ITM Web of Conferences **6**, 02007 (2016)
DOI: 10.1051/itmconf/20160602007
© Owned by the authors, published by EDP Sciences, 2016

Hybrid pre training algorithm of Deep Neural Networks

I. S. Drokin[1, a]

[1]*St. Petersburg State University, Faculty of applied mathematics and control processes, 199034 St. Petersburg, Russia*

Abstract. This paper proposes a hybrid algorithm of pre training deep networks, using both marked and unmarked data. The algorithm combines and extends the ideas of Self-Taught learning and pre training of neural networks approaches on the one hand, as well as supervised learning and transfer learning on the other. Thus, the algorithm tries to integrate in itself the advantages of each approach. The article gives some examples of applying of the algorithm, as well as its comparison with the classical approach to pre training of neural networks. These examples show the effectiveness of the proposed algorithm.

1 Introduction

An application of the pre-training mechanisms for a multilayer perceptron training results in greatly improved quality and speed of the deep networks training. In this paper, we propose a way of the weights initialization using principles of supervised learning and self-taught learning approach [1]. Currently used methods for the weights initialization of a multilayer perceptron are based on restricted Boltzmann machine [2] and auto-encoders (stacked auto-encoders, SAE) [3]. There are generalizations of the auto-encoders approach for convolutional neural networks described in [4], as well as the feature extracting algorithm based on the data representations, obtained during training of stacked auto-encoders [5]. Also there is an algorithm of initialization of the network weights based on principal component analysis [6]. The article proposes an iterative algorithm of weight initialization that is based on hidden layers of neural networks weights refinement through solving the original or similar classification or regression.

2 Algorithm description

Consider the set of the pairs $\{x_i, y_i\}, i = 1 \dots K$, where $\{x_i \in R^n, y_i \in R^m\}$, x_i – is an input sample and y_i - is a target response. It is necessary to construct such a function $f(x)$, so that $f(x_i) = y_i, i = 1 \dots K$. A multilayer perceptron N is considered for the proposed algorithm description. Let $L = \{L_1 \dots L_k\}$ be its hidden layers set, L_0 -is the input layer, $\{x_{i,j}\}_{i=1}^{K}$ – are the outputs of the j-th layer. Let $u(l)$ equals to numbers of the neurons in layer l, $W(L_j) = W_j$ – are neuron's weights of the j-th layer, $N(j)$ – is the j-th hidden layer of the network N. The pre-training neural network algorithm based on auto encoders can be summarized as it is showing in Algorithm 1.

For $j \leftarrow 1, k$ **Repeate**

1. Initialization of the network $\tilde{N}^j$ with one hidden layer size of $u(L_j)$, an input layer size of $u(L_{j-1})$ and an output layer size of $u(L_{j-1})$. Let $\tilde{W}$ were the weights of the hidden layer.
2. Training of the network $\tilde{N}^j$ proceed on sample set $\{x_{i,j-1}, x_{i,j-1}\}, i = 1 \dots K$
3. $\hat{W}_j = \tilde{W}$

Result $\{\hat{W}_j\}_{j=1}^{k}$

Algorithm 1. Network pre-training with SAE application.

Applying of this algorithm allows use of the sample set $\{\hat{W}_j\}_{j=1}^{k}$ for the fairy well initialization of the hidden layer's weights of the network for the entire network training with Backpropagation method. This approach also allows to apply input data the desired response of which is not known.

This approach is proposed to be complemented as follows. Weights of the j-th hidden layer after pre-training with auto-encoder can be refined by training network with one hidden layer on the original or a similar problem, using transfer learning concept [7–9]. That means, we have to use the weights $\tilde{W}$, received on the second paragraph of the j-th step of the algorithm for initialization of the network with one hidden layer, an

[a] Corresponding author : ivan.s.drokin@bk.ru

This is an Open Access article distributed under the terms of the Creative Commons Attribution License 4.0, which permits unrestricted use distribution, and reproduction in any medium, provided the original work is properly cited.

input layer size of $u(L_{j-1})$ and an output layer size of m, and train the network on $\{x_{i,j-1}, y_i\}$, $i = 1 \dots K$ sample set.

Also, on the j-th iteration of the algorithm, the weight of the hidden layers from the first to j-th can be specified by training of the network with j hidden layers from the original or similar network. Weight obtained on the previous steps $\{\hat{W}_t\}_{t=1}^{j}$ are used to initialize the network with j hidden layers, an input layer size of n and an output layer size of m. The network is trained on a sample set of $\{x_i, y_i\}, i = 1 \dots K$. This setting of parameters makes sense for all hidden layers, except of the first and the last, since this setting for the first layer is exactly the same as it was described in the above procedure from the previous paragraph, and for the last layer it is the task of the entire network training.

Thus, the original algorithm is added by two additional learning processes. The general algorithm can be described as it is showing in Algorithm 2.

For $j \leftarrow 1, k$ **Repeate**

1. Initialization of the network $\tilde{N}^j$ with one hidden layer size of $u(L_j)$, an input layer size of $u(L_{j-1})$ and an output layer size of $u(L_{j-1})$. Let $\tilde{W}$ be the weights of the hidden layer.
2. Training of the network $\tilde{N}^j$ is proceed on $\{x_{i,j-1}, x_{i,j-1}\}, i = 1 \dots K$ sample set.
3. Initialization of the network $`N^j$ with one hidden layer size of $u(L_j)$, an input layer size of $u(L_{j-1})$ and an output layer size of m, the weights of the hidden layer are initialised as $\tilde{W}$.
4. Training of the network $`N^j$ is proceed on $\{x_{i,j-1}, y_i\}, i = 1 \dots K$ sample set.
5. Initialization of the network $\hat{N}^j$ with j hidden layers size of $u(L_t), t = \overline{1, j}$, an input layer size of n and an output layer size of m, the weights of the hidden layers are initialised as $\{\hat{W}_t\}_{t=1}^{j-1}$, $\hat{W}_j = W(`N(1))$.
6. Training of the network $\hat{N}$ is proceed on $\{x_i, y_i\}, i = 1 \dots K$ sample set. $t = 1 \rightarrow j$ $\hat{W}_t = W(\hat{N}^j(t))$

Result $\{\hat{W}_j\}_{j=1}^{k}$

Algorithm 2. Consecutive network pre-training.

As it is evident from the description of the algorithm, the weights for the network N initialization with initial configuration will be obtained as a result of the network pre-training. The weights for $k-1$ network with fewer number of layers will be obtained on the pre-training stage just as a small additional bonus. This network can be used for further training and integration of the constructed network to a committee. Also the pre-training scheme allows to use not initial data set, but some other set which is close to the initial one. For example, if the original task is training on the CIFAR-100 data set, the set CIFAR-10 [10] for the pre-training stage can be used.

3 Experiment description

For the experiment, CIFAR-10, CIFAR-100 datasets were used. The dataset CIFAR-10 is a set of 50,000 training samples and 10,000 test samples of color images sized 32 by 32 pixels, divided into ten classes. CIFAR-100 is a set of 50,000 training samples and 10,000 test samples of color images of size 32 by 32 pixels, divided into one hundred classes. Thus, the size of the input layer of the network is 3072, the size of the output layer is 10 for CIFAR-10 and 100 for CIFAR-100 respectively. For further discussion, the network configuration will be briefly written as $n - n_1 - n_2 - \dots - n_k - m$, where n – is the size of the input layer, $n_1, n_2 \dots n_k$ - are the numbers of neurons in the hidden layers, m - is the size of the output layer.

A softplus activation was used for all of the hidden layers, while a softmax was used for the output layer. A categorical crossentropy was applied as an error function. A dropout with a coefficient of 0.3 was used for each hidden layer, the size of a single batch was equal to 16, a stochastic gradient descent with Nesterovsky momentum was used as an optimizer, a learning rate was 0.001, a moment was equal to 0.9. No data volume increasing techniques had been used in the experiment.

As observed parameters we have considered errors of the training process for training or testing sample depending on the epoch of learning. The proposed algorithm is suggested to compare with a classical algorithm of neural network training with pre-training through stacked autoencoders. All the parameters of multilayer perceptrons training and autoencoders in both pre-training algorithms are the same.

For the experiment with using the CIFAR-10 dataset the network configuration of $3072-2560-2048-1536-1025-512-256-10$ was considered, and for the experiment with the CIFAR-100 dataset the configuration of $3072-2560-2048-1536-1025-512-256-100$ was used. Similar model configuration are proposed for the experiment with pre-training on CIFAR-10 and the final training was done using CIFAR-100. The behavior of the errors during the first training epochs were investigated. The tests results for the networks are shown in Fig. 1, Fig. 2, Fig. 3.

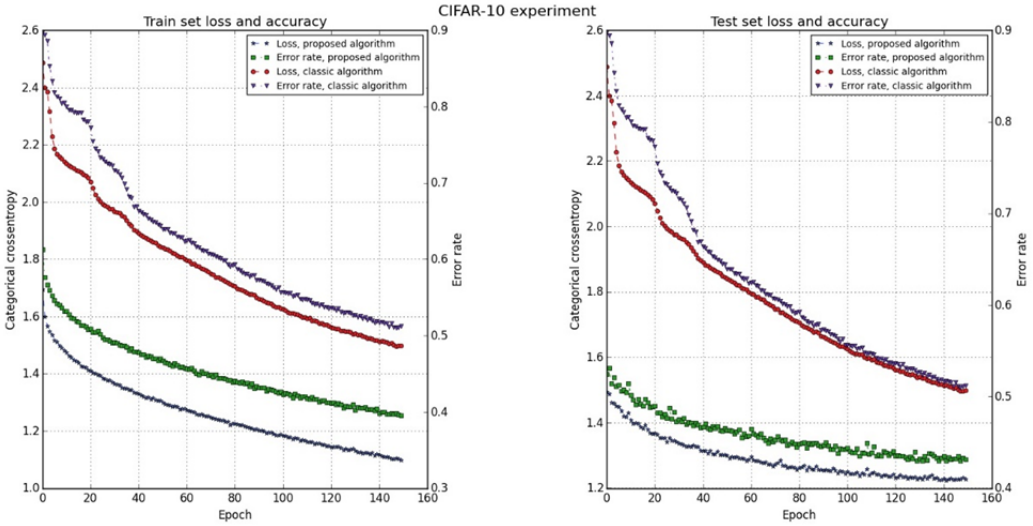

Figure 1. Errors in training and testing samples for CIFAR-10 network

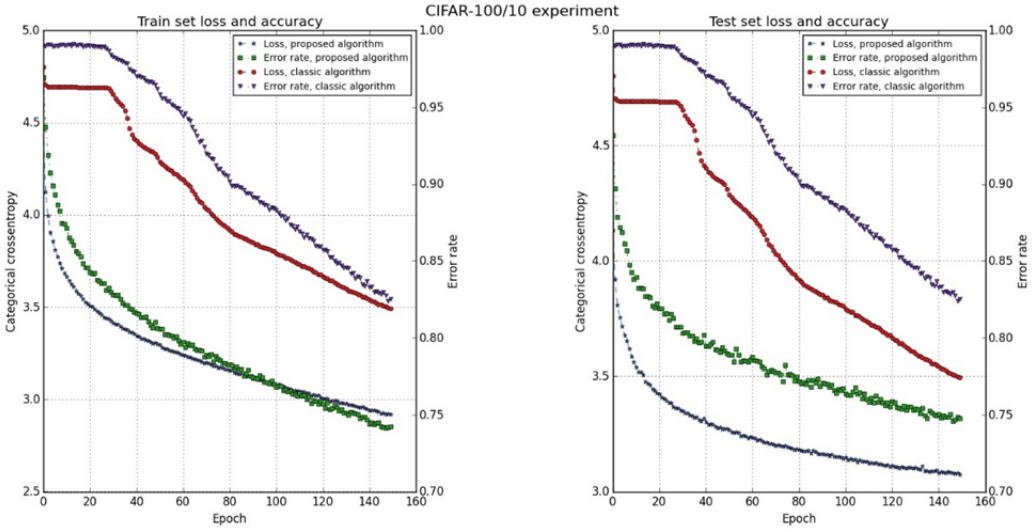

Figure 2. Errors in training and testing samples for CIFAR-100 network

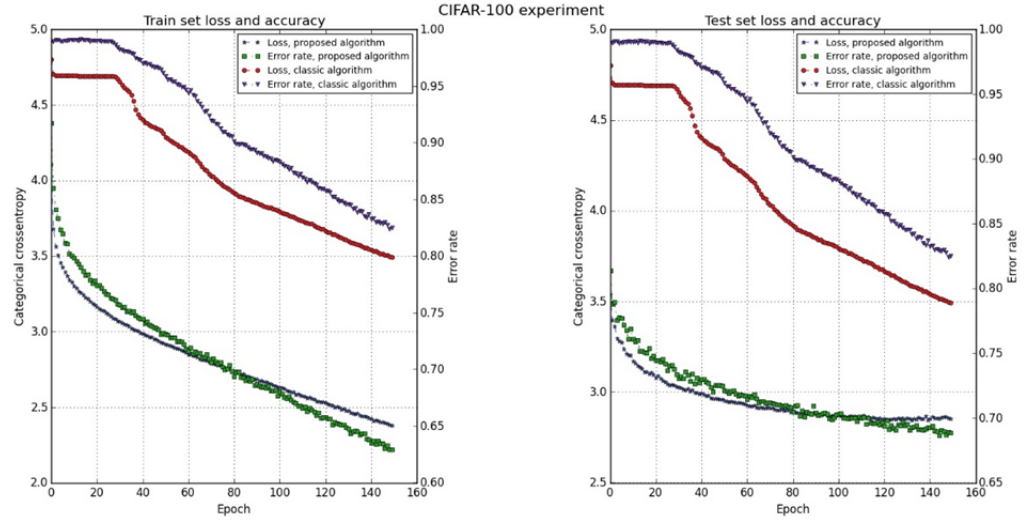

Figure 3. Errors in training and testing samples for CIFAR-100/ CIFAR-10 network

4 Results and findings

As it could be seen from the graphs, the presented method is an effective version of the multilayer perceptron weights initialization. Furthermore, efficiency of the approach is increasing with number of the hidden layers increment. However, the proposed method has a number of disadvantages. In particular, this method requires more computational resources. However, the proposed algorithm also has a number of advantages.

First of all, the algorithm allows to apply self-taught learning and transfer learning approaches. Also using the algorithm permits networks with fewer numbers of layers to be trained (or pre-trained if transfer learning is used). These networks can be used for merge in the committees using, for example, boosting algorithm. Thus, intermediate calculations would be used in the resulting classifier.

The directions for further research could include the following points:

1. In steps 3-6, one can supplement additional hidden layers
2. Steps 3-6 could be used for the first few layers only
3. Combination of the networks into general classifier obtained in steps 4 and 6

The proposal to use steps 3-6 only for the first few layers is based on the observation known as attenuation gradient [13, 14]. According to this observation, first layers of a multilayer perceptron are trained worse than others, and using stages 3-6 only for the first few layers, allows, on the one hand, to obtain the sufficient quality of the weights approximation, but, on the other hand, to save computational resources.

The proposed approach can also be used with Restricted Boltzmann machines for the multilayer perceptrons pre-training as well as with convolutional auto encoders and convolutional neural networks.

5 Conclusion

An hierarchical algorithm of a multilayer perceptron pre-training based on self-taught learning and transfer learning approaches was proposed. There were conducted some tests showing the performance of the approach in this paper. As well as the next steps and directions for further research of the algorithm and its generalizations and improvements were presented. In particular, it is proposed to conduct an analisys of the application of algorithm for convolutional neural networks, as well as for Restricted Boltzmann machines and Deep Belief Networks. The algorithm can use both marked and unmarked data for initialization and training of the neural network. Also an iterative learning process allows to apply an problem close to the original to initialize the hidden layers of the network.

References

1. UFLDL Tutorial. Available http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
2. G.E. Hinton and R.R. Salakhutdinov, Science, **313**, 504 (2006)
3. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio and P.-A. Manzagol, The Journal of Machine Learning Research archive, **11**, 3371 (2010).
4. J. Masci, U. Meier, D. Ciresan and J. Schmidhuber. *21st International Conference on Artificial Neural Networks*, Part I, pp. 52-59 (2011)
5. J. Gehring, Y. Miao, F. Metze and A. Waibel, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 3377-3381 (2013)
6. P. Baldi and K. Hornik, Neural Networks, **2**, 53 (1989)
7. R. Caruana, Machine Learning, **28**, 41 (1997)
8. R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, *Proceedings of the 24th international conference on Machine learning. ACM*, pp. 759-766 (2007)
9. D.C. Ciresan, U. Meier and J. Schmidhuber, *2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6 (2012)
10. The CIFAR dataset. Available http://www.cs.toronto.edu/kriz/cifar.html
11. The MNIST database of handwritten digits. Available http://yann.lecun.com/exdb/mnist/
12. P. Simard, *Proc. IC-DAR,* pp. 958-962 (2003).
13. X. Glorot and Y. Bengio, *International conference on artificial intelligence and statistics*, pp. 249-256 (2010)
14. R. Pascanu, T. Mikolov and Y. Bengio, *Understanding the exploding gradient problem.* Tech. Rep. (Universite De Montreal, 2012). arXiv:arXiv:1211.5063