PHONE RECOGNITION WITH DEEP SPARSE RECTIFIER NEURAL NETWORKS

László Tóth

MTA-SZTE Research Group on Artificial Intelligence
Hungarian Academy of Sciences and University of Szeged
tothl@inf.u-szeged.hu

ABSTRACT

Rectifier neurons differ from standard ones only in that the sigmoid activation function is replaced by the rectifier function, $max(0, x)$. This modification requires only minimal changes to any existing neural net implementation, but makes it more effective. In particular, we show that a deep architecture of rectifier neurons can attain the same recognition accuracy as deep neural networks, but without the need for pre-training. With 4-5 hidden layers of rectifier neurons we report 20.8% and 19.8% phone error rates on TIMIT (with CI and CD units, respectively), which are competitive with the best results on this database.

Index Terms— Deep neural networks, sparse rectifier neural networks, phone recognition

1. INTRODUCTION – RELATION TO PRIOR WORK

Neural networks have long been present in speech recognition [1]. However, researchers almost exclusively used networks with only one hidden layer, and they increased the size of this layer when hoping for increased performance. It has been realized only recently that adding more hidden layers can be more efficient. However, the conventional backpropagation algorithm has difficulties with training this kind of a deep architecture [2]. As a solution, Hinton et al. presented a pre-training algorithm that works in an unsupervised fashion [3]. After pretraining, the backpropagation algorithm can find a much better local optimum even with 7-8 hidden layers [4].

Since their invention, a lot of effort has been made to scale up deep networks to much larger datasets and large vocabulary tasks [5, 6, 7]. Even when implementing the algorithms on a GPU, pre-training can take up a considerable amount of time. For example, Yu et al. mention 62 hours of pre-training and 17 hours of fine-tuning for a 24-hour training data set [6]. Thus, any method that can simply skip pre-training without sacrificing performance is worth examining.

This publication was supported by the European Union and co-funded by the European Social Fund. Project title "Telemedicine-focused research activities in the fields of mathematics, informatics and medical sciences", project number: TÁMOP-4.2.2.A-11/1/KONV-2012-0073.

Here, we study one such promising alternative: deep sparse rectifier neural networks. *Compared to conventional neural nets, they differ only in the type of the activation function used.* However, this slight modification seems to enable them to learn deep structures more efficiently than standard neural nets. *They were proposed not long ago, and, to our knowledge, have thus far been employed only for image recognition and NLP tasks* [8]. To justify their applicability to speech recognition as well, here we evaluate them on the classic TIMIT phone recognition task.

2. RECTIFIER NEURAL NETS

Rectifier neural units were recently successfully applied in standard neural networks [8], and they were also found to improve Restricted Boltzmann Machines [9]. Conventional neural nets and rectifier neural nets differ in only one fundamental respect; namely, the type of activation function used: instead of the usual sigmoid or hyperbolic tangent activation, we apply the rectifier function $max(0, x)$ for the hidden neurons. Fig.1. illustrates the differences between the rectifier and the $tanh$ activation functions (the more popular sigmoid is just a scaled and shifted version of $tanh$).

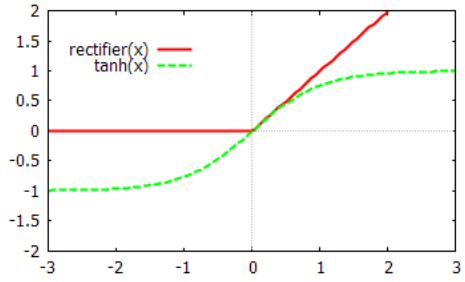

Fig. 1. A comparison of the $tanh$ and the rectifier activations.

As can be seen from comparing the two functions, there are two fundamental differences. One is that the output of rectifier neurons do not saturate as their activity gets higher. We think that this is very important in explaining their good performance in deep architectures: because of this linearity, their is no gradient vanishing effect. The other difference is the hard saturation at 0 for negative activity values. This has

the effect that for a given input only a subset of neurons are active. One may hypothesize that this may harm optimization by blocking gradient backpropagation, but the experimental results do not support this hypothesis: it seems that the hard nonlinearities do no harm as long as the gradient can propagate along some paths [8]. One would also expect that the initialization is crucial in this respect. Fortunately, the method is not as sensitive to the initialization as one might think. A closely related intriguing result that may give an explanation is by Vinyals and Deng [10]. They propose a sparse coding of the spectral vectors x_i by using the formula

$$max(0, D^T x_i - \alpha),$$

where the vectors of D^T form a coding dictionary, and α controls the amount of sparsity introduced. Notice that this is almost exactly what a layer of rectifier neurons compute: the elements of D^T correspond to the weights and α corresponds to the bias. The only difference is that we allow a separate bias for each weight vector. The most important observation of [10] is that a random choice of the dictionary is almost as good as using a sophisticated method. Indeed, we found that after randomly initializing the weights and tuning only the biases, learning starts from a much lower error rate than is usual for sigmoid nets. It requires more study to see whether initializing the weights with some dedicated method (such as the OMP1 algorithm proposed in [10]) instead of doing it randomly can yield improved results and/or faster convergence.

Another conclusion of [10] is that applying a simple "shallow" learner to the sparse-coded data is not as efficient as some deep learner. Rectifier neural nets offer a simple way of deep learning by stacking sparse coders on each other and then fine-tuning the resulting deep structure by backpropagation. Notice that deep rectifier nets are also computationally cheaper than conventional nets, as the exponential in the activation function does not have to be calculated. Based on our measurements, the computation of the sigmoid activation can take 10-15% of the processing time in a non-GPU (e.g. Math Kernel Library) implementation, and some authors take great pains to optimize it even in a GPU environment [11].

2.1. Potential Problems

Thus far we have talked only about the advantages of rectifier nets. There is, however, a potential problem that may arise: the activations might grow without limit, due to the unbounded nature of the activation function. To handle this, Glorot et al. propose to scale the activations to bound them between 0 and 1 [8]. We decided to scale the weights instead, so that the $L1$ norm of each layer remains the same as it was after initialization. What makes this possible is that for a given input the subset of active neurons behaves linearly, so a scaling of the weights is equivalent to a scaling of the activations (of course, apart from the final softmax layer). As a possible alternative, we also found that weight decay provides an equally good performance, and it is a standard technique in neural networks [12]. Notice that weight decay is not much different from our solution, as it also consists of weight scaling, but the scaling factor is controlled by a meta-parameter, and not by the $L1$ norm of the weights. Also, the computational cost of both methods is much smaller than what we gain from the omission of the sigmoid activations.

2.2. Improvement by Enforcing Sparsity

There is a consensus that enforcing the sparsity of a representation can improve the classification performance (see, e.g. [8, 13, 14, 7]). The rectifier activation function inherently induces sparsity. For example, after a random initialization of the weights, about 50% of the units are already inactive for an average sample, and this ratio can be easily increased by introducing a sparsity-inducing penalty term on the activations during training. That is, the target function will look like

$$E + \lambda \sum_{j=1}^{N} \rho(a_j),$$

where E is the standard cross-entropy cost, and $\rho(a_j)$ is the sparsity penalty on activation a_j. Two examples of a proper penalty term are the $L1$ penalty, as recommended by Glorot [8], and the $log(1+a_j^2)$ function, as used by Sivaram [14]. We experimented with both, and found no significant difference between them. The results reported were all obtained with the latter. In each case, the penalty term is applied to all the hidden neurons (that is, rectifier neurons), and not just to the lowest layer, as in [14]. Also, thanks to the rectifier function, the reported sparsity values mean the ratio of true zeros in the activations, and not a calculated sparsity measure, as in [14].

When applying sparsity, the meta-parameter λ has to be tuned. While experimenting with its value, we found that enforcing sparseness from the beginning of the training is harmful. The sparseness penalty proved to be the most effective when turned on only after a couple of epochs, when the weights are relatively stable. This is in accordance with the findings of [7]. In the results reported, λ was set to 0.001, and sparsity was turned on after 7-9 iterations through the training data.

3. EXPERIMENTAL SETTINGS

The results reported are phone recognition errros on the well-known TIMIT database. The training set consisted of the standard 3696 'si' and 'sx' sentences, while testing was performed on the core test set (192 sentences). A random 10% of the training set was held out for validation purposes, and for tuning the meta-parameters. We will refer to this block of the data as the 'development set'. All the experiments use a phone bigram language model estimated from the training data. The 61 phone labels are mapped to the usual set of 39

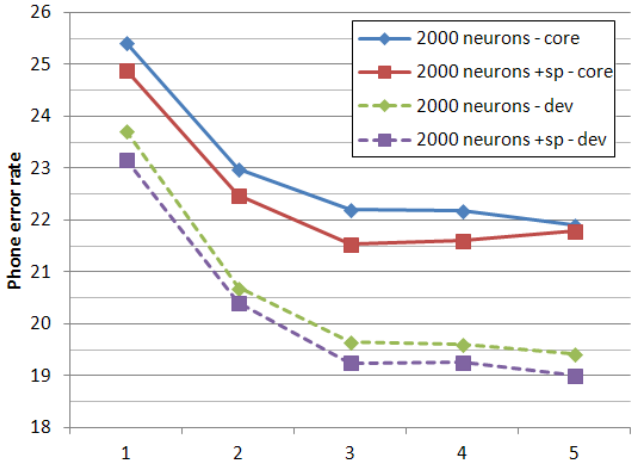

Fig. 2. Phone error rate as a function of the number of hidden layers for MFCC features. Results are shown for the development and the core test sets, with and without sparsity penalty.

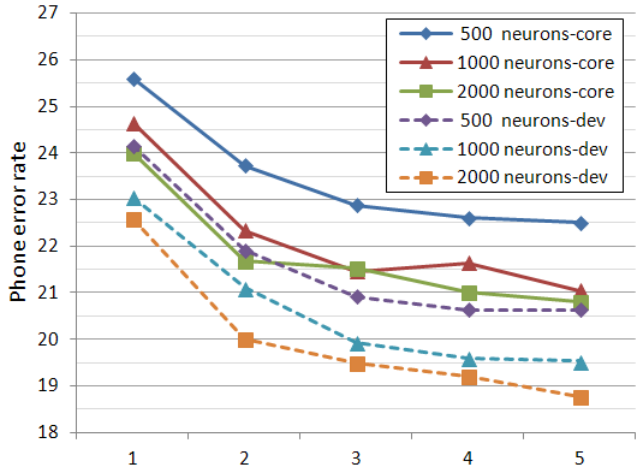

Fig. 3. Phone error rate as a function of the number of hidden layers for FBANK features. Results are shown for the development and the core test sets with different layer sizes.

labels only for evaluation; that is, *after* decoding. This means that during the training of context-independent (CI) 3-state phone models we worked with $61 \cdot 3 = 183$ states. To be able to train context-dependent (CD) phone models as well, the decision tree-based state clustering tool of HTK was applied, resulting in 858 tied states. In both the CI and CD case, the training targets for the neural net (i.e., the state-level labeling of the frames) were obtained by training a conventional HMM (using HTK) and then a forced alignment was performed with it. During decoding no effort was made to fine-tune the language model weight and the phone insertion penalty parameters, they were just set to 1.0 and 0.0, respectively.

Two types of techniques were applied for preprocessing. First, we experimented with the standard MFCC coefficients, extracted from 25 ms frames at 10 ms frame skips. We used 13 MFCC coefficients (including the 0th one), along with the corresponding Δ and $\Delta\Delta$ values. Then we also tried to use the mel filter bank outputs directly. Mohamed et al. got better results with these features [4], and they also gave an analysis of the possible reasons behind this [15]. We had the opportunity to work with exactly the same features as they did in [4], as they kindly gave us the corresponding HTK config file. This preprocesing method extracted the output of 40 mel-scaled filters and the overall energy, along with their Δ and $\Delta\Delta$ values, altogether yielding 123 features per frame. In every case, the neural network was trained on 17 neighboring frames, so the number of inputs was 663 for the MFCC configuration, and 2091 for the FBANK configuration.

The weights of the neural net were initialized using the formula proposed by Glorot et al. [2]; that is, using uniformly distributed random numbers in the interval

$$\left[-c \cdot \frac{\sqrt{6}}{\sqrt{n_n + n_i}}, c \cdot \frac{\sqrt{6}}{\sqrt{n_n + n_i}}\right],$$

where n_n and n_i are the number of neurons and inputs to the given layer, respectively. The optimal value of c was found empirically, and was set to 0.4 in all experiments. The net was trained using semi-batch backpropagation, with the batch size being 100. The initial learn rate was set to 0.001 and held fixed while the error on the development set kept decreasing. Afterwards it was halved after each iteration, and the training was stopped when the improvement in the error was smaller than 0.1% in two subsequent iterations. This way the training took only 13-15 iterations on the average. As we mentioned earlier, apart from the softmax output layer, all neurons of the networks were rectifier neurons, and in the sparsifying experiments the sparsity penalty was applied to all of them.

4. RESULTS

The results for the MFCC feature set are shown in Fig.2, for the case of 2000 neurons per hidden layer. The plots show how the phone error rate improves due to the sparsity penalty and the introduction of more hidden layers. The scores are presented for both the core test set and the development set. As can be seen, the error drops quickly for up to 3 hidden layers, but then for 4 and 5 hidden layers there is little or no improvement. Also, adding the sparsity term to the cost function consistently improves the results by about 0.5% in absolute terms ($\approx$ 2.3% relative). The only exception is the case of 5 hidden layers on the core test set, where the improvement is insignificant. To quantify the sparsity, with the introduction of the penalty term it rises from 64-65% to 72%. Based on the scores on the development set, one should choose the sparse model with 5 hidden layers, which gives a 21.8% phone error rate on the core test set. In comparison, with a pretrained deep net a 22.3% error rate is reported with MFCCs using 6 hidden layers of 3072 units [4].

Net size	Dev.	Core test	Full test
4x2000 hidden units	18.0%	19.8%	19.4%
5x2000 hidden units	17.9%	20.3%	19.4%

Table 1. Phone error rates with CD phone models, using 4 and 5 hidden layers of 2000 units.

Fig.3 shows the results obtained for the FBANK feature set. As in the previous tests the sparsity penalty always helped, in this case the scores without the sparsity term are not shown at all. Instead, we use this figure to demonstrate the effect of changing the layer sizes. Recognition accuracies are shown for 1 to 5 hidden layers with 500, 1000 and 2000 neurons per layer, both for the development and the core test sets. As can be seen, for the development set the improvement is fully consistent with the increase of the number of neurons and layers, while there is a slight fluctuation for the core test set. The best result was got for 5 hidden layers with 2000 hidden neurons per layers, both for the development and the test sets. The phone recognition error rate in this case is 20.8% on the core test set. In comparison, in [4] 20.7% is reported for pretrained deep nets, using 8 layers of 2048 units per layer (though in [16] a slightly better result, 20.5% was obtained, there the input features were also more sophisticated). Based on this, we may regard rectifier neural nets as a competitive alternative to conventional deep nets.

Finally, Table 1 lists the phone error rates obtained using 858 triphone state targets. In this case the difference between the 4 and 5-layer nets is within 0.1% on the development set, while the scores show a large discrepancy for the core test set. As the results on core test set might be unreliable because of the very small size of this set, we decided to evaluate the models on the full test set as well. For this much larger test set the results were consistent with those obtained on the development set, and were well below 20%. Unfortunately, for these experiments with CD units we could not find a result in the literature that would provide a fair comparison, as most authors just work with CI models. Plahl et al. do use CD units and report a 19.1% error rate with deep nets, but they apply a discriminatively trained, boosted feature set [13]. Hence, by comparison, our result of 19.8% seems reasonable.

5. SUMMARY

In this paper we applied deep sparse rectifier neural nets to the TIMIT phone recognition task. To our knowledge, this type of neural net has not yet been attempted for speech recognition. From image recognition studies Glorot et al. concluded that rectifier networks can yield performances just as good as deep neural nets, but without the need for pre-training. Here we extended these results to phone recognition as well. Hence, rectifier neural nets appear to be a promising alternative to sigmoid-based deep neural networks. Of course, more experiments need to be performed on much larger data sets to confirm our findings. Another possible future task is to carry out experiments on the initialization scheme described earlier in the paper.

6. ACKNOWLEDGMENTS

The author would like to thank Xavier Glorot and Abdelrahman Mohamed for their helpful suggestions.

7. REFERENCES

[1] H. Bourlard and N. Morgan, *Connectionist Speech Recognition - A Hybrid Approach*, Kluwer, 1994.

[2] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, 2010, pp. 249–256.

[3] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[4] A. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. ASLP*, vol. 20, no. 1, pp. 14–22, 2012.

[5] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large vocabulary speech recognition," *IEEE Trans. ASLP*, vol. 20, no. 1, pp. 30–42, 2012.

[6] D. Yu, L. Deng, and G. E. Dahl, "Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[7] F. Seide, G. Li, L. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. ASRU*, 2011, pp. 24–29.

[8] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. AISTATS*, 2011.

[9] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. ICML*, 2010, pp. 807–814.

[10] O. Vinyals and L. Deng, "Are sparse representations rich enough for acoustic modeling?," in *Proc. INTERSPEECH*, 2012.

[11] D.L. Ly, V. Paprotski, and D. Yen, "Neural networks on GPUs: Restricted Boltzmann Machines," *Technical Report, Dept. of Electrical and Comp. Eng., University of Toronto*, 2009.

[12] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, 1995.

[13] C. Plahl, T. N. Sainath, B. Ramabhadran, and D. Nahamoo, "Improved pre-training of deep belief networks using sparse encoding symmetric machines," in *Proc. ICASSP*, 2012, pp. 4165–4168.

[14] G.S.V.S. Sivaram and H. Hermansky, "Sparse multilayer perceptron for phoneme recognition," *IEEE Trans. ASLP*, vol. 20, no. 1, pp. 23–29, 2012.

[15] A. Mohamed, G. E. Hinton, and G. Penn, "Understanding how Deep Belief Networks perform acoustic modelling," in *Proc. ICASSP*, 2012, pp. 4273–4276.

[16] G. E. Dahl, M. Ranzato, A. Mohamed, and G. Hinton, "Phone recognition with the mean-covariance Restricted Boltzmann Machine," in *NIPS*, 2010, pp. 469–477.