the consistency of layer-input averaging with the forward pass of dropout training. For one mini-batch in the training corpus, it has been sent into the network for a certain number of times, and the number is just the iteration epochs of the training procedure. When the test number T goes to infinity, layer-input averaging is equivalent to mean network.

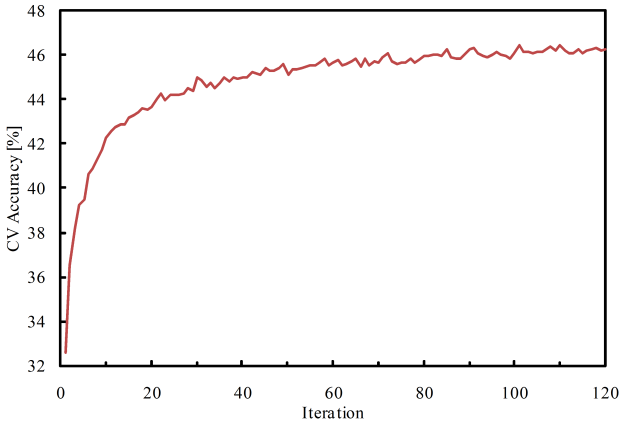

Fig. 3. CV accuracy [%] curve of dropout fine-tuning.

Table 2. *CV accuracy [%] of different models.*

	Standard BP	Dropout	Dropout+SBP
CV acc(%)	46.36	46.45	**49.81**

From Figure 2 we can see that after about 25 times of layer-input averaging, the PER curve already comes near mean network PER (21.10%). This illustrates using "mean network" when testing is actually doing layer-input averaging, and the validity of dropout strategy takes its root in model averaging.

4.3. Dropout fine-tuning on LVCSR

The effectiveness of dropout strategy on the task of LVCSR is tested and verified in this section.

4.3.1. Configuration

A cross-word triphone GMM-HMM model is first trained with discriminative criteria (bMMI), using 1000 hours of Mandarin data. This model serves as a baseline system to generate labels at the frame level. All experiments are conducted using 42 dimension features which consist of 13-dimensional PLP and pitch appended with the first and second order derivatives.

A DNN model which contains 5 hidden layers with 2048 units in each layer and an output layer with 10217 senones is trained on a subset (about 70h) of above dataset. Concatenations of 11 frames are used as input of our network. The baseline result of the DNN model is obtained by standard back-propagation algorithm.

To evaluate the performance, we use two individual test sets namely "clean7k" and "noise360" which are collected by mobile microphone under clean and noise environments, respectively.

4.3.2. Results

The network with weights initialized by pre-trained RBMs is first fine-tuned for 120 epochs using dropout back-propagation with $p_{drop} = 40\%$, and the CV accuracy curve is presented in Figure 3. We also train the same network using standard back-propagation after 100 iterations of dropout fine-tuning. This model is called "Dropout+SBP". CV accuracy and test CER is given in Table 2 and Table 3. In order to accelerate the training process, we use the method of ASGD [10].

Table 3. *CER [%] of different models.*

	CER(%)	
	clean7K	noise360
GMM_bMMI	11.30	36.56
Standard BP	10.34	30.30
Dropout	11.64	34.13
Dropout+SBP	**9.76**	**29.70**

Unfortunately, comparing with standard back-propagation, the dropout strategy does not perform better on both of these two test sets, though the CV accuracy is about equal. If dropout is turned off during the last few iterations of training, the CV accuracy can be promoted from 46.36% to 49.81%, and the relative CER reduction is 5.6% and 2.0% for "clean7k" and "noise360" test sets respectively. However, the performance improvements are not so obvious as our expectations. We will do more research on it in the future.

5. CONCLUSION

This paper presents a further study on the dropout strategy. For the task of phone recognition, we perform dropout fine-tuning on TIMIT with six different dropout probabilities. Experimental results show that with the increase of probability, the convergence speed of CV accuracy curve becomes slower, and more extreme probability (greater than 50%) tends to be worse. For the network used in this paper, 40% dropout achieves the best performance on this task, which is a relative PER reduction of 6.3% compared with baseline result. We design experiments of dropout testing aiming to get a deeper understanding of dropout strategy and verifying the validity of dropout strategy in the view of model averaging. Figure 2 demonstrates using "mean network" is actually doing layer-input averaging, and the validity of dropout strategy lies in model averaging. In the end, over 70 hours of training data is used to analyze the effectiveness of dropout on the task of LVCSR. It turns out that if we fine-tune the network without dropout after certain iterations of dropout fine-tuning, the performance improvement is significant.

6. RELATION TO PRIOR WORK

Dropout strategy is first proposed and examined on different benchmark datasets in the pioneering work [7]. Based on that, we make a further discussion on the dropout strategy, including what the influences of different dropout probabilities are and how to understand dropout in the view of model averaging. What's more, we analyze the effectiveness of this method on the task of LVCSR for the first time.

7. REFERENCES

[1] Frank Seide, Gang Li, and Dong Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks," in *Interspeech*, 2011, pp. 437–440.

[2] Navdeep Jaitly, Patrick Nguyen, Andrew Senior, and Vincent Vanhoucke, "Application of Pretrained Deep Neural Networks to Large Vocabulary Speech Recognition," in *Interspeech*, 2012.

[3] Abdel rahman Mohamed, Tara N. Sainath, George Dahl, Bhuvana Ramabhadran, Geoffrey E. Hinton, and Michael A. Picheny, "Deep Belief Networks using discriminative features for phone recognition," in *ICASSP*, 2011, pp. 5060–5063.

[4] Abdel rahman Mohamed, George E. Dahl, and Geoffrey Hinton, "Acoustic Modeling Using Deep Belief Networks," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, pp. 14–22, 2012.

[5] George E. Dahl, Dong Yu, Li Deng, and Alex Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, pp. 30–42, 2012.

[6] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[7] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre antoine Manzagol, "Extracting and composing robust features with denoising autoencoders," in *International Conference on Machine Learning*, 2008, pp. 1096–1103.

[9] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[10] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu, "Asynchronous Stochastic Gradient Descent for DNN Training," in *ICASSP*, 2013 (to appear).