OPTML 2017: Lifted Neural Networks for Weight Initialization

Geoffrey Negiar* geoffrey_negiar@berkeley.edu
Armin Askari* aaskari@berkeley.edu
Fabian Pedregosa fabianp@berkeley.edu
Laurent El Ghaoui elghaoui@berkeley.edu
EECS Department
University of California
Berkeley, CA 94720-1776, USA
* *equal contribution*

Abstract

We describe a novel model for supervised learning based on the multi-layer feedforward neural network, where the activation functions are encoded via penalties in the training problem, transforming non-smooth activation functions such as ReLUs into smooth optimization problems. The new framework is particularly amenable to block-coordinate algorithms where each step is composed of simple, convex problems. Preliminary experiments show that using the weights learned in our model as initialization for a traditional feedforward network improves generalization accuracy on the MNIST dataset.

1 Introduction

We consider the setting in which are given an input data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ consisting of n samples and m features, and a response matrix $\boldsymbol{Y} \in \mathbb{R}^{n \times p}$ with an output of p features. The classical supervised feedforward neural network model can be defined as the solution to the following optimization problem:

$$\begin{aligned} \min_{(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=0}^{L}, (\boldsymbol{X}_l)_{l=1}^{L+1}} \quad & \mathcal{L}(\boldsymbol{Y}, \boldsymbol{X}_{L+1}) + \textstyle\sum_{l=0}^{L} \pi_l(\boldsymbol{W}_l) \\ \text{s.t.} \quad & \boldsymbol{X}_{l+1} = \phi_l(\boldsymbol{X}_l \boldsymbol{W}_l + \mathbf{1}\boldsymbol{b}_l^T), \;\; 0 \leq l \leq L, \;\; \boldsymbol{X}_0 = \boldsymbol{X}. \end{aligned} \tag{1}$$

where $(\boldsymbol{W}_l, \boldsymbol{b}_l)$ are sequences of weight matrices and bias vectors, (ϕ_l) is a sequence of activation functions that act element-wise on matrix inputs, $\mathbf{1}$ is the vector of ones of the relevant size, (π_l) is a sequence of penalizing functions, typically an ℓ_2 or ℓ_1 norm and $\mathcal{L}$ is a loss such as the squared or softmax loss function. Usual notation of the problem compresses the equality constraints, making invisible the $(\boldsymbol{X}_l)$ sequence.

The development of optimization methods for neural networks has been an active research topic in the last decade with the development of specialized gradient-based algorithms such as ADAM and Adagrad KingmaB14Adam, DBLP:journals/jmlr/DuchiHS11. Although these algorithms perform well in practice, how these should be initialized remains unclear. [4, 1] recommend sampling from a uniform distribution to initialize weights and biases while others either use either random initialization or weights learned in other networks (transfer learning) on different unsupervised or supervised tasks. [8] indicate that initialization is crucial during training and that poorly initialized networks cannot be trained with momentum.

In this paper, we develop a family of alternate models whose weights and biases are used to initialize traditional feedforward networks. These new models mimic the standard problem (1) and allow for efficient algorithms to be used during training. Although the model itself can be used for a supervised or unsupervised learning task, we have found that the method excels most when used as initialization of traditional feedforward networks.

Related Work. Other methods to initialize neural networks have been proposed such as using competitive learning [5] and principal component analysis (PCA) [7]. Although initializing using PCA produces state of the art results, it is limited to auto-encoders while our framework allows for more general learning problems. Similarly, the competitive learning approach is limited to the classification problem and works only for one layer networks while our model can easily be adapted to a broader range of network architectures. Our approach focuses on transforming the non-smooth optimization problem encountered when fitting neural network models into a smooth problem in an enlarged space, which ties to a well developed branch of optimization literature (see *e.g.* section 5.2 of Bubeck:2015:COA:2858997.2858998 and references therein). Our approach can also

OPTML 2017: 10th NIPS Workshop on *Optimization for Machine Learning* (NIPS 2017).

be seen as a generalization of the PReLU proposed by DBLP:journals/corr/HeZR015prelu. Our work can be compared to the standard practice of initializing Gaussian Mixture Models using K-Means clustering: using a simpler but similar algorithm for initialization.

2 Lifted Framework

The key observation behind our model lies in the fact that a given activation function $\phi : \mathbb{R}^p \to \mathbb{R}^q$ can be represented implicitly as the result of a certain optimization problem involving a bi-convex function. More precisely, we assume that there exists a function $D_\phi : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$ with the following two properties: (a) D_ϕ is convex in its first argument when the second is fixed and vice-versa; (b) for any given vector $\boldsymbol{u} \in \mathbb{R}^p$:

$$\phi(\boldsymbol{u}) = \arg\min_{\boldsymbol{v}} \; D_\phi(\boldsymbol{u}, \boldsymbol{v})\,. \tag{2}$$

Most common activation functions can be represented in this form. In this paper, we focus on the rectified linear unit or ReLU, which is the activation behind many recent breakthroughts in deep learning [2]. It can be written in the above form using the following jointly convex function D_{relu}:

$$\phi_{\text{relu}}(\boldsymbol{u}) := \max(\boldsymbol{u}, \boldsymbol{0}) = \arg\min_{\boldsymbol{v}\,:\,\boldsymbol{v}\geq\boldsymbol{0}} \; \|\boldsymbol{v} - \boldsymbol{u}\|_2^2\,, \tag{3}$$

where the max and the positive constraint are component-wise operations. We now define a new model in which the activation function in the standard model (1) are replaced with the the representation (2). We call the result of such optimization problems *lifted neural networks*. These are of the form

$$\begin{aligned} \min_{(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=0}^{L}, (\boldsymbol{X}_l)_{l=1}^{L+1}} \quad & \mathcal{L}(\boldsymbol{Y}, \boldsymbol{X}_{L+1}) + \sum_{l=0}^{L} \Big(D_{l+1}(\boldsymbol{X}_l \boldsymbol{W}_l + \boldsymbol{1}_m \boldsymbol{b}_l^T, \boldsymbol{X}_{l+1}) + \pi_l(\boldsymbol{W}_l) \Big) \\ \text{s.t.} \quad & \boldsymbol{X}_0 = \boldsymbol{X} \quad , \end{aligned} \tag{4}$$

where D_l is the function in (2) associated with the activation ϕ_l. In the case of regression, we can represent the last layer with the function $D_{L+1}(\boldsymbol{Y}, \boldsymbol{Z}) = \|\boldsymbol{Y} - \boldsymbol{Z}\|_F^2$; in the case of classification, we may use a cross entropy loss. Despite the increased number of variables, the new model has the following useful characteristics:

- For any $l = 1, \ldots, L$, the problem is convex in $\boldsymbol{X}_l$ when all the other variables (weights $(\boldsymbol{W}_j, \boldsymbol{b}_j)_{j=0}^{L}$, and $(\boldsymbol{X}_j)_{j\neq l}$) are fixed.
- When the functions D_l are jointly convex, as is the case with ReLUs and a host of other activations, for fixed $\boldsymbol{W}$-variables, the problem is convex in $(\boldsymbol{X}_l)_{l=1}^{L}$.
- For fixed $\boldsymbol{W}$-variables (weights $\boldsymbol{W}_l, \boldsymbol{b}_l$), the problem is convex in the $\boldsymbol{X}$-variables $\boldsymbol{X}_l$, $l = 1, \ldots, L$.
- Likewise, for fixed $\boldsymbol{X}$-variables, the problem is convex in the $\boldsymbol{W}$-variables.

These characteristics allow for efficient block-coordinate descent methods to be applied to our learning problem. Each step reduces to a convex optimization problem, such as ridge regression, non-negative least-squares, or sparse logistic regression. We describe one algorithm in section 2.1.

The prediction rule in this setting, i.e., how to generate an output $\widehat{\boldsymbol{Y}}$ for a given $\boldsymbol{X}$, will be different from that of a standard neural network. It is however based on the same principle: the prediction rule can be obtained by solving the problem (4), where the weights are now *fixed*, the input data is replaced by the test point, and the predicted value is a *variable*. In our new framework, the resulting problem is convex and can be solved efficiently. In practice, we have found that applying the feedforward prediction rule, i.e. using the output of a layer as input to the next, also gives similar performance.

2.1 Block-coordinate descent (BCD) algorithm

As a specific example, consider a regression network where each activation is a ReLU, except the last, which has no activation other than a squared loss. Using the representation (3), the corresponding model, which we refer to as *chained ridge regression*, is

$$\begin{aligned} \min_{(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=0}^{L}, (\boldsymbol{X}_l)_{l=1}^{L}} \quad & \|\boldsymbol{Y} - \boldsymbol{X}_L \boldsymbol{W}_L - \boldsymbol{b}_l \boldsymbol{1}^T\|_F^2 + \lambda \textstyle\sum_{l=0}^{L-1} \Big(\|\boldsymbol{X}_{l+1} - \boldsymbol{X}_l \boldsymbol{W}_l - \boldsymbol{1}\boldsymbol{b}_l^T\|_F^2 + \rho_l \|\boldsymbol{W}_l\|_F^2 \Big) \\ \text{s.t.} \quad & \boldsymbol{X}_l \geq 0, \;\; l = 1, \ldots, L, \;\; \boldsymbol{X}_0 = \boldsymbol{X}. \end{aligned} \tag{5}$$

where $\lambda > 0$ is a hyperparameter. The model has the computationally friend characteristics listed in the previous section. In addition, the sub-problems involved are particularly simple, as we now detail.

Optimizing over W-variables. The update in the weighting matrices $(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=0}^{L}$ is a set of simple least-squares ridge regression problems. More precisely, at layer l we solve the subproblem:

$$\min_{\boldsymbol{W},\boldsymbol{b}} \| \begin{pmatrix} \boldsymbol{X}_l & \mathbf{1} \end{pmatrix} \begin{pmatrix} \boldsymbol{W}_l \\ \boldsymbol{b}_l^T \end{pmatrix} - \boldsymbol{X}_{l+1} \|_F^2 + \rho_l \|\boldsymbol{W}_l\|_F^2 .$$

This has a closed-form solution and can be greatly improved by using sketching methods for least-squares problems, as seen in DBLP:journals/corr/Woodruff14,pilanci2016iterative. The updates can be done all in parallel across the L layers, since each is independent of the others.

Optimizing over X-variables. In this step we minimize over the matrices $(\boldsymbol{X}_l)_{l=1}^{L+1}$ cyclically as well. For a given $l \in \{1, \ldots, L\}$, the minimization over $\boldsymbol{X}_l$ with $\boldsymbol{X}_j$, $j \neq l$ fixed, reduces to a problem of the form

$$\boldsymbol{X}_l = \arg\min_{\boldsymbol{Z} \geq 0} \|\boldsymbol{Z}\boldsymbol{W}_l - \boldsymbol{X}_{l+1} - \mathbf{1}\boldsymbol{b}_{l+1}^T\|_F^2 + \|\boldsymbol{Z} - \boldsymbol{X}_{l-1}\boldsymbol{W}_{l-1} - \mathbf{1}\boldsymbol{b}_{l-1}^T\|_F^2.$$

A similar result holds for the last layer, for a general loss $\mathcal{L}$. The above is a (matrix) non-negative least-squares, for which many efficeint methods are available such as [6, 3]. The cost of updating all columns can be reduced by taking into account that all columns' updates share the same coefficient matrix W_l.

Convergence guarantees. Contrary to the classical feedforward networks with ReLU activations, our loss is a smooth loss function with convex constraints. Hence, it is possible to use existing results on the convergence of block coordinate descent to obtain global convergence guarantees to a stationary point xu2013block and an $\mathcal{O}(1/\sqrt{t})$ convergence rate in expectation for the gradient mapping in the case of randomized coordinate updates patrascu2015efficient .

3 Numerical Results

The model described in this paper was compared against a traditional neural network with equivalent architectures on the MNIST dataset. For the classification problem, the dataset was split into 60,000 training samples and 10,000 test samples with a softmax cross entropy loss. This is a similar model to the one specified in (5), with the only difference that the last layer loss is changed from an ℓ_2 loss to a softmax cross entropy loss. In addition to comparing the models, the weights and biases learned in the augmented neural network were used as initialization parameters for training a standard neural net to compare their performance, both in classification and convergence during training. For all models, ReLU activations were used. Table (1) summarizes the error rates for the different architectures.

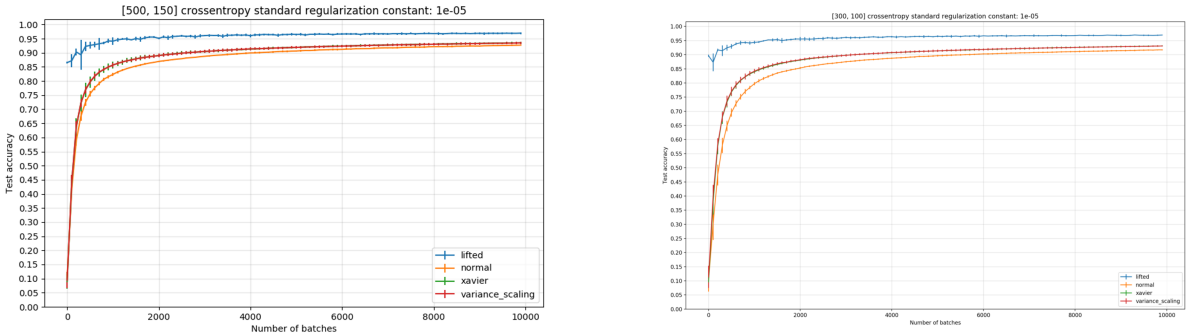
Architecture	Our Model	NN [random]	NN [init]
$28 \times 28 - 300 - 10$	0.102 ± 0.001	0.022 ± 0.001	$\mathbf{0.0210 \pm 0.0017}$
$28 \times 28 - 1000 - 10$	0.096 ± 0.004	0.019 ± 0.001	$\mathbf{0.0182 \pm 0.0007}$
$28 \times 28 - 300 - 100 - 10$	0.139 ± 0.003	0.071 ± 0.015	$\mathbf{0.0224 \pm 0.0005}$
$28 \times 28 - 500 - 150 - 10$	0.128 ± 0.002	0.080 ± 0.025	$\mathbf{0.0218 \pm 0.0005}$
$28 \times 28 - 500 - 300 - 150 - 100 - 10$	0.148 ± 0.002	0.83 ± 0.07	$\mathbf{0.0223 \pm 0.0005}$

Table 1: Error rate on the test set using different networks, best result is highlighted in boldface. Two 1-layer networks with 300, 1000 hidden units, two 2-layer 500-150, 300-100 hidden units and one 4-layer network were tested. NN[random] is a standard neural network with random initialization while NN[init] is a neural network initialized with the weights and biases learned from training our model.The neural networks were trained for 20 epochs using RMSprop in Tensorflow tensorflow2015-whitepaper.

Remark 1 *In our experiments for the 4 layer model, all our randomly initialized runs converged to bad local minima, yielding* 83% *classification error on average. All experiments were run for* 20 *epochs, i.e.* 5000 *batches. Our initialization yields much more stable training.*

Remark 2 *Although our model does not perform as good as the other models on this task, using it as initialization results in increased accuracy for all network architectures.*

Our results indicate that for networks with more layers, our initialization improves the convergence towards a stationary point which has a lower cost than the local minimum attained from random initialization. As seen in the two layer and four layer networks, our initialization significantly improves the error rate on the test set while for the one layer networks gives only slight improvements.



batches

Figure 1: Plot of accuracy (%) vs number of batch iterations on a held-out validation set during training for two different architectures. The batch size was fixed at 256. **Left:** Neural network composed of 2 ReLU layers with 500, 150 hidden units respectively. **Right:** Two layer neural network with 300 - 100 hidden units and ReLU activation.

4 Conclusion and Future Work

In this work we have proposed a novel model for supervised learning. Our method replaces the potentially nonsmooth activation function on multi-layer neural networks by an optimization problem in an enlarged ("lifted") space. We applied this technique to build a model which we later use as initialization on feedforward neural networks with ReLU activations.

Exeperimental results have shown that the weights of our trained model serve as a good initialization for the parameters of classical neural networks, outperforming deep neural networks with random initialization. Furthermore, our approach obtains the largest margin of improvement on very deep networks, a very challenging case from the optimization perspective. In future work will extend our framework beyond the ReLU activation function and study its applicability to other neural network architectures such as convolutional and recurrent networks in our framework and using the parameters of our model as initialization for those networks.

References

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Dongmin Kim, Suvrit Sra, and Inderjit S. Dhillon. A non-monotonic method for large-scale non-negative least squares. *Optimization Methods and Software*, 28(5):1012–1039, 2013.

[4] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[5] Richard Maclin and Jude W. Shavlik. Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[6] Mert Pilanci and Martin J Wainwright. Iterative Hessian sketch: Fast and accurate solution approximation for constrained least-squares. *The Journal of Machine Learning Research*, 17(1), 2016.

[7] Mathias Seuret, Michele Alberti, Rolf Ingold, and Marcus Liwicki. PCA-initialized deep neural networks applied to document image analysis. *CoRR*, abs/1702.00177, 2017.

[8] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, 2013.