Training Neural Networks by Optimizing Random Subspaces of the Weight Space

Ewa Skubalska-Rafajłowicz(✉)

Department of Computer Engineering, Faculty of Electronics,
Wrocław University of Science and Technology, Wrocław, Poland
ewa.rafajlowicz@pwr.wroc.pl

Abstract. This paper describes a new approach to feed-forward neural networks learning based on a random choice of a set of neurons which are temporally active in the process of neural network weight adaptation. The rest of the network weights is locked out (frozen). In contrast to the "dropout" method introduced by Hinton et al. [15], the neurons (along with their connections) are not removed from the neural network during training, only their weights are not modified, i.e. stay constant. This means that in every epoch of training only the random part of the neural networks (a chosen set of neurons and its connections) adapts. Freezing of neurons suppresses overfitting and prevents drastic increment of weights during the learning process, since the overall structure of the neural networks does not change. In many cases the approach based on training only some parts of the neural network (subspaces of the weight space) shortens the time of training. Experimental results for medium size neural networks used for modeling regression are also provided.

Keywords: Neural network training · Stochastic gradient decent · Random subspace optimization · Over-fitting · Deep learning

1 Introduction

In this paper we propose a new strategy of optimization based methods of neural networks learning. In this strategy only a randomly chosen part of neurons along with their connections, i.e., in-going and out-going weights is adjusted in every epoch. The rest of the network remains temporarily unchanged. We will say that the neurons not chosen for training are temporarily frozen. Changing at once only a part of the decision variables is applied in many optimization (and other numerical) methods starting from the coordinate descent (Gauss-Seidel) strategy or alternating direction method of multipliers [6,19]. However, these methods are based on fixed partition of optimized variables. On the other hand, random selection of neurons, when only one unit is updated at a time is performed in asynchronous learning of the Hopfield network [13]. The approach proposed here introduces an additional level of variability to the process of neural network learning similarly as stochastic gradient does, enlarging exploration capability of

© Springer International Publishing Switzerland 2016
L. Rutkowski et al. (Eds.): ICAISC 2016, Part I, LNAI 9692, pp. 148–157, 2016.
DOI: 10.1007/978-3-319-39378-0_14

the neuron weight space. Furthermore, this strategy allows neurons to be more evenly exploited, since only some neurons, i.e., a randomly selected subspace of weights will be updated in the actual epoch of training. As a consequence, it will result in more evenly distributed network weights and better neural network generalization possibility [2].

From this point of view, one can see some similarities between freezing randomly chosen parts of the learned neural network that is proposed here and the Dropout method [15]. Dropout is a random procedure in which only randomly chosen neural network neurons (hidden units) are trained, whereas the rest of the neural net neurons and their incoming and outcoming connections are temporarily removed from the neural network during a small period time of training. Stochastic gradient decent (SGD) training is lately often combined with the dropout method [1,15,21]. It is known that dropout gives visible improvements on many benchmark data sets for speech and image classification [1,8,21,24]. Dropout is used for preventing a trained neural network from overfitting and it usually improves the performance of SGD training of deep neural nets [5,8,14,20,23].

Although the freezing strategy can be applied not only to learn feed-forward neural nets (FFNN), in this paper we restrict ourselves to FFNN architecture with one hidden layer of sigmoid neurons and a linear output neuron, i.e., the simplest structure which is known to be the universal approximator [7,16] and SGD learning algorithm. Nevertheless, it should be noted, that the freezing strategy can be used also with many other popular optimization based FFNN training methods, such as Levenberg-Marquardt, conjugate gradient methods, BFSG [10,12], among many others.

This paper is structured as follows. The next section describes the structure of our FFNN and the problems of training it in the context of overfitting in nonlinear regression neural network models training. Section 3 presents details of the learning strategy proposed here with randomly frozen neurons. Section 4 shows some simulation results which explain and confirm proposals and conclusions developed in the previous sections. In Sect. 5 we summarize the provided simulation results and indicate possible further developments.

2 Network Structure, Learning Optimal Weights and the Problem of Overfitting

We assume that the neural network is a usually used FFNN with one or more hidden layers. The output of the network is given by:

$$y_{net}(X) = \sum_{j=1}^{M} w_j^{(0)} \phi_j^{(0)}(X), \tag{1}$$

where $X \in R^n$ $(X = (x_1, \ldots, x_n)^T)$ is an input vector, M is the number of neurons in the hidden layer (the last hidden layer), and $w_j^{(0)}$ is a weight joining j-th neuron to the output of the network. For the network with exactly one

hidden layer we have

$$\phi_j^{(0)}(X) = f_{act}\left(\sum_{i=1}^{n} w_{ij}^{(-1)} x_i + b_j\right),\ j = 1, \ldots, M, \tag{2}$$

where f_{act} is an activation function. In general, for the network with $K > 1$ hidden layers,

$$\phi_j^{(0)}(X) = f_{act}\left(\sum_{i=1}^{M^{(-1)}} w_{ij}^{(-1)} \phi_i^{(-1)}(X) + b_j^{(-1)}\right),\ j = 1, \ldots, M, \tag{3}$$

where $\phi_j^{(0)}(X)$ is an ouput of j-th neuron in the last hidden layer and consecutively

$$\phi_j^{(-k)}(X) = f_{act}\left(\sum_{i=1}^{M^{(-k-1)}} w_{ij}^{(-k-1)} \phi_i^{(-k-1)}(X) + b_j^{(-k-1)}\right), \tag{4}$$

where $j = 1, \ldots, M^{(-k)},\ k = 1, \ldots, K-1,\ M^{(-K)} = n$ and $\phi_i^{(-K)}(X) = x_i,\ i = 1, \ldots, n$ and $M^{(0)} = M$.

Usually the problem of neural network training is over-parameterized. It means that the number of data points in the learning sequence $(X_j, y_j), j = 1, \ldots, N$, i.e., N is often smaller than the number of the neural network parameters.

From the theoretical point of view, if $N \leq M$ and matrix

$$\chi = \begin{bmatrix} \phi_1^{(0)}(X_1), & \phi_2^{(0)}(X_1), & \ldots, & \phi_M^{(0)}(X_1) \\ & \cdots & \cdots & \\ \phi_1^{(0)}(X_N), & \phi_2^{(0)}(X_N), & \ldots, & \phi_M^{(0)}(X_N) \end{bmatrix} \tag{5}$$

is a matrix of a full rank (i.e., has a rank N), then there exist weights in the last hidden layer $W^{(0)} = (w_1^{(0)}, \ldots, w_M^{(0)})^T$ such that neural networks (1–3) interpolates the learning sequence.

It is well known that the least mean square errors minimization leads in such a case to the partial solution $W^{(0)} = (\chi^T\chi)^{-1}\chi^T Y$, where $Y = (y_1, y_2, \ldots, y_N)^T$. A similar approach is often used when training small RBF networks [12]. Also, extreme learning machines [17] utilize such kinds of ideas. Unfortunately, matrix χ is usually badly conditioned and may lead to significant numerical errors. Although the rest of the neural network weights can be used for preconditioning, the perfect solutions are also undesirable, since interpolating or almost interpolating networks behave poorly on test data (other than training ones).

3 Description of the Algorithm

We will use for FFNN training the Stochastic Gradient Descent (SGD) algorithm with mini-batches of the size MB and additional momentum term [12,21,22].

Traditional SGD picks at random a small set of training examples (i.e., random mini-batch) at each iteration and updates all network weights on the basis of this set only. In random weight subspace optimization we will randomly select (uniformly, but without repetitions) L neurons among all M neurons forming the hidden layer. If the network contains more hidden layers the similar random selection will be repeated on every hidden layer. The weights of all in-going and out-going connections of the selected neurons will be updated. The rest of the neural networks weights will be unchanged. We say that this part of the networks is temporarily frozen. Here we assume that the same weight subspace selection will be used during the whole epoch. Furthermore, we fix that each epoch consists of N/MB iterations. As a consequence, every epoch consists of the same number of learning example presentations, independently of mini-batch size N.

Thus, in every epoch the set of neurons is randomly divided onto two complementary sets: neurons which remain frozen, i.e., are not updated in this epoch, and the rest of the neural network neurons which will be trained in this epoch. More precisely, only their in-going and out-going connections which are not connected to the frozen neurons will be updated.

$$w(t+1) := w(t) + \Delta w(t+1) \tag{6}$$

$$\Delta w(t+1) := \mu\, \Delta w(t) - \alpha\, \nabla E_{MB}/MB, \tag{7}$$

where t is the number of iteration, and $w(t)$ symbolizes weight of any non-frozen network connection at time t. μ stands for the momentum coefficient and α is the learning rate. ∇E_{MB} denotes the stochastic gradient obtained with respect to a randomly selected mini-batch of training samples. It is possible to use momentum in two different ways - the global and the local. In the global momentum regime only $\Delta w(0) = 0$. In the local momentum regime Δw is set to zero at the beginning of each epoch. The last approach allows us to additionally stabilize training.

It may be useful to start with a full neuron set training (without frozen neurons), however it was not necessary in our experiments. It should be noted that in general, stochastic gradient ∇E_{MB} is computed for the whole network. In contrary, the number of the weight updates is strongly reduced depending only on L.

3.1 Learning Rate and Other Training Parameters

We have used a small constant learning rate α during all the training. We established the learning rate value on the basis of experiments. Namely, we have used the learning rate that gives stable convergence in the 10 initial epochs.

3.2 Some Comments About Dropout

The dropout method can be implemented in different ways (see [8,11,15,21,25]) and is usually used for classification (logistic regression). We have applied this

method for comparison reasons during training FFNN for regression. It was important to have similar circumstances when we are freezing some neurons instead of dropping them out. Thus, we have chosen the dropout strategy without any regularization, but accompanied by a larger momentum term, i.e., $\mu = 0.99$ (according to indications given in [21]). To stabilize the training process we have to reset the momentum term at the beginning of every epoch. Otherwise the learning rate should be much smaller than that used by the freezing approach. At test time one single neural network without dropout was used. The outgoing weights of every neuron were scaled down according to the probability $1 - p$, where p is a probability that a given neuron will be temporarily removed from the network. In our experiments $p = 0.5$.

4 Numerical Experiments Results

We have performed SGD training of neural networks with one hidden layer of 100 neurons as a model of the following regression problem.

$$Y = f(X_1, X_2) + \varepsilon = \exp\left[-5X_1 + 1\right] + 0.5 \exp\left[-0.25(11X_2 - 2)^2\right] + \varepsilon, \quad (8)$$

where $X_1, X_2 \in [0, 1]$ are independent uniformly distributed random variables and ε is a Gaussian white noise with variance $\sigma^2 = 0.01$, i.e., $\varepsilon \sim \mathcal{N}(0, 0.1)$. The training sequence consists of 100 independent and identically distributed (i.i.d.) data points. Similarly, the testing set contains also a hundred i.i.d. elements.

We have used stochastic gradient descent with 5 and 10 -sized mini-batches and a traditional mean squared error as a criterion function. Furthermore, apart from MSE for the learning and the testing set also a mean absolute error (MAE) was also monitored. It is easy to check that if we know the true regression function $f(X_1, X_2)$, the expected value of MAE is

$$E\{|Y - f(X_1, X_2)|\} \approx 0.799 \quad (9)$$

and the expected value of MSE is 0.01.

Although it seems that regression model (8) is relatively easy to approximate by a small size FFNN with one hidden layer, in our learning experiments we have have used a neural network equipped with 100 neurons in its hidden layer.

The initial weights are taken uniformly from $[0, 1]$ interval. All experiments shown in Tables 1 and 2 were performed, starting from the same initial point. The columns labeled by (DR 50) contain errors obtained for droput of 50 neurons. Training all 100 neurons was performed without freezing. Monte Carlo errors given in these Tables were computed on the basis of the 100000 random samples. Tables 1 and 2 contain final errors obtained after 5000 epochs of training. The training process details from the first 1000 epochs are depicted in Fig. 1.

The theoretical analysis and many experiments [3,4,9,20] indicate, that that local minima are not a problem for larger size FFNN. Instead of local minima, an extremely large number of saddle points where the gradient is zero is observed. So, regardless of the initial weights, the neural network nearly always

Table 1. Training and testing errors for mini-batch of size 5 after 5000 epochs of training with local momentum $\mu = 0.9$ (0.99 for dropout) and constant learning rate $\alpha = 0.05$

Number of trained neurons	100	10	20	30	40	50	DR 50
Training errors							
MAE_{train}	0.093	0.091	0.091	0.083	0.088	0.091	0.108
MSE_{train}	0.013	0.014	0.013	0.010	0.012	0.013	0.019
Testing errors							
MAE_{test}	0.120	0.128	0.121	0.103	0.117	0.120	0.139
MSE_{test}	0.023	0.025	0.023	0.016	0.021	0.023	0.030
Monte Carlo errors							
MAE_{MC}	0.121	0.123	0.121	0.106	0.112	0.117	0.135
MSE_{MC}	0.023	0.024	0.024	0.018	0.20	0.022	0.030

arrives at solutions of very similar quality, especially if the size of the network is large. Thus, we have performed some additional experiments: for the full network learned without freezing and for the network with $L = 30$, i.e., when 70 neurons from one hundred were frozen in every epoch. In all the cases the size of mini-batch was equal to 5 and each experiment was repeated five times. The averaged results of training after 5000 epochs are given in Table 3. One can see, that the mean errors are very close to that presented in Table 1. According to these facts we restrict further experiments to the one common starting point. Tables 4 and 5 provide the training and testing errors after 10000 epochs of training for mini-batches of size 5 and 10, respectively.

Table 2. Training and testing errors for mini-batch of size 10 after 5000 epochs of training with local momentum $\mu = 0.9$ (0.99 for dropout) and the constant learning rate $\alpha = 0.05$

Number of trained neurons	100	10	20	30	40	50	DR 50
Training errors							
MAE_{train}	0.098	0.100	0.096	0.089	0.089	0.092	0.118
MSE_{train}	0.014	0.015	0.014	0.013	0.012	0.013	0.023
Testing errors							
MAE_{test}	0.123	0.127	0.122	0.113	0.117	0.120	0.149
MSE_{test}	0.024	0.026	0.025	0.021	0.022	0.023	0.035
Monte Carlo errors							
MAE_{MC}	0.118	0.127	0.117	0.107	0.113	0.117	0.146
MSE_{MC}	0.028	0.026	0.022	0.018	0.021	0.022	0.036

Table 3. Averaged training and testing errors for mini-batch of size 5 after 5000 epochs of training with local momentum $\mu = 0.9$ and the constant learning rate $\alpha = 0.05$

Number of trained neurons	MAE training	MSE training	MAE testing	MSE testing
100	0.094	0.014	0.125	0.025
30	0.084	0.011	0.106	0.017

Table 4. Training and testing errors for mini-batch of size 5 after 10000 epochs of training with local momentum $\mu = 0.9$ and the constant learning rate $\alpha = 0.05$

Number of trained neurons	100	10	30	DR 50
Training errors				
MAE_{train}	0.093	0.090	0.089	0.103
MSE_{train}	0.013	0.012	0.012	0.017
Testing errors				
MAE_{test}	0.121	0.121	0.114	0.134
MSE_{test}	0.023	0.024	0.020	0.027
Monte Carlo errors				
MAE_{MC}	0.116	0.115	0.107	0.131
MSE_{MC}	0.022	0.022	0.019	0.028

Table 5. Training and testing errors for mini-batch of size 10 after 10000 epochs of training with local momentum $\mu = 0.9$ (0.99 for dropout) and the constant learning rate $\alpha = 0.05$

Number of trained neurons	100	10	50	DR 50
Training errors				
MAE_{train}	0.096	0.097	0.092	0.103
MSE_{train}	0.013	0.015	0.013	0.018
Testing errors				
MAE_{test}	0.121	0.127	0.120	0.136
MSE_{test}	0.023	0.026	0.023	0.029
Monte Carlo errors				
MAE_{MC}	0.116	0.127	0.115	0.128
MSE_{MC}	0.022	0.026	0.021	0.027

It can be once more observed that SGD is a very slowly convergent method, especially for regression problems. The best results we have obtained for $L = 30$ and a very long training time. It was $MSE_{learn} = 0.0084$, $MAE_{learn} = 0.074$ for the learning sequence and $MSE_{test} = 0.0107$, $MAE_{test} = 0.085$ for testing. The number of epochs was about 23500 for local momentum and 13500 for global momentum. Validity of the solution was confirmed by Monte Carlo simulations

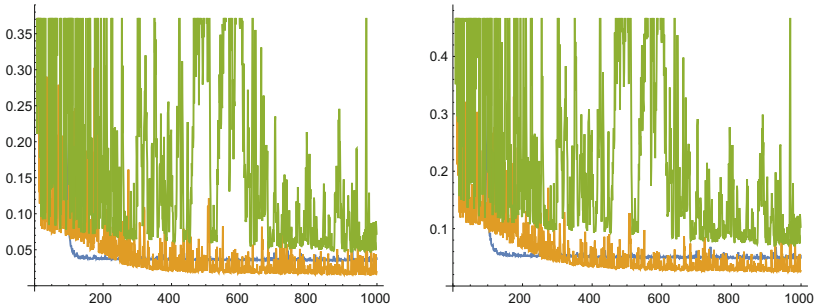

Fig. 1. Results of first 1000 epochs of the training. Left panel: MSE for the learning set. Right panel: MSE for the testing set. Blue – traditional FFNN, yellow – the network with freezing $L = 30$, green – dropout for 50 neurons left. (Color figure online)

using 100000 random observations, namely $MSE_{MC} = 0.011$ and $MAE_{MC} = 0.084$. It was a slightly better result than that obtained for the testing set.

5 Discussion and Conclusions

In this paper we have presented a new neural network training strategy, when in every epoch of training only a randomly chosen and structurally connected network weights are updated. The learning algorithm temporarily keeps constant the rest of the neuron weights. We say that these weights are frozen. We have shown experimentally, that for SGD method of learning this approach can provide better and faster solutions than the classical strategy of the whole network training. It occurred that the proposed method provides a very good generalization and – in contrast to the dropout – it is much more stable and it does not need a regularization.

We have decided to use a testing set of the same size as the learning set. Both of them were relatively small, i.e., the number of samples N was equal to the number of neurons in the hidden layer. This assumption allows us to trustworthy control of the training process. Errors obtained by the Monte Carlo method confirm credibility of the solutions.

It should be noted that in our example the smaller size mini-batches accelerate the training. Furthermore, too large learning rates, even if they do not destabilize the training process, result in larger finally obtainable values of MSE_{train} and MSE_{test}.

In our experiments (without any regularization) freezing gave better results than the dropout method. Nevertheless, further comparisons are needed.

The performed experiments suggest that over-parametrized (with respect to the number of data samples) FFNN models can be learned efficiently without over-fitting symptoms. We claim that the random subspace selection is efficient, although slow in combination with SGD, for large scale multivariate optimization problem. From the stochastic optimization point of view it is important to

provide a safe and trusty method of decreasing the learning rate during the optimization process [18]. However some adaptive methods of changing the learning rate have been known for decades [12], and still there is no satisfactory solution from the point of view of SGD.

Furthermore, it should be noted, that the freezing strategy can be combined not only with SGD type FFNN training algorithms, but also with other popular optimization based training methods, for example, Levenberg-Marquardt, conjugate gradient, quasi-Newton methods and their modifications [10,12].

Acknowledgments. This research was supported by $S50242$ grant at the Faculty of Electronics, Wrocław University of Science and Technology.

References

1. Ba, J., Frey, B.: Adaptive dropout for training deep neural networks. In: Advances in Neural Information Processing Systems (NIPS), pp. 3084–3092 (2013)
2. Bartlett, P.L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Trans. Inf. Theor. **4**(2), 525–536 (1998)
3. Baldi, P., Hornik, K.: Learning from examples without local minima. Neural Netw. **2**(1), 53–58 (1989)
4. Choromanska, A., Henaff, M., Mathieu, M., Arous, G.B., LeCun, Y.: The loss surface of multilayer networks. In: Proceedings of the Conference on AI and Statistics (2014). http://arxiv.org/abs/1412.0233
5. Bengio, Y.: Learning deep architectures for AI. Found. Trends Mach. Learn. **2**(1), 1–127 (2009)
6. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
7. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control, Signals Syst. **2**(4), 303–314 (1989)
8. Dahl, G.E., Sainath, T.N., Hinton, G.E.: Improving deep neural networks for LVCSR using rectified linear units and dropout. In: IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP 2013), Vancouver (2013)
9. Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K.: Identifying and attacking the saddle point problem in highdimensional non-convex optimization. In: Proceedings of Advances in Neural Information Processing Systems, vol. 27, pp. 2933–2941 (2014)
10. Fine, T.: Feedforward Neural Network Methodology. Statistics for Engineering and Information Science. Springer, New York, Inc (1999)
11. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: Proceedings of the 30th International Conference on Machine Learning, pp. 1319–1327. ACM (2013)
12. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall PTR, Upper Saddle River (1998)
13. Hertz, J., Krogh, A., Palmer, R.G.: Introduction to the Theory of Neural Computation. Addison-Wesley, Redwood City (1991)

14. Hinton, G.E., Osindero, S., Teh, Y.: A fast learning algorithm for deep belief nets. Neural Comput. **18**, 1527–1554 (2006)
15. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors (2012). http://arxiv.org/abs/1207.0580
16. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**, 251–257 (1991)
17. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1), 489–501 (2006)
18. Kushner, H., Yin, G.: Stochastic Approximation and Recursive Algorithms and Applications, II edn. Springer-Verlag, New York, Inc (2003)
19. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, New York (2006)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning (Review). Nature **521**, 436–444 (2015)
21. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
22. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA (2013)
23. Schmidhuber, J.: Deep Learning and Neural Networks: An Overview. arXiv:1404.7828 (2014)
24. Qiu, X., Zhang, L., Ren, Y., Suganthan, P., Amaratunga, G.: Ensemble deep learning for regression and time series forecasting. 2014 IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL), pp. 1–6 (2014). doi:10.1109/CIEL.2014.7015739
25. Wang, S.I., Manning, C.D.: Fast dropout training. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA (2013)