General instructions for the assessment: The better assessment, the higher mark, i.e. 0 = the poorest mark, 10 = the best mark (the only exception is the 'Amount of rewriting' field where 0 means 'no rewriting necessary' and 10 means 'the paper must be entirely rewritten'). Please, use your common sense or ask the organizers if unsure. This form should be filled in using Adobe Acrobat – please, do not use the built-in PDF viewer in your browser

Originality – Rate how original the work is:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Significance – Rate how significant the work is:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Relevance – Rate how relevant the work is:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Presentation – Rate the presentation of the work:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Technical quality – Rate the technical quality of the work:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Overall rating – Rate the work as a whole:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Amount of rewriting – Express how much of the work should be rewritten:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Reviewer's expertise – Rate how confident you are about the above rating:

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Main contributions – Summarise main contributions:

Positive aspects – Recapitulate the positive aspects:

Negative aspects – Recapitulate the negative aspects:

Comment (optional) – A message for the **author(s)**:

Internal comment (optional) – An internal message for the **organizers**:

A Regularization Post Layer: An Additional Way how to Make Deep Neural Networks Robust

Anonymous*

Affiliation
email@email

Abstract. Neural Networks (NNs) are prone to overfitting. Especially, the Deep Neural Networks in the cases where the training data are not abundant. There are several techniques which allow us to prevent the overfitting, e.g., L1/L2 regularization, unsupervised pre-training, early training stopping, dropout, bootstrapping or cross-validation models aggregation. In this paper, we proposed a regularization post-layer that may be combined with prior techniques, and it brings additional robustness to the NN. We trained the regularization post-layer in the cross-validation (CV) aggregation scenario: we used the CV held-out folds to train an additional neural network post-layer that boosts the network robustness. We have tested various post-layer topologies and compared results with other regularization techniques. As a benchmark task, we have selected the TIMIT phone recognition which is a well-known and still favorite task where the training data are limited, and the used regularization techniques play a key role. However, the regularization post-layer is a general method, and it may be employed in any classification task.

Keywords: Phone Recognition, Acoustic Modeling, Neural Networks, Regularization, Neural Networks Ensemble

1 Introduction

A usual problem that occurs during neural network training is called overfitting: the accuracy on the training set is driven to be a very nice, but when new data is presented to the network the accuracy drops. The network has nicely memorized the training data, but it has not learned to generalize the new ones.

A method for improving network generalization is to use a network that is just large enough to provide an adequate fit. The larger/deeper network we use, the more complex the functions the network can create. If we use a small enough network, it will not have enough power to overfit the data. Unfortunately, it is hard to know beforehand how large a network should be for a specific task. In the automatic speech recognition (ASR) domain, too shallow networks do not have an ability to learn the complex dependencies in the speech data. Therefore, it is preferred to use a deep enough network in combination with some techniques that prevent the overfitting. A brief overview of the most popular techniques follows.

* This research was supported by some Grant.

1.1 Early Stopping

A possible way to regularize a neural network is *early stopping*, meaning that the training procedure is monitored by a validation set. When the validation error increases for a specified number of iterations, the training is stopped.

1.2 L1, L2 Regularization

The other way to better neural networks generalization ability is to add an extra term to the error function that will penalize complexity of the network [3]. Penalizing the number of nonzero weights is called *L1* regularization: a coefficient times a sign of weight values are added to the error gradient function. *L2* regularization uses a sum of the squares: a fraction of weights values is added to the error gradient function. We can set the L1 and L2 coefficients for an entire network or on a per-layer basis.

1.3 Dropout

The term *dropout* refers to dropping out units (hidden and visible) in a neural network [10]. The dropping a unit out means temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set in an interval between 0 and 0.5. For the input units, however, p is usually closer to 0 than to 0.5. At training time, a new random set of weights is sampled and trained for each training sample. At test time, a very simple approximate averaging method works well in practice. The weights of this network are scaled-down versions of the trained weights. The outgoing weights are multiplied by $1 - p$ at test time. This ensures that for any hidden unit the expected output is the same as the actual output at test time [10].

The dropout technique described above prevents feature co-adaptation by encouraging independent contributions from different features. However, repeatedly sampling a random subset of active weights makes training much slower. Wang et al. in [14] experimented with a Gaussian approximation of integrated dropout sampling. This approximation, justified by the central limit theorem and empirical evidence, gives an order of magnitude speedup and more stability.

1.4 Shakeout

From the statistic point of view, the dropout works by implicitly imposing a *L2* regularizer on the weights. Kang et al. presented in [6] a new training scheme: *shakeout*. Instead of randomly discarding units as the dropout does at the training stage, the shakeout method randomly chooses to enhance or inverse the contributions of each unit to the next layer. This scheme leads to a combination of L1 regularization and L2 regularization imposed on the weights of the model. The technique was called *shakeout* because of the randomly shaking process and the L1 regularization effect pushing network weights to zero. The models trained by shakeout generalize better than the standardly trained networks. Moreover, in adverse training conditions, such as with limited data, the shakeout training scheme outperforms the dropout training.

1.5 Unsupervised Pre-Training

Since gradient-based optimization often appears to get stuck in poor solutions starting from random initialization, Hinton et al. in [5] proposed a greedy layer-wise unsupervised pre-training learning procedure. This procedure relies on the training algorithm of restricted Boltzmann machines (RBM) and initializes the parameters of a deep belief network (DBN), a generative model with many layers of hidden causal variables. The greedy layer-wise unsupervised training strategy helps the optimization by initializing weights in a region near a good local minimum, but also implicitly acts as a sort of regularization that brings better generalization and encourages internal distributed representations that are high-level abstractions of the input [7]. The main idea is a using unsupervised learning at each stage of a deep network as a part of a training procedure for DBN. Upper layers of a DBN are supposed to represent more abstract concepts that explain the input observation, whereas lower layers extract low-level features from the input. In other words: this model first learns simple concepts, on which it builds more abstract concepts. In combination with the early stopping technique, the DBN pre-trained networks provide better generalization ability.

1.6 NNs Ensemble: Models Combination and Aggregation

Ensemble method or combining multiple classifiers is another way to improve the generalization ability [4], [15], [17]. By averaging or voting the prediction results from multiple networks, we can significantly reduce the model classification variance. The motivation of combining several neural networks is to improve a new data classification performance over individual networks. Perrone et. al show theoretically in [8] that the performance of the ensemble can not be worse than any single model used separately if the predictions of the individual classifier are unbiased and uncorrelated.

Bootstrap Aggregation - Bagging Bootstrap aggregation, also called bagging, is a method for generating multiple versions of a predictor and using these to get an aggregated predictor [2]. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets. The vital element is the instability of the prediction method. If perturbing the training set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Cross-Validation Aggregation - Crogging In classification, cross-validation (CV) is widely employed to estimate the expected accuracy of a predictive algorithm by averaging predictive errors across mutually exclusive sub-samples of the data. Similarly, bootstrapping aims to increase the validity of estimating the expected accuracy by repeatedly sub-sampling the data with replacement, creating overlapping samples of the data. Beyond error estimation, bootstrap aggregation or bagging is used to make a NNs ensemble. Barrow et al. considered in [1] similar extensions of cross-validation to create diverse models. By bagging, it was proposed to combine the benefits of cross-validation

and prediction aggregation, called crogging. In [1], the crogging approach significantly improved prediction accuracy relative to the bagging.

2 Regularization Post-Layer (RPL)

The RPL technique is based on cross-validation aggregation; however, it is more advanced. Cross-validation ensures that all observations are used for both training and validation, though not simultaneously, and each observation is guaranteed to be used for model estimation and validation the same number of times. Furthermore, the validation set available in CV can be used to control for overfitting in neural network training using early stopping. k-fold cross-validation allows the use of all k validation sets in performing early stopping, and this potentially further reduces the risk of overfitting. Moreover, prediction values for all k folds can be obtained in validation mode. These validation-predictions for all folds – all training data – make a new valuable training set for an additional NN layer that we called regularization post-layer.

The RPL input dimension is equal to the number of classes, and the output dimension is the same. We assume that the main NN ends by a softmax layer which majority of classification NNs uses. To be compatible, RPL must end by the softmax layer too. Generally, we do not need to restrict RPL to the single layer. However, to be robust, the single layer is more than enough. We prefer using of log-softmax values of the main NN as an input to RPL because RPL ends by additional softmax layer that assumes log-domain inputs. Because of log-domain processing, a multiplication in RPL corresponds to power at the output/probability domain, and an addition in RPL corresponds to a multiplication at the output. We lacked in a function that corresponds to an addition at the output. Therefore, we propose to add a log-addition part to RPL.

Let x_i be an log-sofmax output vector of the main NN, where $i = 1 \dots N$. N is number of training samples. In general case, we propose the RPL topology as follows:

$$y_i = \text{softmax}\Big(\text{addlog}(x + Wx + b, lb)\Big), \tag{1}$$

where *addlog* is a log-addition function that can be computed in a robust way as follows:

$$\text{addlog}(a, b) = \max(a, b) + \log\left(1 + \exp\Big(\min(a, b) - \max(a, b)\Big)\right) \tag{2}$$

In equation 1, W is a square matrix, b is a vector of offsets, and lb is a vector of offsets in non-log-domain. $x + Wx$ form helps to initialize better W and regularize the matrix W. In this form, the matrix W may be initialized close to zero and stabilized by L1/L2 regularization during training. The vector b may be initialized close to zero also. The vector lb is in log-domain, so it cannot contain zero or negative values, we initialized it to a small positive value e.g. log(1e-6). We trained RPL in the same way as the main NN: By stochastic gradient descend (SGD) with identical training parameters.

Because of RPL is the last layer, the training error or overfitting of RPL is critical. Therefore, for a high number of classes, where matrix W is large, some additional regularization or treatment is necessary. Besides L1/L2 regularization, one may try a diagonal matrix W only or decomposition the square matrix W to a multiplication of two

square matrices $W = AB$, where an additional tunable hidden dimension parameter of matrices A and B is available.

In the testing phase, k outputs of main NNs are available. We follow the crogging technique: we aggregate all k outputs first and then proceed to the RPL that produces the single final output.

3 Regularization Techniques in Automatic Speech Recognition (ASR)

Training of NN as an acoustical model for ASR has some unique issues. The main fundamental issue is the data dependency. Usually, the training data samples are defined to be independent, but it does not hold in ASR. In a stream of processed audio signal subsequent samples are mutually dependent and very similar. Also, data produced by the same speaker or under the same condition (recording hardware, codec, environment) are dependent. It is well known in the community and could be partially avoided by splitting between training/validation parts is done at speaker-level. Cross-validation and bootstrapping need to be done at speaker-level also. The dependency in data lowers the "effective" amount of training data and we need to process a higher number of training samples during training relatively to another task where the data samples are independent. Therefore, ASR NNs are more prone to overfitting and the proper combination of regularization techniques belongs to key parts that affect the total ASR performance.

Between the most often used techniques in ASR belongs early stopping, L2 regularization, dropout, and DBN unsupervised pre-training. The pre-training also reduces total training time because the training starts much closer to a local optimum. On the other hand, pre-training is hard to use for the most recent NN types as (B)LSTM and TDNN. Using of dropout in ASR is sometimes tricky; p-values closer to one is used, or dropout is utilized in the first several training epochs only. While the dropout method was not designed for noise robustness, in [9] it was demonstrated that it was useful for noisy speech as it produced a network that was highly robust to variabilities in the input. NNs ensemble methods including bagging and crogging have not been popular in ASR because of high computation requirement also in the test phase. Most of the ASR systems needs to keep computation burden low. However, offloading most of the NN computation to a GPU offers a space to apply more complex NNs, including NNs ensemble.

4 Experiments

As a benchmark task, we have selected the TIMIT phone recognition which is a well-known and still popular task where the training data are limited, and the used regularization technique plays a key role. As a baseline system, we used Kaldi and its TIMIT training example which is publicly available and offer easy experiments repeatability.

4.1 Speech Data, Processing, and Test Description

As mentioned above, we used TIMIT data available from LDC as a corpus LDC93S1. Then, we ran the Kaldi TIMIT example script, which trained various NN-based phone

recognition systems with a common HMM-GMM tied-triphone model and alignments. The common baseline system consisted of the following methods: It started from MFCC features which were augmented by δ and $\delta\delta$ coefficients and then processed by LDA. Final feature vectors dimension is 40. We obtained final alignments by HMM-GMM tied-triphone model with 1909 tied-states. We trained the model with MLLT and SAT methods, and we use fMLLR for the SAT training and a test phase adaptation. We dumped all training, development and test fMLLR processed data, and alignments to disk. Therefore, it is easy to do compatible experiments from the same common starting point. We employ a bigram language/phone model for final phone recognition. A bigram model is a very weak model for phone recognition; however, it forces focus to the acoustic part of the system, and it boosts benchmark sensitivity. The training, as well as the recognition, was done for 48 phones. We mapped the final results on TIMIT core test set to 39 phones (as it is usual by processing TIMIT corpora), and phone error rate (PER) was evaluated by the provided NIST script to be compatible with previously published works. In contrast to the Kaldi recipe, we used a different phone decoder to be able to test novel types of NNs. The decoder is simple, but it does not tie mapped triphones and it process entire "full-triphone" lattice. It gives slightly better results than Kaldi standard WFST decoder. To be comparable, we used our full-triphone decoder for all results in this paper.

4.2 NNs Topology and Initialization

We used a common NN topology for all tests because the focus of this article is NN regularization and generalization ability. It follows the standard Kaldi recipe: we stacked input fMLLR feature 11 frames long window to 440 NN input dimension. All the input vectors were transformed by affine transform to normalize input distribution. The net had 6 layers with 1024 sigmoid neurons and the final softmax layer with 1909 neurons. We used the DBN pre-training. In this point, we exported the NN parameters from Kaldi to our Theano-based training tool and made 10 epochs of SGD. Ten epochs are not enough to train a good model, but it was our common starting point for all our experiments. This way, we reduced the total computation time. We trained all NNs with L2 regularization with value 1e-4. Note that we also tested more recent types of NNs: LSTM NN and TDNN. However, we were not able to obtain better results compared to above mentioned deep NN (DNN).

4.3 Results

In Table 1, we show the summary of results. *DNN* means above described NN randomly initialized and trained by SGD with L2 regularization and early stopping. *DBN_DNN* is identical to *DNN* but with DBN unsupervised pre-training. *DBN_DNN_sMBR* is *DBN_DNN* followed by sequence-discriminative training [13]. *DBN_DNN_Dropout* is *DBN_DNN* trained with dropout (probability of dropping out $p = 0.1$). *DBN_DNN_Bagging_5* is an aggregation of 5 bootstraped NNs, all trained from the same DBN initialization. *DBN_DNN_Crogging_5* is an aggregation of 5 NNs done by a 5-fold cross-validation, all NNs were trained from the same DBN initialization. *DBN_DNN_RPL*$_{Full}$ and *DBN_DNN_RPL*$_{Diag}$ are based on *DBN_DNN_Crogging_5* but

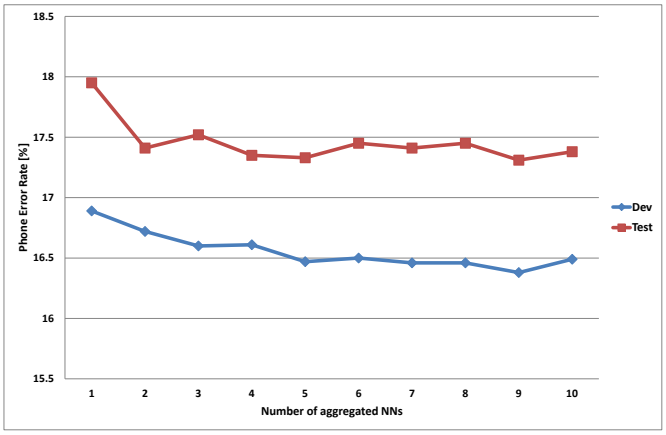

Fig. 1. Bagging PER for various number of NNs aggregated.

Table 1. Phone Error Rate [%] on TIMIT for various acoustic model NNs

NNs	Dev	Test
DNN	21.6	23.0
DBN_DNN_Dropout	18.2	19.7
DBN_DNN_Bagging_5	16.5	17.3
DBN_DNN_sMBR	16.4	17.1
DBN_DNN	16.3	17.0
DBN_DNN_Crogging_5	16.5	16.9
DBN_DNN_RPL$_{Full}$	16.4	16.8
DBN_DNN_RPL$_{Diag}$	16.2	16.5

the RPL with full- or diagonal- matrix W was added and trained. The results in Table 1 show that the DBN unsupervised pre-training is a key to get a low PER. Progress over *DBN_DNN* baseline was rather small. Sequence-discriminative training did not help on this task. A big surprise was the dropout result. Here the dropout results were unsatisfactory, and it was a problem to train the dropout NN in Kaldi successfully. However, other published more positive result with dropout on TIMIT [11], [16]. Bagging produced slightly higher PER that the *DBN_DNN* baseline. We also did an experiment with a change of a count of aggregated NNs in bagging method. We show the accurate bagging result in Figure 1. From the bagging analysis, it is clear that 5 NNs was enough for aggregating and more NNs did not brings any improvements. The greatest difference is between single NN and 2-NNs bagging. Crogging itself produced similar results as the baseline. We obtained an improvement with RPL only. With 1909 classes (tied-states of HMM), the diagonal RPL worked better than the full matrix variant, and we obtained PER close to the best published TIMIT core test results [16], [12].

5 Conclusion

In this paper, we propose the regularization post layer as an additional way to improve a deep neural networks generalization ability. It may be combined with other techniques as L1/L2 regularization, early stopping, dropout and others. It is based on cross-validation training and aggregation of NNs ensemble; therefore, the main drawback is higher computational requirements in the test phase. However, this drawback may be overcome by using of GPUs to accelerate NNs evaluation.

On TIMIT benchmark task, only a using of regularization post layer gives better results than DNN with DBN pre-training and we obtained PER close to the best published TIMIT core test results.

References

1. Barrow, D.K., Crone, S.F.: Crogging (Cross-Validation Aggregation) for Forecasting - A Novel Algorithm of Neural Network Ensembles on Time Series Subsamples. Proceedings of the International Joint Conference on Neural Networks (2013)
2. Breiman, L.: Bagging predictors. Machine learning 24(2), 123–140 (1996)
3. Girosi, F., Jones, M., Poggio, T.: Regularization theory and neural networks architectures. Neural Computation 7(2), 219–269 (1995)
4. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(10), 993–1001 (1990)
5. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation 18(7), 1527–1554 (2006)
6. Kang, G., Li, J., Tao, D.: Shakeout: A New Regularized Deep Neural Network Training Scheme. Proceedings of the AAAI Conference pp. 1751–1757 (2016)
7. Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P.: Exploring Strategies for Training Deep Neural Networks. Journal of Machine Learning Research (JMLR) 1, 1–40 (2009)
8. Perrone, M.P., Cooper, L.N.: When networks disagree: Ensemble methods for hybrid neural networks. Tech. rep., DTIC Document (1992)
9. Seltzer, M.L., Yu, D., Wang, Y.: An Investigation of Deep Neural Networks for Noise Robust Speech Recognition. Proceedings of the ICASSP pp. 7398–7402 (2013)
10. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout : A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research (JMLR) 15, 1929–1958 (2014)
11. Toth, L.: Combining Time- and Frequency- Domain Convolution in Convolutional Neural Network-Based Phone Recognition. Proceedings of the ICASSP (2014)
12. Tóth, L.: Convolutional Deep Maxout Networks for Phone Recognition. Proceedings of the INTERSPEECH pp. 1078–1082 (2014)
13. Vesely, K., Ghoshal, A., Burget, L., Povey, D.: Sequence-Discriminative Training of Deep Neural Networks. Proceedings of the INTERSPEECH pp. 2345–2349 (2013)
14. Wang, S.I., Manning, C.D.: Fast Dropout Training. Proceedings of the 30th International Conference on Machine Learning 28, 118–126 (2013)
15. Zhang, G.: Neural Networks for Classification: A Survey. IEEE Transactions on Systems, Man and Cybernetics) 30(4), 451–462 (2000)
16. Zhang, S., Bao, Y., Zhou, P., Jiang, H., Dai, L.: Improving Deep Neural Networks for LVCSR Using Dropout and Shrinking Structure. Proceedings of the ICASSP pp. 6899–6903 (2014)
17. Zhou, Z.H., Wu, J., Tang, W.: Ensembling Neural Networks: Many Could be Better than All. Artificial Intelligence 137(1-2), 239–263 (2002)