IMPROVING DEEP NEURAL NETWORKS FOR LVCSR USING RECTIFIED LINEAR UNITS AND DROPOUT

George E. Dahl ⋆ *Tara N. Sainath*† *Geoffrey E. Hinton*⋆

⋆ Department of Computer Science, University of Toronto
† IBM T. J. Watson Research Center, Yorktown Heights, NY 10598

ABSTRACT

Recently, pre-trained deep neural networks (DNNs) have outperformed traditional acoustic models based on Gaussian mixture models (GMMs) on a variety of large vocabulary speech recognition benchmarks. Deep neural nets have also achieved excellent results on various computer vision tasks using a random "dropout" procedure that drastically improves generalization error by randomly omitting a fraction of the hidden units in all layers. Since dropout helps avoid overfitting, it has also been successful on a small-scale phone recognition task using larger neural nets. However, training deep neural net acoustic models for large vocabulary speech recognition takes a very long time and dropout is likely to only increase training time. Neural networks with rectified linear unit (ReLU) non-linearities have been highly successful for computer vision tasks and proved faster to train than standard sigmoid units, sometimes also improving discriminative performance. In this work, we show on a 50-hour English Broadcast News task that modified deep neural networks using ReLUs trained with dropout during frame level training provide an 4.2% relative improvement over a DNN trained with sigmoid units, and a 14.4% relative improvement over a strong GMM/HMM system. We were able to obtain our results with minimal human hyper-parameter tuning using publicly available Bayesian optimization code.

Index Terms— neural networks, deep learning, dropout, acoustic modeling, broadcast news, LVCSR, rectified linear units, Bayesian optimization

1. INTRODUCTION

Up until a few years ago, most state of the art speech recognition systems were based on hidden Markov models (HMMs) that used mixtures of Gaussians to model the HMM emission distributions. However, [1] showed that hybrid acoustic models that replaced Gaussian mixture models (GMMs) with pre-trained, deep neural networks (DNNs) could drastically improve performance on a small-scale phone recognition task, results that were later extended to a large vocabulary voice search task in [2]. Since then, several groups have demonstrated dramatic gains from using deep neural network acoustic models on large vocabulary continuous speech recognition (LVCSR) tasks (see [3] for a recent review).

Even with unsupervised pre-training and large training sets, wide and deep neural networks are still vulnerable to overfitting. Dropout is a technique for avoiding overfitting in neural networks that has been highly effective on non-speech tasks and the small-scale TIMIT phone recognition task [4], although it can increase training time. Rectified linear units (ReLUs) in our experience achieve the same training error faster (as [5] also found) than sigmoid units and, although sometimes superior to sigmoid units, can often overfit more easily. Rectified linear units are thus a natural choice to combine with dropout for LVCSR.

In this paper, we explore the behavior of deep neural nets using ReLUs and dropout on a 50-hour broadcast news (BN) task [6], focussing our experiments on using dropout during the frame level training phase. We show that the modified deep neural networks (DNNs) using ReLUs and dropout provide a 4.2% relative error reduction over a standard pre-trained DNN and a 14.4% relative improvement over a strong GMM-HMM system.

The rest of this paper is organized as follows. In Section 2, we describe the dropout method, while in Section 3 Rectified Linear units (ReLU) are presented. Experiments and results are presented in Section 4. Finally, Section 5 concludes the paper and discusses future work.

2. DROPOUT

Dropout is a powerful technique introduced in [4] for improving the generalization error of large neural networks. In [5], dropout yielded gains on a highly competitive computer vision benchmark. Although [4] applied dropout to the 3 hour TIMIT phone recognition task, we are not aware of any application of dropout to LVSR.

Dropout discourages brittle co-adaptations of hidden unit feature detectors by adding a particular type of noise to the hidden unit activations during the forward pass of training. The noise zeros, or "drops out," a fixed fraction of the activations of the neurons in a given layer, similar to the type of noise recommended for the input of denoising auto-encoders in [7]. However, unlike in denoising autoencoder pre-training,

dropout is used in all hidden and input layers and occurs during supervised training with end-to-end backpropagation. Furthermore, since at test time dropout does not occur and there may not be additional training with it disabled, during training we multiply the net input from the layer below by a factor of $\frac{1}{1-r}$, where r is the dropout probability for units in the layer below. Specifically, to compute the activation $\mathbf{y}_t$ of the tth layer of the net during forward propagation, we use:

$$\mathbf{y}_t = f\left(\frac{1}{1-r}\mathbf{y}_{t-1} * \mathbf{m}\mathbf{W} + \mathbf{b}\right),$$

where f is the activation function for the tth layer, $\mathbf{W}$ and $\mathbf{b}$ are respectively the weights and biases for the layer, $*$ denotes element-wise multiplication, and $\mathbf{m}$ is a binary mask with entries drawn i.i.d. from *Bernoulli*$(1-r)$ indicating which activations are not dropped out. Dropout can also be viewed as training a very large number of different neural nets with different connectivity patterns and tied weights for all units that are not dropped out. By randomly sampling which units remain anew for each training case, this averaging can be performed in a particularly efficient way. The factor of $\frac{1}{1-r}$ used during training ensures that at test time, when all units get used, the correct total input will reach each layer.

Although dropout guards against overfitting and produces far more robust models, adding so much noise during training slows down learning, in our experience by about a factor of two. Since overfitting is much easier to avoid, larger models should be used to obtain the best results which also can slow down training although it enables better results.

3. RECTIFIED LINEAR UNITS

In this paper we use the term rectified linear unit (ReLU) to refer to unit in a neural net that use the activation function $\max(0, x)$. In computer vision research, ReLUs have been used both as activation functions in more standard neural nets and as units in restricted Boltzmann machines (RBMs), where they must be dealt with using approximations since they invalidate the probabilistic model [8, 9, 5]. To our knowledge, the only published use of these units in speech to date was in [10] where Gaussian-ReLU RBMs were used to learn features from raw, unlabeled acoustic data to later use in phone recognition experiments on TIMIT. Our work differs from [10] in that our focus is on LVSR, we exploit the synergistic effects of combining ReLUs with dropout, and we experiment with ReLUs at all hidden layers in our network instead of only using them as part of a stand-alone RBM feature extraction module.

Although ReLUs are quite simple to incorporate into a standard feed-forward neural net, in order to maintain the RBM probabilistic model we can view a single unit with the $\max(0, x)$ nonlinearity as an approximation to a set of replicated binary units with tied weights and shifted biases [9]. As in [9], we perform RBM pre-training with ReLU units by adding Gaussian noise with sigmoid variance (called an NReLU unit in [9]). In other words, during pre-training we sample a hidden unit state $y = \max(0, x + \epsilon)$, where x is the net input to the hidden unit and ϵ is drawn from a normal distribution with mean zero and variance $\frac{1}{1+e^{-x}}$. During fine-tuning, we simply use the $\max(0, x)$ nonlinearity.

4. EXPERIMENTS

4.1. Baseline

We performed all experiments on 50 hours of English Broadcast News [6]. We used the DARPA EARS `rt03` set for development/validation and performed final testing on the DARPA EARS `dev04f` evaluation set.

The GMM system is trained using the recipe from [11], which is briefly described below. The raw acoustic features are 19-dimensional PLP features with speaker-based mean, variance, and vocal tract length normalization. Temporal context is included by splicing 9 successive frames of PLP features into supervectors, then projecting to 40 dimensions using linear discriminant analysis (LDA). The feature space is further diagonalized using a global semi-tied covariance (STC) transform. The GMMs are speaker-adaptively trained, with a feature-space maximum likelihood linear (fMLLR) transform estimated per speaker in training and testing. Following maximum-likelihood training of the GMMs, feature-space discriminative training (fBMMI) and model-space discriminative training are done using the boosted maximum mutual information (BMMI) criterion. At test time, unsupervised adaptation using regression tree MLLR is performed. The GMMs use 2,203 quinphone states and 150K diagonal-covariance Gaussians.

The pre-trained deep neural net (DNN) systems use the same fMLLR features and 2,203 quinphone states as the GMM system described above, with an 11-frame context (± 5) around the current frame. The DNN consists of six hidden layers, each containing 1,024 sigmoidal units, and a softmax layer with 2,203 output targets. These features and DNN architecture were considered optimal in [12] for this Broadcast News task. The DNN training begins with greedy, layerwise, generative pre-training and then continues with cross-entropy discriminative training, followed by Hessian-free sequence-training [6]. To enable fair comparisons with the large nets we tried in this work, we also trained versions of the DNN baseline with 2048 and 3072 hidden units per layer, with 2048 units producing the best results for the basic pre-trained deep neural net setup.

4.2. Interactions Between Hessian Free Optimization and Dropout

Although dropout is trivial to incorporate into minibatched stochastic gradient descent (SGD), the best way of adding it to 2nd order optimization methods is an open research question. Although all frame level training we performed used

minibatched stochastic gradient descent, we decided to use the Hessian-free optimizer (HF) of [6] for all full sequence training in this work since it has large parallelization advantages over the SGD sequence training system we had available. However, one consequence of this choice is the undesirable interaction between dropout and HF. To find each search direction, HF uses the conjugate gradient (CG) algorithm to iteratively optimizes a local quadratic approximation to the objective function. CG depends on having reliable curvature and gradient information computed from large batches since it makes such strong use of it. Having a single fixed objective function during CG is even more important than the accuracy of the estimates. Dropout adds too much noise to the gradient for CG to function properly if the randomization occurs anew on each gradient evaluation. However, we do not yet know the best times to refresh the dropout pattern during HF so we simply disabled dropout during full sequence training.

4.3. Bayesian Optimization and Training Details

We used the Bayesian optimization method described in [13] to optimize training metaparameters and model hyperparameters on the validation set. Bayesian optimization is a gradient-free global optimization method that in the case of Snoek et al. models the unknown function from hyperparameters to validation word error rate as being drawn from a Gaussian process prior. As experiments complete, the algorithm updates the posterior distribution for the function and proposes new experiments to run that optimize the integrated (over Gaussian process covariance function hyperparameters) expected improvement below the current minimum error achieved by any experiment so far. We used public code released by the authors of [13] and enabled a constraint finding option to deal with divergent training runs. The constrained Bayesian optimization version has a separate Gaussian process model to classify points as feasible or not and combines this model with the Gaussian process regression model when suggesting new jobs.

All frame level training used the cross entropy criterion and minibatched stochastic gradient descent. Frame level training used gnumpy [14] and CUDAMat [15] to train using a GPU. We allowed Bayesian optimization to set twelve hyperparameters for frame level supervised training: seven dropout rates $r_0 \dots r_6$ (one per non-output layer with $0 \leq r_i \leq 0.5$), a single global initial learning rate $\alpha \in [4 \times 10^{-4}, 4 \times 10^{-1}]$, a single global momentum $\mu \in [0, 0.95]$, a number of times to anneal the learning rate before stopping $a \in [3, 11]$, a minimum improvement $m \in [0, 0.025]$ in validation cross entropy required to not anneal the learning rate, and a factor γ to divide the learning rates by when annealing them. We optimized $\log_2 \gamma$ on $[0.5, 2.5]$. Every half-epoch we computed validation cross entropy and tested to see if the learning rates should be annealed according to the schedule.

Since we did not rerun pre-training for each Bayesian optimization job, we set the pre-training metapamaters by hand. Training ReLU RBMs with CD1 can be more difficult than training Binary RBMs. We found it useful to only do 2.5 epochs of pre-training for each layer compared to more than twice that for sigmoid units.

The Bayesian optimization software we used is sufficiently effective to remove most of the human judgement from hyperparameter optimization. When there are only a few hyperparameters, it isn't hard to select reasonable values by hand, but optimizing more hyperparameters facilitates fairer comparisons between established models that researchers have experience tuning and new models they have less intuition about. However, there are still two areas that require human intervention: defining the search space and deciding when to terminate Bayesian optimization. Additionally, Bayesian optimization can be accelerated by seeding the optimization with jobs with hyperparameters selected using results from related hyperparameter search problems. For example, if one trains a neural net with a thousand units per hidden layer and finds good settings for its hyperparameters, these settings can initialize a Bayesian optimization run that searches for hyperparameters for a related network using two thousand hidden units per layer. We selected seed jobs for our larger nets based on hyperparameter optimizations for smaller networks.

4.4. Results

Table 1 shows word error rate results on the development (validation) set and test set before full sequence training was performed. Each model listed is the one that performed best on the validation data given the constraints in the model description. As shown in the table, the single best model uses ReLUs, dropout, and relatively large hidden layers with weights initialized using unsupervised RBM pre-training. This model achieved a word error rate of 18.5 on the test set, beating other deep neural net models and the strong discriminatively trained GMM baseline, even before any full sequence training of the neural net models.

To confirm that the unsupervised pre-training was worthwhile even with the approximations required for ReLU RBMs and the smaller number of pre-training epochs used relative to binary RBMs, we ablated the pre-training and optimized all remaining hyperparameters. Even without pre-training, the ReLU nets using dropout had about the same test set performance as the best baseline and somewhat better performance than the best pre-trained sigmoid nets. For the 2k unit per hidden layer, un-pre-trained ReLU nets, we also added additional hyperparameters (layer specific learning rates and weight costs) and ran extensive Bayesian optimization experiments to confirm that we could not find a configuration with lower validation errors even in the larger hyperparameter search space. Since these experiments alone involved over

Model	rt03	dev04f
ReLUs, 3k, dropout, CE	10.7	18.5
no PT, ReLUs, 3k, dropout, CE	11.0	18.9
no PT, ReLUs, 2k, dropout, CE	11.2	19.0
Sigmoids, 3k, CE	11.3	19.4
Sigmoids, 2k, CE	11.1	19.4
Sigmoids, 1k, CE	11.6	19.9

Table 1. Results without full sequence training (with cross entropy a.k.a CE). All models used pre-training unless "no PT" was specified and used 1k,2k, or 3k hidden units per layer.

Model	rt03	dev04f
GMM baseline	10.8	18.8
ReLUs, 3k, dropout, sMBR	9.6	16.1
Sigmoids, 2k, sMBR	9.6	16.8

Table 2. Results with full sequence training

5000 GPU hours of computation, they were not feasible to perform on the nets with 3k units per layer, although for those nets we performed our usual hyperparameter optimization in the search space described in section 4.3.

The Bayesian optimization procedure rarely turned off dropout for ReLU nets and typically kept dropout off for sigmoid nets, even with 3k units per layer. This indicates that the Bayesian optimization procedure learned that dropout wasn't helpful for sigmoid nets of the sizes we trained. In general, ReLUs and dropout seem to work quite well together. Without dropout, ReLU nets can overfit very quickly compared to sigmoid nets. Early stopping is often able to avoid the worst overfitting in sigmoid nets on LVSR tasks, but for ReLU nets, dropout combined with early stopping worked much better.

Table 2 shows results for the best ReLU net and the best sigmoid net was trained for an equal amount of wall clock time using 11 and 16 HF iterations of full sequence training respectively. We trained the ReLU and sigmoid nets for an equal amount of time to make a fair comparison in the absence of dropout during full sequence training and because of the expense of full sequence training with nets of this size. We expect that the very best results will necessitate the use of dropout during full sequence training as well and we plan to explore this in future work. The ReLU net shows a more modest gain over the sigmoid net after full sequence training than was evident after frame level training, possibly because the ReLU net no longer gains the benefit of training with dropout during the full sequence training. Full sequence training for nets as large as these is very time consuming and will most likely slow down further if HF is modified to support dropout and especially if gradient descent sequence training gets used instead.

5. CONCLUSIONS AND FUTURE WORK

In this paper, motivated by successes in computer vision, we have explored using rectified linear units and dropout in deep neural nets for LVCSR for the first time. ReLUs and dropout yielded word error rate improvements relative to a state of the art baseline and, with the help of Bayesian optimization software, we were able to obtain these improvements without much of the hand tuning typically used to obtain the very best deep neural net results. These Bayesian optimization experiments also gave us evidence that ReLUs and dropout have synergistic effects and we recommend that they be used together. Although in our experiments we did not detect a large gain from using dropout with sigmoid nets, on other tasks or with other network architectures dropout might provide a larger benefit.

Given the gains from dropout and ReLUs for frame level training, determining the best way to exploit dropout in full sequence training is an exciting direction for future work. At least in principle, dropout can be used with sequence training performed with gradient descent, but there might be better options. The dropout noise can be analytically integrated out for single layer nets, but this is unlikely to be possible for deeper nets without some sort of aggressive approximation. Fixing the units that are dropped out during each CG run within HF might be sufficient to allow dropout to be combined with HF, but to date all uses of dropout in the literature have been with first order optimization algorithms. Extracting bottleneck features [16] using deep nets trained with dropout would obviate most of the need for sequence training of the nets, but would abandon the goal of end-to-end training using a sequence level criterion. Nevertheless, it is a promising alternative we hope to explore.

Acknowledgements

We would like to thank Hagen Soltau, Brian Kingsbury, George Saon and Stanley Chen for their contributions towards the IBM toolkit and recognizer utilized in this paper. We would also like to thank Jasper Snoek for helping us use the Bayesian optimization software.

6. REFERENCES

[1] Abdel rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton, "Acoustic modeling using deep belief networks," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 14 –22, jan. 2012.

[2] Geore E. Dahl, Dong Yu, Li Deng, and Alex Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30 –42, jan. 2012.

[3] Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *Signal Processing Magazine*, 2012.

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

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Neural Information Processing Systems*, 2012.

[6] Brian Kingsbury, Tara N. Sainath, and Hagen Soltau, "Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization," in *Proc. Interspeech*, 2012.

[7] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, William W. Cohen, Andrew McCallum, and Sam T. Roweis, Eds. 2008, pp. 1096–1103, ACM.

[8] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun, "What is the best multi-stage architecture for object recognition?," in *Proc. International Conference on Computer Vision (ICCV'09)*. 2009, pp. 2146–2153, IEEE.

[9] Vinod Nair and Geoffrey E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, Johannes Fürnkranz and Thorsten Joachims, Eds., Haifa, Israel, June 2010, pp. 807–814, Omnipress.

[10] Navdeep Jaitly and Geoffrey E. Hinton, "Learning a better representation of speech soundwaves using restricted Boltzmann machines," in *ICASSP*, 2011, pp. 5884–5887.

[11] Hagen Soltau, George Saon, and Brian Kingsbury, "The IBM Attila Speech Recognition Toolkit," in *Proc. SLT*, 2010.

[12] Tara N. Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and Abdel rahman Mohamed, "Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition," in *Proc. ASRU*, 2011.

[13] Jasper Snoek, Hugo Larochelle, and Ryan Prescott Adams, "Practical Bayesian optimization of machine learning algorithms," in *Neural Information Processing Systems*, 2012.

[14] Tijmen Tieleman, "Gnumpy: an easy way to use GPU boards in Python," Tech. Rep. UTML TR 2010-002, University of Toronto, Department of Computer Science, 2010.

[15] Volodymyr Mnih, "Cudamat: a CUDA-based matrix class for python," Tech. Rep. UTML TR 2009-004, Department of Computer Science, University of Toronto, November 2009.

[16] Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran, "Auto-Encoder Bottleneck Features Using Deep Belief Networks," in *Proc. ICASSP*, 2012.