ON RECTIFIED LINEAR UNITS FOR SPEECH PROCESSING

M.D. Zeiler[1], M. Ranzato[2], R. Monga[2], M. Mao[2], K. Yang[2], Q.V. Le[2],*
P. Nguyen[2], A. Senior[2], V. Vanhoucke[2], J. Dean[2], G.E. Hinton[3]

[1]New York University, USA [2]Google Inc., USA [3]University of Toronto, Canada

ABSTRACT

Deep neural networks have recently become the gold standard for acoustic modeling in speech recognition systems. The key computational unit of a deep network is a linear projection followed by a point-wise non-linearity, which is typically a logistic function. In this work, we show that we can improve generalization and make training of deep networks faster and simpler by substituting the logistic units with rectified linear units. These units are linear when their input is positive and zero otherwise. In a supervised setting, we can successfully train very deep nets from random initialization on a large vocabulary speech recognition task achieving lower word error rates than using a logistic network with the same topology. Similarly in an unsupervised setting, we show how we can learn sparse features that can be useful for discriminative tasks. All our experiments are executed in a distributed environment using several hundred machines and several hundred hours of speech data.

Index Terms— Rectified Linear units, Deep Learning, Neural Networks, Unsupervised Learning, Hybrid System

1. INTRODUCTION

Recent years have seen a surge of interest in neural networks for acoustic modeling in speech recognition systems. Compared to traditional Gaussian Mixture Models (GMMs), neural networks have two main advantages. They scale better with the input dimensionality allowing the use of larger context windows and they automatically learn discriminative features from data alleviating the problem of manually engineering and selecting features. Together these two factors have yielded dramatic improvements in terms of word error rate.

In their seminal work, Mohamed et al. [1] proposed to use a system composed of many layers of logistic units. In order to overcome the notoriously difficult problem of optimizing very deep networks, they proposed to use a layer-wise unsupervised learning algorithm, called Restricted Boltzmann Machine (RBM) [2], as a way to provide a sensible initialization and they demonstrated significant improvements over the baseline GMM.

One issue with this training procedure is that tracking convergence of RBMs is difficult and the overall layer-wise training procedure is laborious and time-consuming, even when using specialized hardware like GPUs. This challenge continued to motivate researchers to design better unsupervised algorithms [3, 4] and better optimization methods for training deep neural nets [5, 6].

Inspired by recent work on deep learning for vision applications [7, 8, 9], we propose to replace the logistic non-linearity with a half-rectification non-linearity which is linear for positive values and zero otherwise. Because of the shape of this non-linearity, we call the resulting deep network a "hinge deep neural network" (HDNN), and the units that compose the HDNN "rectified linear units" (ReLUs) [7].

This small change brings several advantages. First, in our experience it eliminates the necessity to have a "pretraining" phase using unsupervised learning [8]. We demonstrate empirically that we can easily and successfully train extremely deep networks even from random initialization. Second, the convergence of HDNN is faster than in a regular logistic neural net with the same topology. Third, HDNN is very simple to optimize. Even vanilla stochastic gradient descent with constant learning rate yields very good accuracy. Fourth, HDNN generalizes better than its logistic counterpart. And finally, rectified linear units are faster to compute because they do not require exponentiation and division, with an overall speed up of 25% on the 4 hidden layer neural network we tested on.

We conjecture that the reason why rectified linear units are so beneficial for efficient learning of deep neural nets is twofold. First, from the optimization perspective, HDNN is piece-wise linear. If we restrict our attention to the units that are non-zero, the whole system reduces to a linear convex system whose optimization is straightforward even using first order optimizers. Second, HDNN seems to generalize better because the internal representation produced by the network is much more regularized. Unlike logistic units that produce small positive values when the input is not aligned with the internal weights, rectified linear units often output exact zeros. For instance, we found that on average about 80% of the units in a HDNN are zero after training. Improved generalization can be seen as the effect of the increased sparsity of the internal representation or also by interpreting HDNN as a system with stacked binary linear SVMs (as opposed to logistic regression classifiers), and it is well known that such

*This work was done while M.D. Zeiler was an intern at Google.

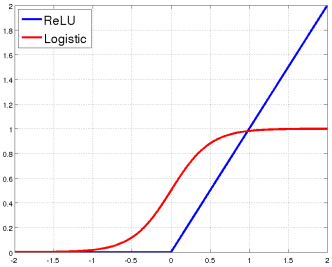

Fig. 1. The proposed non-linearity, ReLU, and the standard neural network non-linearity, logistic.

classifiers enjoy better generalization properties [9].

Although our work does not reveal any benefit to unsupervised learning for HDNNs, it is still interesting to learn features without any supervision for the purpose of automatic discovery of phonetic elements and potentially, for training on languages with small amounts of labeled data. We propose a very simple sparse autoencoder method that can learn very interpretable and discriminative features when using ReLUs.

Overall, we demonstrate a clear advantage of ReLUs over logistic units both in the supervised and unsupervised setting. Our empirical validation uses a recently introduced distributed framework [10]. This allows us to train a network with 43 million parameters on four hundred machines using 1500 cores to process 1.2 billion frames per day, more than 6 times faster than using a single NVIDIA GeForce GTX 580 GPU.

2. SUPERVISED LEARNING

Our supervised learning set up is conventional, except for the use of the proposed activation function. The network is given both an input $\mathbf{x}$ (typically a few consecutive frames of a spectrogram representation) and a label representing the state of the HMM for that input. The network processes the input through a sequence of non-linear transformations. In particular, at the i-th layer the network computes: $\mathbf{h}^i = f(W^i\mathbf{h}^{i-1} + \mathbf{b}^i)$, where $W^i \in \mathbb{R}^{M\times N}$ is a matrix of trainable weights, $\mathbf{b}^i \in \mathbb{R}^M$ is a vector of trainable biases, and $\mathbf{h}^i \in \mathbb{R}^N$ is the i-th hidden layer (or the input $\mathbf{x}$ if i is equal to 0) and $\mathbf{h}^{i+1} \in \mathbb{R}^M$ is the $(i+1)$-th hidden layer.

In our work, we propose to use as f the following pointwise non-linear function: $f(u) = \max(0, u)$. The resulting unit in the network is dubbed ReLU [7]. We have also experimented and compared to other functions as well. We tested the widely used *logistic*, $f(u) = 1/(1+\exp(-u))$, and *hyperbolic tangent*, $f(u) = \tanh(u)$. Since hyperbolic tangent performed slightly worse than logistic, we only report the latter for our baseline comparisons.

In order to predict the label, the topmost layer of the network uses a softmax non-linearity which outputs probability values. If the network has L layers, then the prediction for the probability of the k-th class is: $p(k|\mathbf{x}) = \exp((W^L)_k\mathbf{h}^{L-1} + b^L_k)/\sum_{j=1}^C \exp((W^L)_j\mathbf{h}^L + b^L_j)$, where $(W^L)_j$ is j-th row of the last layer weight matrix, b^L_j is the j-th entry in the last layer vector of biases and C is the number of classes.

Training the parameters of the network (the weight matrices and biases at all layers) is performed by minimizing the cross entropy loss over the training set. The contribution of each sample $\mathbf{x}$ to the loss is: $L^{\text{sup}}(\theta) = -\sum_{j=1}^C \mathbf{t}_j \log p(j|\mathbf{x};\theta)$, where $\mathbf{t}$ is a 1-of-C encoding of the target class label and θ collectively denotes all parameters of the neural network, namely $\{(W^i, \mathbf{b}^i), i = 0, \ldots, L\}$. We will discuss in sec. 4 how we minimize this loss function.

3. UNSUPERVISED LEARNING

In our unsupervised experiments, we use a method described in appendix B of [11]. This is a very efficient and effective sparse feature learning method which can be understood as a sparse auto-encoder neural network using ReLU units as features. Training proceeds layer by layer, from the bottom to the top in sequence. For each layer, the features (that will be subsequently used as data to train the layer above) are: $\mathbf{h}^i = \max(0, W^i\mathbf{h}^{i-1} + \mathbf{b}^i)$. During training we couple this layer with an auxiliary layer that reconstructs the input from the features using $\hat{\mathbf{h}}^{i-1} = \max(0, W^i_r\mathbf{h}^i + \mathbf{b}^i_r)$, where $W^i_r \in \mathbb{R}^{N\times M}$ is a matrix of trainable weights, $\mathbf{b}^i_r \in \mathbb{R}^N$ is a vector of trainable biases, and $\hat{\mathbf{h}}^{i-1} \in \mathbb{R}^N$ is the reconstruction of $\mathbf{h}^{i-1}$. When i is 0 and we reconstruct the input $\mathbf{x}$, the ReLUs units of the reconstruction layer are replaced by linear units because $\mathbf{x}$ takes also negative values. The parameters are learned by minimizing a loss function. The contribution of each sample to the loss at the i-th layer is: $L^{\text{unsup}}(\theta) = ||\hat{\mathbf{h}}^{i-1} - \mathbf{h}^{i-1}||^2_2 + \lambda||\mathbf{h}^i||_1$, with $\lambda \geq 0$. The first term measures the squared reconstruction error and it is useful to guarantee that features preserve information of the input. The latter term makes the learning algorithm discover sparse features, that is, features with few non-zero values. This is important to restrict the capacity of the model (overall when there are more features than input dimensions) and force it to capture the regularities of the input data. Since the loss can be trivially decreased by scaling down W^i while scaling up W^i_r, we re-parameterize W_r as follows: $(W^i_r)_j = (\tilde{W}^i_r)_j/||(\tilde{W}^i_r)_j||_2$, where $(W^i_r)_j$ is the j-th column of W^i_r, and we learn the parameters of matrix $\tilde{W}^i_r$.

This method can be interpreted as a special case of PSD [12, 13] and sparse coding [14] when inference of features is computed in just one step. Unlike RBM's objective function which is intractable, this method can be optimized very efficiently and it enjoys the use of ReLU units because they naturally produce sparse features.

4. LEARNING IN A DISTRIBUTED FRAMEWORK

In order to support training on vast amount of data in very short time, we use our recently proposed distributed framework [10]. The hidden units of the network are partitioned across several machines and each machine further parallelizes

computation across several cores. Parallel distributed computation is used across the samples in a mini-batch as well as across the nodes of the neural network.

In the experiments of sec. 5 we use this framework and learn the parameters of the system by *asynchronous* stochastic gradient descent (SGD) [10]. Training proceeds as follows. The network is replicated P times. Each replica is an exact copy of the model, with possibly slightly stale parameters and operating on a random subset of the training data. Besides the P replicas, there is also a sharded parameter server hosting the most updated version of the parameters. Once a model replica has finished computing the gradients on its mini-batch, it sends them to the parameters server which uses them to update the parameters. Finally, the parameter server sends back an updated copy of the parameters to that model replica. This mechanism allows many model replicas to work concurrently but asynchronously on the same training problem and to quickly update the parameters, while being tolerant to machine failure and high latency.

In this work, we investigated three different ways to update the parameters in the parameter server. Let θ_i^t be the i-th parameter after $t-1$ weight updates. In vanilla SGD the parameters are updated using: $\theta_i^{t+1} = \theta_i^t - \eta \partial L / \partial \theta_i^t$, where η is the learning rate. In SGD with Adagrad [10, 15], each parameter has its own adaptive learning rate; the parameter update is $\theta_i^{t+1} = \theta_i^t - \eta_i^t \partial L / \partial \theta_i^t$ with $\eta_i^t = \eta / \sqrt{\sum_{s=1}^{t} (\partial L / \partial \theta_i^s)^2}$. Finally, in SGD with momentum the parameters are updated by: $\theta_i^{t+1} = \theta_i^t - \eta \Delta_i^{t+1}$, with $\Delta_i^{t+1} = 0.9 \Delta_i^t + \partial L / \partial \theta_i^t$. While Adagrad aims at gently scaling and annealing learning rates, momentum speeds up learning along those gradient directions that are persistent during training.

5. EXPERIMENTS

All experiments are performed using several hundred hours of US English data collected using Voice Search, Voice Typing and read data. The test set follows the same distribution of the training set but uses independent sources. The setup for the hybrid decoding is exactly the same as the one described in earlier work [16].

In the supervised setting, a baseline GMM-HMM system is trained and used to generate 7969 context-dependent tied acoustic states. This system is also used to produce state labels for every input frame using forced alignment. These labels are the target for the supervised network.

In both the supervised and unsupervised settings, the input to the network consists of 26 consecutive frames, each comprising 40 log-energy filter bank outputs representing 25ms of speech. Consecutive frames are 10ms apart. The overall input dimensionality is 1040, although spectral analysis reveals that 95% of the variance is concentrated in the first leading 100-dimensional principal components.

All layers of our networks have 2560 hidden units and training has been performed by partitioning each network

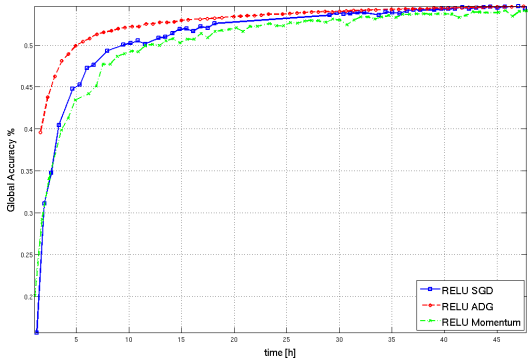

Fig. 2. Frame accuracy as a function of time of a 4 hidden layer HNN trained with different optimizers.

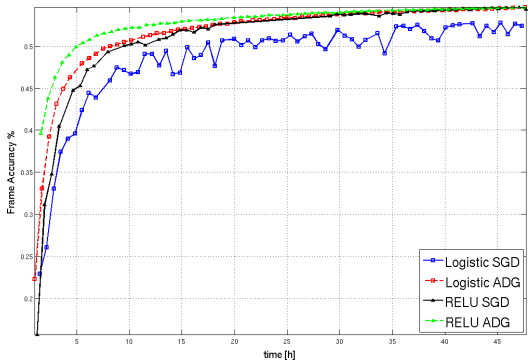

Fig. 3. Frame accuracy as a function of time for a 4 hidden layer neural net trained with either logistic or ReLUs and using as optimizer either SGD or SGD with Adagrad (ADG).

across 4 machines using up to 4 CPUs each. The number of model replicas P has been set to 100. All parameters in the weight matrices are intialized at random while the biases are initialized at zero. Learning rates have been cross-validated. Typically, HDNN uses a learning which is 10 times smaller than a logistic DNN.

5.1. Supervised Learning Experiments

The results we report are obtained by training for one week. In the first experiment shown in fig. 2, we compare the three different optimization strategies we described in sec. 4, on a 4 hidden layer HNN initialized at random. In terms of wall clock time to reach a given frame accuracy on the validation set, Adagrad exhibits the fastest convergence time, although plain SGD eventually reaches the same overall frame accuracy. Momentum instead performs slightly worse.

Similar findings were observed using a network with logistic units. However, plain SGD does not perform as well as Adagrad in this case, see fig. 3. Unlike HNN, a logistic network does need accelerated first order methods to yield good frame accuracy. It seems that optimization is much harder in logistic networks than HNNs. Fig. 3 shows that a logistic network trained with Adagrad can achieve the same accuracy than a HNN trained with either Adagrad or even plain SGD. However, we found that the performance in terms of

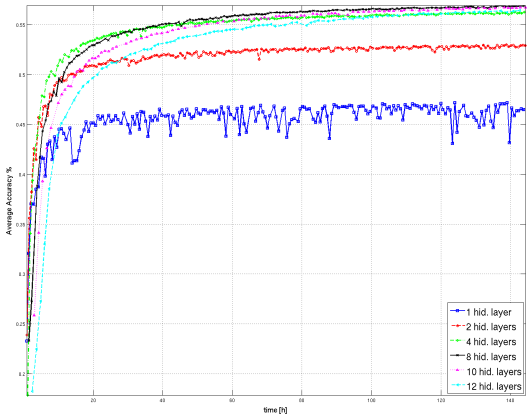

Fig. 4. Validation frame accuracy over time using HNN with different number of hidden layers and SGD.

Nr. hid. layers	1	2	4	8	10	12
WER %	16.0	12.8	11.4	10.9	11.0	11.1

Table 1. Word error rate of HNN with varying number of hidden layers.

word error rate is superior when using HNN and SGD. The word error rate of a logistic network trained with Adagrad is 11.8% (slightly better than when training using SGD), while the word error rate of HNN is 11.7 and 11.4% when using Adagrad and SGD, respectively. Since a difference of 0.1% is statistically significant in our data set, we conclude that HNNs are not only easier to train but also they generalize better.

Finally, fig. 4 shows that extremely deep HNNs (we tested up to 12 hidden layers) can be successfully trained from random initialization. Since we did not allocate more resources for the deeper networks, their compute and convergence time is slower. However, they do not get stuck in the optimization and produce among the best results. Table 1 reports the corresponding word error rates on the test set. The 8 hidden layer neural network produces the best rate of 10.9%, but this is closely followed by the 10 and 12 hidden layer HNN.

We also tested very deep logistic networks from random initialization but did observe that the optimization gets stuck when using 8 hidden layers and more. After one week, 8 hidden layers logistic network achieves a mere word error rate of 12.0%.

5.2. Unsupervised Learning Experiments

To validate the use of ReLUs for unsupervised learning, we trained using the same input data as in the previous section (but without making use of the labels). After learning, we first inspected the learned features. Fig. 5 shows a subset of the 2560 features where each tile corresponds to the weights connected to a hidden unit. Some features resemble Gabor functions localized in the time-frequency domain, but others are more complex and seem to capture the structure and the temporal dynamics of formants. We also tested the use of logistic units and linear units, but could not learn any structure set of features. To better interpret the ReLU features, we looked for the best matching input sample in the validation set. Leftmost part of fig. 5 shows how some filters resemble closely actual inputs, suggesting that some of these features may have discovered phonetic elements in an unsupervised way. To validate this hypothesis, we used each single feature as a threshold classifier and checked whether its output correlates with the phone label of the input. Table 2 shows that this is indeed the case for some phones. Since the large fraction of frames has label "silence", there is a very large number of negative inputs and achieving a precision of 50% at a recall greater than 1% is considered remarkable.

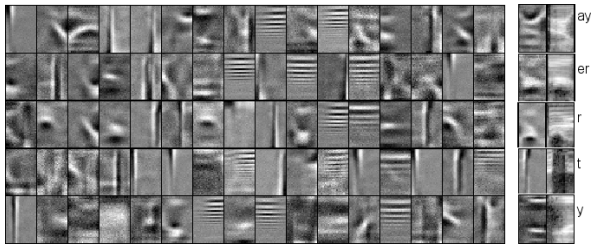

Fig. 5. Left: Random subset of the 2560 filters learned in an unsupervised way. The vertical axis is over frequencies and horizontal axis is over time (26 frames, each 10ms long). Right: Example of how some filters (left part) match samples on a validation set (right part with corresponding phone label).

Phone label	Precision %	Recall %	Accuracy %
er	57.0	2.0	98.5
iy	49.6	11.0	96.7
r	52.5	6.0	97.2

Table 2. Unsupervised discovery of phones: a threshold on a single feature can be a high accuracy phone detector.

Nr. hid. layers	0	1	2	3
Frame accuracy %	28.5	37.8	38.3	39.2

Table 3. Test frame accuracy using a linear classifier on the features learned in an unsupervised way.

The discrimination ability of ReLUs is expected to improve when we consider the whole feature set. We therefore trained a linear logistic regression classifier on the whole feature vector. Table 3 reports the frame accuracy as we learn more layers of features and demonstrates that features do get more discriminative as we stack them, although with diminishing returns. Using these features to initialize a deep HNN did not improve performance, however. We observed faster initial convergence but not better accuracy after a few hours of training.

6. CONCLUSION

In this empirical study we advocate the use of ReLU in deep networks since a) they are easier to optimize, b) they converge faster, c) they generalize better and d) they are faster to compute. Future work will leverage unsupervised learning and ReLUs for tasks where labeled data is very scarce.

7. REFERENCES

[1] A.R. Mohamed, G.E. Dahl, and G.E. Hinton, "Deep belief networks for phone recognition," NIPS 22 workshop on deep learning for speech recognition, 2009.

[2] G.E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, pp. 1771–1800, 2002.

[3] C. Plahl, T.N. Sainath, B. Ramabhadran, and D. Nahamoo, "Improved pre-training of deep belief networks using sparse encoding symmetric machines," in *ICASSP*, 2012.

[4] T.N. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-encoder bottleneck features using deep belief networks," in *ICASSP*, 2012.

[5] B. Kingsbury, T.N. Sainath, and H. Soltau, "Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization," in *Interspeech*, 2012.

[6] L. Deng, B. Hutchinson, and D. Yu, "Parallel training of deep stacking networks," in *Interspeech*, 2012.

[7] V. Nair and G.E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[8] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 15, pp. 315–323, 2011.

[9] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large scale distributed deep networks," in *NIPS*, 2012.

[11] M. Ranzato, "Unsupervised learning of feature hierarchies," Ph.D. thesis, ch. 1, 2009.

[12] K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "Fast inference in sparse coding algorithms with applications to object recognition," Tech. Rep., Computational and Biological Learning Lab, Courant Institute, NYU, 2008, Tech Report CBLL-TR-2008-12-01.

[13] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *ICML*, 2010.

[14] C.J. Rozell, D.H. Johnson, Baraniuk R.G., and B.A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, 2008.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online leaning and stochastic optimization," in *COLT*, 2010.

[16] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *Interspeech*, 2012.