ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

Comparison of Four- and Six-Layered Configurations for Deep Network Pretraining

Jan Hänninen and Tommi Kärkkäinen

Department of Mathematical Information Technology
P.O. Box 35, 40014 University of Jyväskylä - Finland

Abstract. Using simpler building blocks to initially construct a deep network, with their finetuning for the full architecture, is known to improve the deep learning process. However, in many cases the pretrained networks are obtained using different training algorithms than used in their final combination. Here we introduce and compare four possible architectures to pretrain a deep, feedforward network architecture, using exactly the same formulation throughout. Based on the analytical formulations and experimental results, one of the tested configurations is concluded as the recommended approach for the initial phase of deep learning.

1 Introduction

The feedforward neural network (aka MultiLayered Perceptron, MLP) provides a natural transformation architecture for nonlinear dimension reduction referred as *autoencoding*. The general approach for a given data $\{\mathbf{x}_i\}_{i=1}^N$ is conceptually very simple: use $\mathbf{x}_i$'s both as input and as the desired output in learning and squeeze the vectors in the layered transformation using a small hidden, central layer. Such an approach has, though, a known challenge related to local search/optimization: are we finding "good" solutions [1].

A lot of deep network work focuses on probabilistic networks like Deep Belief Networks or Reduced/Deep Bolzmann Machines, with binary representation. Especially the archetypical approach in [1] uses Restricted Boltzmann Machine (RBM) to determine a set of pretrained two-layered networks, which are then fine-tuned by optimizing all the weights of the cascadic structure. As emphasized in [2], such an approach can be referred as the breakthrough to effective training strategies for deep architectures. Based on a comparison of different blockwise construction techniques for deep networks in [3], it is concluded (p. 7): "greedy layer-wise procedure essentially helps to better optimize the deep networks, probably by initializing the hidden layers so that they represent more meaningful representations of the input, which also yields to better generalization." Thorough review on deep learning, along with the known difficulties, is also provided in [4]. More recently, another review with a proper attempt to track the historical development behind the various techniques involved, is provided in [5].

The main reason why a deep network might be difficult to train as a whole, especially when starting from a randomly initialized weights, is known as *the vanishing gradient problem* (see [5], Section 5.9 and references therein). Actually one can see from the explicit, analytic formulae (see Section 2) that when

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

any derivative of a hidden activation function (esp. on the saturation regions on the tails) becomes small, then all the weight-derivatives of such component's consequents in backpropagation also vanish. Also vanishing gradients can be attempted to be circumvented by first constructing more shallow, partial transformations which are fully optimized afterwards. From the optimization point of view, pretraining finds a better starting point in the whole search space for the actual determination of all the weights with a local optimization approach for finetuning.

Feedforward network with one hidden layer is structurally very similar to the Reduced Bolzmann Machine and the LSE gradient is said to approximate the RBM's probabilistic log-likelihood gradient [6]. Especially both approaches can be used for the pretraining of a deep network. Again along the lines of [1], such one-hidden-layer networks are typically used to both pretrain and to determine the sizes of the partial configurations which are cascaded as the hidden layers in the actual deep architecture. Our purpose here is to derive and test slightly deeper structures than one-hidden-layer that could be suitable for pretraining and which would precisely coincide with the corresponding layers in the deep network. Furthermore, according to the results in [7], training a deep neural network using better optimization solvers compared to the steepest descent based BP (or stochastic gradient descent, i.e., on-line BP) improves the obtained performance. Therefore, we also apply here available advanced nonlinear optimizer to determine the weights instead of any BP variant with hand-tuned metaparameters (e.g., learning rate). The downside without an application of parallel or distributed computing model is restriction to small data sets. However, the characteristics of the compared configurations can still be revealed.

The contents of the article are as follows: after this introduction, we provide the basic formulae for the kind of networks in Section 2. Then, in Section 3, the comparison framework and results of the comparison are given. Finally, the overall conclusions are drawn in Section 4.

2 Derivation of autoencoders

Next we briefly derive and describe the basic forms of deep networks to be compared below, using the formalism introduced in [8]. Hence, let $f(\cdot)$ denote the activation function which, for a layer of size m, compose the so-called *diagonal function-matrix* $\mathcal{F} = \mathcal{F}(\cdot) = \mathrm{Diag}\{f_i(\cdot)\}_{i=1}^{m}$ where $f_i \equiv f$. Then the output of a feedforward network with L layers and linear activation on the final layer, for an input vector $\mathbf{x}$, reads as

$$\mathbf{o} = \mathbf{o}^L = \mathcal{N}(\mathbf{x}) = \mathbf{W}^L \mathbf{o}^{(L-1)}, \tag{1}$$

where $\mathbf{o}^0 = \mathbf{x}$ and $\mathbf{o}^l = \mathcal{F}(\mathbf{W}^l \mathbf{o}^{(l-1)})$ for $l = 1, \ldots, L-1$. We assume in what follows that L is even. When none of the layers contain the so-called bias nodes, then the dimensions of the weight-matrices are given by $\dim(\mathbf{W}^l) = n_l \times n_{l-1},\ l = 1, \ldots, L$, where n_0 is the length of the learning data vectors $\{\mathbf{x}_i\}$,

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

$n_L = n_0$ in the autoencoding context, and $n_l, 0 < l < L$, determine the sizes (number of neurons) of the hidden layers with $n_{L/2} < n_0$.

In order to determine the weights of the autoencoder, we minimize the least-squares error cost function

$$\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L) = \frac{1}{2N}\sum_{i=1}^{N} \|\mathbf{W}^L \mathbf{o}_i^{(L-1)} - \mathbf{x}_i\|^2. \tag{2}$$

Then (see [8]) the gradient-matrices $\nabla_{\mathbf{W}^l}\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L),\ l = L, \ldots, 1$, for (2) are of the form

$$\nabla_{\mathbf{W}^l}\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{d}_i^l\, [\mathbf{o}_i^{(l-1)}]^T,$$

where

$$\mathbf{d}_i^L = \mathbf{e}_i = \mathbf{W}^L \mathbf{o}_i^{(L-1)} - \mathbf{y}_i, \tag{3}$$

$$\mathbf{d}_i^l = \mathrm{Diag}\{(\mathcal{F})'(\mathbf{W}^l\, \mathbf{o}_i^{(l-1)})\}\, (\mathbf{W}^{(l+1)})^T\, \mathbf{d}_i^{(l+1)}. \tag{4}$$

As proposed in the introduction, our purpose here is to restrict ourselves to four- and six-layered networks satisfying (2)–(4). In addition, once more following [1], we also consider the corresponding *self-adjoint autoencoders* where we formally restrict into $\mathbf{W}^{L-k} = (\mathbf{W}^{(k+1)})^T$ for $k = 0, \ldots L/2 - 1$. Hence, this refers to an autoencoder where the decoder is an algebraich adjoint (kind of transpose because of the diagonal function matrix convention) of the encoder. Using similar derivation as in [8] it follows that, for the four-layered autoencoder $(\mathbf{W}^1)^T\mathcal{F}((\mathbf{W}^2)^T\mathcal{F}(\mathbf{W}^2\mathcal{F}(\mathbf{W}^1\mathbf{x})))$, the gradient matrices $\nabla_{\mathbf{W}^l}\mathcal{J}, l = 1, 2$, for (2) are of the form

$$\nabla_{\mathbf{W}^2}\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N}\left[\mathbf{d}_i^2(\mathbf{o}_i^1)^T + \mathbf{o}_i^2(\mathbf{d}_i^3)^T\right], \quad \nabla_{\mathbf{W}^1}\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N}\left[\mathbf{d}_i^1\mathbf{x}_i^T + \mathbf{o}_i^3\mathbf{e}_i^T\right].$$

Hence, the derivative matrix $\nabla_{\mathbf{W}^l}\mathcal{J}$ for the self-adjoint autoencoder simply reads as $\nabla_{\mathbf{W}^l}\mathcal{J} + [\nabla_{\mathbf{W}^{L-(l-1)}}\mathcal{J}]^T$, which is used in the symmetric, six-layered network. To this end, we observe from $\nabla_{\mathbf{W}^1}\mathcal{J}$ and $\nabla_{\mathbf{W}^2}\mathcal{J}$ that the self-adjoint structure incorporate more information inside the derivative matrix making its zero-condition at a local optimum more involved. This indicates that the self-adjoint autoencoders might be more tolerant against the vanishing gradient problem compared to the fully decoupled structure.

3 Experimental results

In the experiments, we compare the full and symmetric, four- and six-layered autoencoders. The comparison is based on a set of small datasets, obtained from the UCI repository, as described in Table 1. All variables are preprocessed using the min-max scaling into $[-1, 1]$ with tanh as the activation function for the feedforward network (see [8]). Moderate sizes of N are due to restricting the

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

computational burden, resulting from using the full-fledged nonlinear optimizer *fminunc* from *Octave* to minimize (2). For exploration, local search with random generation of the initial weights from $\mathcal{U}([-1,1])$ is repeated five times with the smallest error selected as the actual result. We also restrict ourselves to small n_0, because the tested configurations are further restricted in such a way that the squeezing always provides two-dimensional encoding, i.e., $n_2 = 2$ for 4-layered (4L) and $n_3 = 2$ for 6-layered (6L) autoencoder. In this way, we can compare the obtained accuracy to the corresponding result of the linear (least-squares) autoencoder provided by the classical PCA with the two principal components.

The comparison is focused on the autoencoding accuracy assuming the same number of unknown weights in the four configurations tested. As the error measure we apply (see [9])

$$e_{\mathrm{MR}} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\sum_{j=1}^{n_0}(\mathcal{N}(\mathbf{x}_i)-\mathbf{x}_i)_j^2} \quad \text{(Mean-Root-Squared-Error)}.$$

Furthermore, we restrict $n_3 = n_1$ in 4L and $n_5 = n_1$ & $n_4 = n_2$ in 6L. Then the number of weights in 4L or 6L autoencoders is fully determined by the size of the hidden layer(s) between input/output and the central layers, which is then varied.

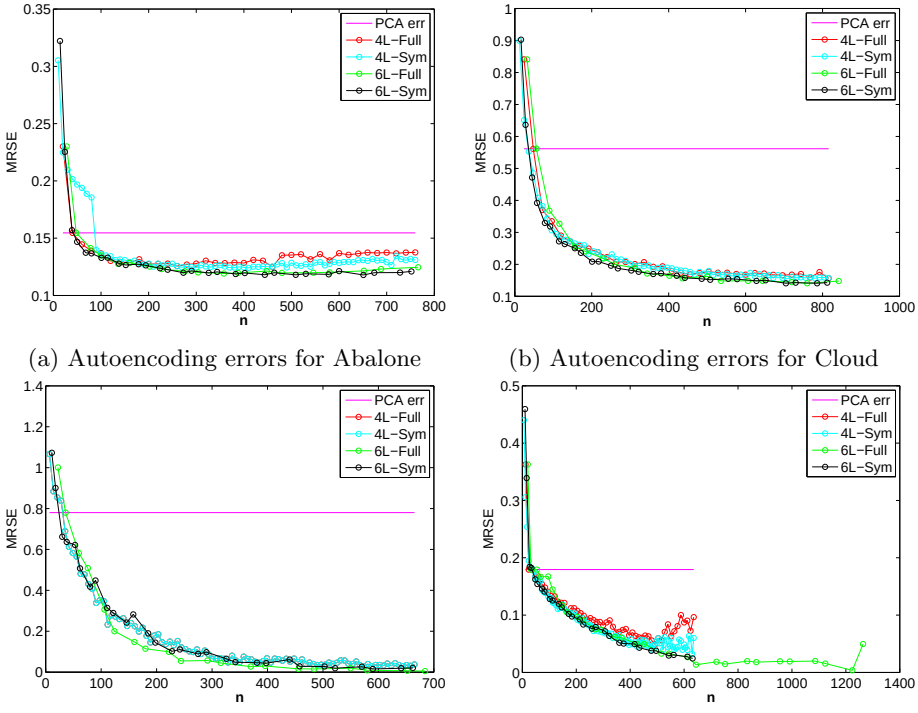
Results of the comparison are given in Table 1. For each dataset, we first make a pairwise comparison of two different methods as follows: for coarser grid of the number of unknowns n, for the e_{MR}'s which are under the level of the PCA error, the number of better results (smaller error) are counted. Then, the method which has larger number of smaller errors is ranked better than the method compared with. In this way, we rank all methods for all datasets. This ranked-based evaluation is finally summarized by simply counting together all the individual rankings to create the final preferences (see [10] for a similar evaluation protocol).

Visualization of errors for the four configurations with four datasets are given

Dataset	N	n_0: Type of variables	4-Full	4-Sym	6-Full	6-Sym
Abalone	4178	8: 1 nominal, 7 real	4	3	2	1
Cloud	1024	10: real	4	3	2	1
Glass	214	9: real	3	4	1	2
Haberman	306	3: int	4	3	2	1
Iris	150	4: real	4	2	3	1
Seeds	210	4: real	3	4	1	2
Teaching Ass. Eval.	151	5: 2 bin, 2 categ, int	2	4	1	3
All			24 (4.)	23 (3.)	12 (2.)	11 (1.)

Table 1: Datasets and method rankings in the experiments.

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.



(a) Autoencoding errors for Abalone

(b) Autoencoding errors for Cloud

(c) Autoencoding errors for TAE

(d) Autoencoding errors for Iris

Fig. 1: Autoencoding errors graphs

in Figure 1. For 'Iris' (bottom right) the largest structure, full 6L, is prolonged wrt ns to illustrate the increase of the error for larger number of unknowns.

4 Conclusions

The essence of the comparison here was to consider the quality of different feed-forward networks for the same number of unknowns to be optimized when solving (2). Based on the computational experiments, cf. Table 1 and Figure 1, we obtained clear separation between the different types of architectures. The deeper models with six layers were clearly better than the four-layered ones. This supports the Betti number based analysis of the representation capability as proposed in [11], where it was shown that deep networks are able to realize functions with higher complexity compared to the shallow ones. Actually, as can be seen from the placement points in the pairwise rankings, the two groups of layers, 4L and 6L, were very close to each other in the overall assessment. One tightening element, a slight outlier in the results, was the ranking with "Teacher Assistant Evaluation, TAE" dataset, where the full networks showed better performance compared to the algebraich adjoint ones. But, for this particular dataset, the

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

variables were the most discrete ones in the comparison. To this end, even if very close to each other, both the rank summary and the better avoidance of the vanishing gradient problem, i.e. the inherent difficulty of optimizing the deep weights, with noting the special character of TAE, allow us to conclude that the six-layered symmetric autoencoder is the recommended approach for pretraining a deep, feedforward network. This recommendation is supported by noting the increased error trends in Figures 1a and 1d for all the other techniques. In our future work, we try to improve the efficiency of the training algorithms to tackle larger datasets, because the solution of the nonlinear optimization problem with sufficient accuracy is computationally challenging even for the size of problems as considered here.

References

[1] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[2] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, 2010.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160. MIT Press, 2007.

[4] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, 2009.

[5] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[6] Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, 2009.

[7] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, 2011. 8 pages.

[8] T. Kärkkäinen. MLP-network in a layer-wise form with applications to weight decay. *Neural Computation*, 14(6):1451–1480, 2002.

[9] T. Kärkkäinen. On cross-validation for MLP model evaluation. In *Structural, Syntactic, and Statistical Pattern Recognition*, Lecture Notes in Computer Science (8621), pages 291–300. Springer-Verlag, 2014.

[10] M. Saarela and T. Kärkkäinen. Analyzing student performance using sparse data of core bachelor courses. *Journal of Educational Data Mining*, 7(1):3–32, 2015.

[11] M. Bianchini and F. Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565, 2014.