Experience in Training (Deep) Multi-Layer Perceptrons to Classify Digits

Jens Hocke, Thomas Martinetz

Institute for Neuro- and Bioinformatics, University of Lübeck

1 Introduction

Multi-Layer Perceptrons (MLPs) have been in use for decades. It seemed for a long time, that MLPs have reached their limits, but recent advances caught our attention. Ciresan et al. [1] show that by using a proper training set deep MLPs can outperform all other state of the art machine learning algorithms on the MNIST dataset for handwritten digits. However, a vast amount of training samples is needed, which has to be generated artificially with special transformations. The drawback is that appropriate transformations might be known for handwritten digits but not in general. Work by Hinton et al. and Bengio et al. [2, 3] suggest that unsupervised pre-training helps to find deep MLPs with better generalization from the training set, and thus avoiding to generate extra training data. We are interested in how the better generalization is archived and tested therefore alternative similar architectures. Here we present some observations we made in our first tests on the MNIST dataset. There is a lot of room for improvements to reach the error rates of Ciresan's approach.

2 Training of a Multi-Layer Perceptron

The standard approach to training a MLP is gradient descent on a cost function. To archive a faster convergence than with batch learning on a data set like MNIST with many repeating patterns, usually stochastic gradient descent is used. The most common cost functions are Mean Squared Error (MSE) and Cross-Entropy (CE). For the results shown below we used CE because it converged faster in our experiments. Applying it to a fully connected single layer network of 10 output units (one for each class) we get a test error of about 8.08 percent. By adding a hidden layer with 1000 fully connected units the error rate decreases to 1.72 percent.

Already the above network with 1000 hidden units has many more weights than training samples and reaches zero percent training error. It is an underdetermined system with many solutions having zero percent training error. By adding another layer, the classifier becomes even more powerful and the solution space for zero training error becomes even larger. There is no reason to expect a better generalization. Our experiments confirm this. A network with two hidden layers of 1000 and 500 units performs even worse on the test set (1.82 %). Ciresan et al. [1] benefited from an increased number of layers, but they circumvented

the underdeterminacy by artificially creating a basically infinitely large training set by applying elastic deformations to the original data. But to generate this data it must be known which transformations are appropriate for the dataset.

Is it possible to make use of the additional power of the classifier without generating extra data? Hinton and Bengio [2, 3] suggest the use of autoencoders. These ensure that the information loss in every layer is minimal. After this kind of unsupervised training the entire network is retrained in the usual way using back-propagation. Interestingly, this yields better generalization than plain back-propagation without pretraining. It is not clear what the autoencoder does. If the hidden layer is smaller than the previous layer, an encoding similar to PCA is found. But for a larger hidden layer, the problem is again underdetermined. The simplest solution would be the identity (Direct connection of input and output). However, in practice this is not the solution found. Bengio et al. hypothesize this may be caused by a weight-decay they used preventing large weights, or stochastic gradient descent finding an arbitrary solution. The weights found by the autoencoder are usually only used as initialization, as the starting point for the back propagation learning of large (deep) underdetermined networks. It seems, that this choice of the starting point lets the network converge to a good solution in the solution space. Erhan et al. [4] hypothesize that pretraining is a regularizer with an infinite penalty on certain regions of the parameter space. Would it not be better to use autoencoders explicitly as a regularization for the network? This strategy can be motivated by the fact that the brain is not only performing one specific classification task with its visual input, but many different ones. Then for every task different features are used, thus almost all features need to be encoded in the hidden layer, which is enforced by the autoencoder. We have tested this approach, but did not archive a good generalization (1.82 %).

Bengio et al. use a layer-wise training for the autoencoder. Would it also work to add iteratively one hidden layer and train only the newly added layer and the output neurons to classify correctly? This approach should ensure that all information needed for the classification is passed from the lower layers to the output layer. However, our experiments show that this scenario does not lead to better results. On the test set the performance is just as good as or even worse than using only one hidden layer (1.73 %). By retraining the entire network using the previously found weights as initialization leads to a slight improvement (1.70 %), but still worse than the autoencoder result. It is interesting to note that in all cases the features of the first hidden layer resemble parts of digits, if the networks are trained with noisy samples.

To summarize, so far the effect of pretraining deep MLPs with autoencoders is not yet really understood, but also does not really lead to competitive results on the MNIST dataset.

References

1. Ciresan, D., Meier, U., Gambardella, L., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. Neural computation **22**(12) (2010) 3207–3220

2. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–507
3. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. Advances in neural information processing systems **19** (2007) 153
4. Erhan, D., Manzagol, P., Bengio, Y., Bengio, S., Vincent, P.: The difficulty of training deep architectures and the effect of unsupervised pre-training. Artificial Intelligence **5** (2009) 153–160