Lifted Neural Networks

Armin Askari
Geoff Negiar
Rajiv Sambarhya
Laurent El Ghaoui

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-218
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-218.html

October 8, 2021

Lifted Neural Networks

Armin Askari [1] Geoffrey Negiar [1] Rajiv Sambharya [1] Laurent El Ghaoui [1 2]

Abstract

We describe a novel family of models of multi-layer feedforward neural networks in which the activation functions are encoded via penalties in the training problem. Our approach is based on representing a non-decreasing activation function as the argmin of an appropriate convex optimization problem. The new framework allows for algorithms such as block-coordinate descent methods to be applied, in which each step is composed of a simple (no hidden layer) supervised learning problem that is parallelizable across data points and/or layers. Experiments indicate that the proposed models provide excellent initial guesses for weights for standard neural networks. In addition, the model provides avenues for interesting extensions, such as robustness against noisy inputs and optimizing over parameters in activation functions.

1. Introduction

Given current advances in computing power, dataset sizes and the availability of specialized hardware/ software packages, the popularity of neural networks continue to grow. The model has become standard in a large number of tasks, such as image recognition, image captioning and machine translation. Current state of the art is to train this model by variations of stochastic gradient descent (SGD), although these methods have several caveats. Most problems with SGD are discussed in (Taylor et al., 2016).

Optimization methods for neural networks has been an active research topic in the last decade. Specialized gradient-based algorithms such as Adam and ADAGRAD (Kingma & Ba, 2015; Duchi et al., 2011) are often used but were shown to generalize less than their non adaptive counterparts by (Wilson et al., 2017). Our work is related to two main currents of research aimed at improving neural network optimization: using non gradient-based approaches and initializing weights to accelerate convergence of gradient-based algorithms. To our knowledge this paper is the first to combine the two. In addition our novel formalism allow for interesting extensions towards handling constraints, robustness, optimizing network topology, etc.

(Taylor et al., 2016) and (Carreira-Perpinan & Wang, 2014) propose an approach similar to ours, adding variables in the training problem and using an l^2-norm penalization of equality constraints. They both break down the network training problem into easier sub-problems and use alternate minimization; however they do not exploit structure in the activation functions. For Convolutional Neural Networks (CNN), (Berrada et al., 2016) model the network training problem as a difference of convex functions optimization, where each subproblem is a Support Vector Machine (SVM).

On the initialization side, (LeCun et al., 1998; Glorot & Bengio, 2010) recommend sampling from a well-chosen uniform distribution to initialize weights and biases while others either use random initialization or weights learned in other networks (transfer learning) on different tasks. (Sutskever et al., 2013) indicate that initialization is crucial during training and that poorly initialized networks cannot be trained with momentum. Other methods to initialize neural networks have been proposed, such as using competitive learning (Maclin & Shavlik, 1995) and principal component analysis (PCA) (Seuret et al., 2017). Although 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; this ties to a well developed branch of optimization literature (see *e.g.* section 5.2 of (Bubeck, 2015) and references therein). Our approach can also be seen as a generalization of the parameterized rectified linear unit (PReLU) proposed by (He et al., 2015). Our work can be compared to the standard practice of initializing Gaussian Mixture Models using K-Means clustering; our

[*] Equal contribution [1] Department of Electrical Engineering and Computer Science, University of California — Berkeley, United States [2] Department of Industrial Engineering and Operations Research, University of California — Berkeley, United States. Correspondence to: Armin Askari <aaskari@berkeley.edu>, Geoffrey Negiar <geoffrey_negiar@berkeley.edu>.

model uses a simpler but similar algorithm for initialization.

Paper outline. In Section 2, we begin by describing the mathematical setting of neural networks and our proposed optimization problem to train the model. Section 3 provides an example illustrating the basic idea. Section 4 outlines how to encode activation functions as argmins of convex or bi-convex optimization problems. Section 5 then expands the approach of Section 3 to cover a number of useful activation functions, as well as classification tasks. Section 6 describes a block-coordinate descent method to solve the training problem. Section 7 describes numerical experiments that support a finding that the models can be used as a fast weight initialization scheme.

2. Background and Notation

Feedforward neural networks. We begin by establishing notation. We are given an input data matrix $X = [x_1, \ldots, x_m] \in \mathbb{R}^{n \times m}$ and response matrix $Y \in \mathbb{R}^{p \times m}$ and consider a supervised problem involving a neural network having $L \geq 1$ hidden layers. At test time, the network processes an input vector $x \in \mathbb{R}^n$ to produce a predicted value $\hat{y}(x) \in \mathbb{R}^p$ according to the prediction rule $\hat{y}(x) = x_{L+1}$ where x_{L+1} is defined via the recursion

$$x_{l+1} = \phi_l(W_l x_l + b_l), \quad l = 0, \ldots, L, \tag{1}$$

with initial value $x_0 = x \in \mathbb{R}^n$ and $x_l \in \mathbb{R}^{p_l}$, $l = 0, \ldots, L$. Here, ϕ_l, $l = 1, \ldots, L$ are given activation functions, acting on a vector; the matrices $W_l \in \mathbb{R}^{p_{l+1} \times p_l}$ and vectors $b_l \in \mathbb{R}^{p_{l+1}}$, $l = 0, \ldots, L$ are parameters of the network. In our setup, the sizes $(p_l)_{l=0}^{L+1}$ are given with $p_0 = n$ (the dimension of the input) and $p_{L+1} = p$ (the dimension of the output).

We can express the predicted outputs for a given set of m data points contained in the $n \times m$ matrix X as the $p \times m$ matrix $\hat{Y}(X) = X_{L+1}$, as defined by the matrix recursion

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}^T), \quad l = 0, \ldots, L, \tag{2}$$

with initial value $X_0 = X$ and $X_l \in \mathbb{R}^{p_l \times m}$, $l = 0, \ldots, L$. Here, $\mathbf{1}$ stands for the vector of ones in $\mathbb{R}^m$, and we use the convention that the activation functions act column-wise on a matrix input.

In a standard neural network, the matrix parameters of the network are fitted via an optimization problem, typically of the form

$$\begin{aligned} \min_{(W_l,b_l)_0^L,(X_l)_1^L} \quad & \mathcal{L}(Y, X_{L+1}) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ \text{s.t.} \quad & X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}_m^T), \quad l = 0, \ldots, L \\ & X_0 = X \end{aligned} \tag{3}$$

where $\mathcal{L}$ is a loss function, $\rho \in \mathbb{R}_+^{L+1}$ is a hyper-parameter vector, and π_l's are penalty functions which can be used to encode convex constraints, network structure, etc. We refer to the collections $(W_l, b_l)_{l=0}^L$ and $(X_l)_{l=1}^L$ as the (W, b)- and X-variables, respectively.

To solve the training problem (3), the X-variables are usually eliminated via the recursion (2), and the resulting objective function of the (W, b)-variables is minimized without constraints, via stochastic gradients. While this appears to be a natural approach, it does make the objective function of the problem very complicated and difficult to minimize.

Lifted models. In this paper, we develop a family of models where the X-variables are kept, and the recursion constraints (1) are approximated instead, via penalties. We refer to these models as "lifted" because we lift the search space of (W, b)-variables to a higher-dimensional space of (W, b, X)-variables. The training problem is cast in the form of a matrix factorization problem with constraints on the variables encoding network structure and activation functions.

Lifted models have many more variables but a much more explicit structure than the original, allowing for training algorithms that can use efficient standard machine learning libraries in key steps. The block-coordinate descent algorithm described here involves steps that are parallelizable across either data points and/or layers; each step is a simple structured convex problem.

The family of alternate models proposed here have the potential to become competitive in their own right in learning tasks, both in terms of speed and performance. In addition, such models are versatile enough to tackle problems deemed difficult in a standard setting, including robustness to noisy inputs, adaptation of activation functions to data, or including constraints on the weight matrices. Our preliminary experiments are limited to the case where the lifted model's variables are used as initialization of traditional feedforward network. However, we discuss and layout the framework for how these models can be used to tackle other issues concerning traditional networks such as robustness and optimizing how to choose activation functions at each layer.

3. Basic Idea

To describe the basic idea, we consider a specific example, in which all the activation functions are the ReLUs, except for the last layer. There ϕ_L is the identity for regression tasks or a softmax for classification tasks. In addition, we assume in this section that the penalty functions are of the form $\pi_l(W) = \|W\|_F^2$, $l = 0, \ldots, L$.

We observe that the ReLU map, acting componentwise on a vector input u, can be represented as the "argmin" of an optimization problem involving a jointly convex function:

$$\phi(u) = \max(0, u) = \arg\min_{v \geq 0} \; \|v - u\|_2. \tag{4}$$

As seen later, many activation functions can be represented as the "$\arg\min$" of an optimization problem, involving a jointly convex or bi-convex function.

Extending the above to a matrix case yields that the condition $X_{l+1} = \phi(W_l X_l + b_l \mathbf{1}^T)$ for given l can be expressed via an "$\arg\min$":

$$X_{l+1} \in \arg\min_{Z \geq 0} \; \|Z - W_l X_l - b_l \mathbf{1}^T\|_F^2.$$

This representation suggests a heuristic to solve (3), replacing the training problem by

$$\begin{aligned} \min_{(W_l, b_l), (X_l)} \quad & \mathcal{L}(Y, W_L X_L + b_L \mathbf{1}^T) + \sum_{l=0}^{L} \rho_l \|W_l\|_F^2 \\ & + \sum_{l=0}^{L-1} \left(\lambda_{l+1} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F^2 \right) \\ \text{s.t.} \quad & X_l \geq 0, \;\; l = 1, \ldots, L-1, \;\; X_0 = X. \end{aligned} \tag{5}$$

where $\lambda_{l+1} > 0$ are hyperparameters, ρ_l are regularization parameters as in (3) and $\mathcal{L}$ is a loss describing the learning task. In the above model, the activation function is not used in a pre-defined manner; rather, it is *adapted* to data, via the non-negativity constraints on the "state" matrices $(X_l)_{l=1}^{L+1}$. We refer to the above as a "lifted neural network" problem.

Thanks to re-scaling the variables with $X_l \rightarrow \sqrt{\lambda_l} X_l$, $W_l \rightarrow \sqrt{\lambda_{l+1}/\lambda_l} W_l$, and modifying ρ_l's accordingly, we can always assume that all entries in λ are equal, which means that our model introduces just one extra scalar hyperparameter over the standard network (3).

The above optimization problem is, of course, challenging, mainly due to the number of variables. However, for that price we gain a lot of insight on the training problem. In particular, the new model has the following useful characteristics:

- For fixed (W, b)-variables, the problem is convex in the X-variables X_l, $l = 1, \ldots, L$; more precisely it is a (matrix) non-negative least-squares problem. The problem is fully *parallelizable across the data points*.
- Likewise, for fixed X-variables, the problem is convex in the (W, b)-variables and *parallelizable across layers and data points*. In fact, the (W, b)-step is a set of parallel (matrix) ridge regression problems.

These characteristics allow for efficient block-coordinate descent methods to be applied to our learning problem. Each step reduces to a basic supervised learning problem, such as ridge regression or non-negative least-squares. We describe one algorithm in more detail in section 6.

The reader may wonder at this point what is the prediction rule associated with our model. For now, we focus on extending the approach to broader classes of activations and loss functions used in the last layer; we return to the prediction rule issue in our more general setting in section 5.2.

4. Activations as $\arg\min$ Maps

In this section, we outline theory on how to convert a class of functions as the "$\arg\min$" of a certain optimization problem, which we then encode as a penalty in the training problem. We make the following assumption on a generic activation function ϕ.

BCR Condition. The activation function $\phi : \mathbb{R}^k \rightarrow \mathbb{R}^j$ satisfies the bi-convex representation (BCR) condition if it can be represented as follows:

$$\forall x \in \mathbb{R}^k, \; \phi(x) = \arg\min_{z \in \mathbb{R}^j} \; \mathcal{D}_\phi(x, z),$$

where $\mathcal{D}_\phi : \mathbb{R}^k \times \mathbb{R}^j \rightarrow \mathbb{R}$ is a bi-convex function (convex in x for fixed z and vice-versa), which is referred to as a *BC-divergence* associated with the activation function.

We next examine a few examples, all based on divergences of the form

$$D_\phi(x, z) = \Phi(z) - x^T z,$$

where Φ is a convex function. This form implies that, when Φ is differentiable, ϕ is the gradient map of a convex function; thus, it is monotone.

Strictly monotone activation functions. We assume that ϕ is strictly monotone, say without loss of generality, strictly increasing. Then, it is invertible, and there exists a function, denoted ϕ^{-1}, such that the condition $x = \phi^{-1}(z)$ for $z \in \mathbf{range}(\phi)$ implies $z = \phi(x)$. Note that ϕ^{-1} is strictly increasing on its domain, which is $\mathbf{range}(\phi)$.

Define the function $\Phi : \mathbb{R} \rightarrow \mathbb{R}$, with values

$$\Phi(z) = \int_0^z \phi^{-1}(u) \, du \text{ if } z \in \mathbf{range}(\phi), \tag{6}$$

and $+\infty$ otherwise.

The function Φ is convex, since ϕ^{-1} is increasing. We then consider the problem

$$\min\{ \; \Phi(z) - xz \; : \; z \in \mathbf{range}(\phi) \}. \tag{7}$$

Note that the *value* of the problem is nothing else than $\Phi^*(x)$, where Φ^* is the Fenchel conjugate of Φ.

By construction, the problem (7) is convex. At optimum, we have $x = \phi^{-1}(z)$, hence $z = \phi(x)$. We have obtained

$$\phi(x) = \arg\min_z \ \Phi(z) - xz \ : \ z \in \textbf{range}(\phi).$$

Examples. As an example, consider the sigmoïd function:

$$\phi(x) = \frac{1}{1+e^{-x}},$$

with inverse

$$\phi^{-1}(z) = \log\frac{z}{1-z}, \ \ 0 < z < 1,$$

and $+\infty$ otherwise.

Via the representation result (6), we obtain

$$\phi(x) = \arg\min_{0 \le z \le 1} \ z\log z + (1-z)\log(1-z) - xz$$

Next consider the "leaky ReLU" function

$$\phi(x) = \left\{ \begin{array}{ll} \alpha x & \text{if } x < 0, \\ x & \text{if } x \ge 0, \end{array} \right.$$

where $0 < \alpha < 1$. We have

$$\phi^{-1}(z) = \left\{ \begin{array}{ll} (1/\alpha)z & \text{if } z < 0, \\ z & \text{if } z \ge 0, \end{array} \right.$$

with domain the full real line; thus

$$\begin{aligned} \Phi(z) &= \int_0^z \phi^{-1}(u)\, du \\ &= \frac{1}{2}\max\left(\frac{1}{\alpha}\max(0,-z)^2, \max(0,z)^2\right) \end{aligned} \tag{8}$$

As another example, consider the case with $\phi(x) = \text{arctanh}(x)$. The inverse function is

$$\phi^{-1}(z) = \frac{1}{2}\log\frac{1+z}{1-z}, \ \ |z| \le 1$$

For any $z \in [-1,1]$, $\Phi(z)$ takes the form

$$\begin{aligned} \Phi(z) &= \frac{1}{2}\int_0^z \left(\log(1+u) - \log(1-u)\right)\, du \\ &= \frac{1}{2}\left((1-z)\log(1-z) + (1+z)\log(1+z)\right) + \text{cst.} \end{aligned}$$

Sometimes there are no closed-form expressions. For example, for the so-called "softplus" function $\phi(x) = \log(1+e^x)$, the function Φ cannot be expressed in closed form:

$$\Phi(z) = \int_0^z \log(e^u - 1)\, du, \ \ \textbf{dom}\Phi = \mathbb{R}_+.$$

This lack of a closed-form expression does not preclude algorithms from work with these types of activation functions. The same is true of the sigmoid function.

Non-strictly monotone examples: ReLU and piece-wise sigmoid. The above expression (7) works in the ReLU case; we simply restrict the inverse function to the domain $\mathbb{R}_+$; specifically, we define

$$\phi^{-1}(z) = \left\{ \begin{array}{ll} +\infty & \text{if } z < 0, \\ z & \text{if } z \ge 0, \end{array} \right.$$

We then have $\textbf{dom}\Phi = \mathbb{R}_+$, and for $z \ge 0$:

$$\Phi(z) = \int_0^z u\, du = \frac{1}{2}z^2.$$

We have obtained

$$\phi(x) = \arg\min_{z \ge 0} \ \Phi(z) - xz.$$

The result is consistent with the "leaky" ReLU case in the limit when $\alpha \to 0$. Indeed, in that case with Φ given as in (8), we observe that when $\alpha \to 0$ the domain of Φ collapses from the whole real line to $\mathbb{R}_+$, and the result follows.

In a similar vein, consider the "piecewise" sigmoid function,

$$\phi(x) = \min(1, \max(-1, x)),$$

This function can be represented as

$$\phi(x) = \arg\min_z \ z^2 - 2xz \ : \ |z| \le 1.$$

Finally the sign function is represented as

$$\textbf{sign}(x) = \arg\min \ -zx \ : \ |z| \le 1.$$

Non-monotone examples. The approach can be sometimes extended to non-monotonic activation functions. As an example, the activation function $\phi(x) = \sin x$ has been proposed in the context of time-series. Here, we will work with

$$\Phi(z) = \int_0^z \arcsin(u)\, du = z\arcsin z + \sqrt{1-z^2} + \text{cst.},$$

with domain $[-1,1]$. The function is convex, and we can check that

$$\phi(x) = \arg\min_{z\,:\,|z|\le 1} \ \Phi(x) - xz$$

Jointly convex representations. Some activation functions enjoy a stronger condition, which in turn leads to improved properties of the corresponding lifted model.

> **JCR Condition.** The activation function $\phi : \mathbb{R}^k \to \mathbb{R}^j$ satisfies the jointly convex representation (JCR) condition if it satisfies the CR condition with a jointly convex function $\mathcal{D}_\phi(x,z)$.

Note that, for the JCR condition to hold, the activation function needs to be monotone. Because of non-uniqueness, we may add a term that is not a function of the variable being optimized (i.e. in the condition above, an arbitrary function of u) to the JC-divergence in order to improve the overall structure of the problem. This is highlighted below and discussed in Section 6.

The JCR condition applies to several important activation functions, beyond the ReLU, for which

$$\max(x,0) = \arg\min_z \mathcal{D}_\phi(x,z) = \begin{cases} \|x-z\|_2^2 & \text{if } z \geq 0, \\ +\infty & \text{otherwise.} \end{cases}$$

Note that the JC-divergence for the ReLU is not unique; for example, we can replace the l_2-norm by the l_1-norm.

The "leaky" ReLU with parameter $\alpha \in (0,1)$, defined by $\phi(x) = \max(x/\alpha, x)$, can be written in a similar way:

$$\max(x/\alpha, x) = \arg\min_z \ \|x-z\|_2^2 \ : \ z \geq (1/\alpha)x.$$

The piece-wise sigmoïd, as defined below, has a similar variational representation: with $\mathbf{1}$ the vector of ones,

$$\min(1, \max(0,x)) = \arg\min_z \ \|x-z\|_2^2 \ : \ 0 \leq z \leq \mathbf{1}.$$

In order to address multi-class classification problems, it is useful to consider a last layer with an activation function that produces a probability distribution output. To this end, we may consider an activation function which projects, with respect to some metric, a vector onto the probability simplex. The simplest example is the Euclidean projection of a real vector $u \in \mathbb{R}^k$ onto the probability simplex in $\mathbb{R}^k$:

$$\phi(x) = \arg\min_z \ \|x-z\|_2^2 \ : \ z \geq 0, \ \ z^T\mathbf{1} = 1.$$

Max-pooling operators are often used in the context of image classification. A simple example of a max-pooling operator involves a p-vector input x with two blocks, $x = (x^{(1)}, x^{(2)})$, with $x(i) \in \mathbb{R}^{p_i}$, $i = 1,2$, with $p = p_1 + p_2$. We define $\phi \ : \ \mathbb{R}^p \to \mathbb{R}^2$ by

$$\phi(x) = (\max_{1 \leq i \leq p_1} \ x_i^{(1)}, \max_{1 \leq i \leq p_2} \ x_i^{(2)}) \in \mathbb{R}^2. \tag{9}$$

Max-pooling operators can also be expressed in terms of a jointly convex divergence. In the above case, we have

$$\phi(x) = \arg\min_z \ \mathbf{1}^T z + \mathbf{1}^T(x - Dz)_+,$$

where D is an appropriate block-diagonal matrix of size $p \times 2$ that encodes the specifics of the max-pooling, namely in our case $D = \mathbf{diag}(\mathbf{1}_{p_1}, \mathbf{1}_{p_2})$.

Extension to matrix inputs. Equipped with a divergence function that works on vector inputs, we can readily extend it to matrix inputs with the convention that the divergence is summed across columns (data points). Specifically, if $X = [x_1, \ldots, x_m] \in \mathbb{R}^{k \times m}$, we define ϕ by $Z = \phi(X) = [z_1, \ldots, z_m] \in \mathbb{R}^{h \times m}$ as acting column-wise. We have

$$\phi(X) := [\phi(x_1), \ldots, \phi(x_m)] = \arg\min_Z \ \mathcal{D}_\phi(X, Z),$$

where, with some minor abuse of notation, we define a matrix version of the divergence, as follows:

$$\mathcal{D}_\phi([x_1, \ldots, x_m], [z_1, \ldots, z_m]) = \sum_{i=1}^m \mathcal{D}_\phi(x_i, z_i).$$

5. Lifted Framework

5.1. Lifted neural networks

Assume that the BCR or JCR condition is satisfied for each layer of our network and use the short-hand notation $D_l = D_{\phi_l}$ for the corresponding divergences. Condition (2) is then written as

$$X_{l+1} \in \arg\min_{X \in \mathcal{X}} \ D_l(X, W_l X_l + b_l \mathbf{1}^T), \ \ l = 0, \ldots, L.$$

The lifted model consists in replacing the constraints (2) with penalties in the training problem. Specifically, the lifted network training problem takes the form

$$\begin{aligned} \min_{(W_l, b_l), (X_l)} \ & \mathcal{L}(Y, W_L X_L + b_L \mathbf{1}^T) + \sum_{l=0}^L \pi_l(W_l) \qquad (10) \\ & + \sum_{l=0}^{L-1} \lambda_{l+1} D_l(W_l X_l + b_l \mathbf{1}^T, X_{l+1}) \\ & \text{s.t. } X_0 = X, \ \ X_l \geq 0, \ l = 1, \ldots, L-1 \end{aligned}$$

with $\lambda_1, \ldots, \lambda_{L+1}$ given positive hyper-parameters. As with the model introduced in section 3, the lifted model enjoys the same *parallel and convex* structure outlined earlier. In particular, it is convex in X-variables for fixed W-variables. If we use a weaker bi-convex representation (using a bi-convex divergence instead of a jointly convex one), then convexity with respect to X-variables is lost. However, the model is still convex in X_l for a given l when all the other variables are fixed; this still allows for block-coordinate descent algorithms to be used.

As a specific example, consider a multi-class classification problem where all the layers involve ReLUs except for the last. The last layer aims at producing a probability distribution to be compared against training labels via a cross entropy loss function. The training problem writes

$$\begin{aligned}\min_{(W_l,b_l),(X_l)} \; & -\mathbf{Tr}\, Y^T \log s(W_L X_L + b_L \mathbf{1}^T) + \sum_{l=0}^{L} \rho_l \|W_l\|_F^2 \\ & + \sum_{l=0}^{L-1} \lambda_{l+1} \|X_{l+1} - W_l X_l - b_l \mathbf{1}^T\|_F^2 \\ \text{s.t. } & X_0 = X, \;\; X_l \geq 0, \;\; l = 1, \ldots, L-1\end{aligned} \qquad (11)$$

where the equality constraint on X_{L+1} enforces that its columns are probability distributions. Here, $s(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^n$ is the softmax function. We can always rescale the variables so that in fact the number of additional hyper-parameters λ_l, $l = 1, \ldots, L-1$, is reduced to just one.

5.2. Lifted prediction rule

In our model, the prediction rule will be different from that of a standard neural network, but it is based on the same principle. In a standard network, the prediction rule can be obtained by solving the problem

$$\hat{y}(x) = \min_y \; \mathcal{L}(y, x_{L+1}) \; : \; (2), \;\; x_0 = x,$$

where the weights are now *fixed*, and $y \in \mathbb{R}^p$ is a *variable*. Of course, provided the loss is zero whenever its two arguments coincide, the above trivially reduces to the standard prediction rule: $\hat{y}(x) = x_{L+1}$, where x_{L+1} is obtained via the recursion (2).

In a lifted framework, we use the same principle: solve the training problem (in our case, (10)), using the test point as input, fixing the weights, and letting the predicted output values be variables. In other words, the prediction rule for a given test point x in lifted networks is based on solving the problem

$$\begin{aligned}\hat{y} = \arg\min_{y,(x_l)} \; & \mathcal{L}(y, W_L x_L + b_L) \\ & + \sum_{l=0}^{L-1} \lambda_{l+1} D_l(W_l x_l + b_l, x_{l+1}) \\ \text{s.t. } & x_0 = x.\end{aligned} \qquad (12)$$

The above prediction rule is a simple convex problem in the variables y and x_l, $l = 1, \ldots, L$. In our experiments, we have found that applying the standard feedforward rule of traditional networks is often enough.

6. Block-Coordinate Descent Algorithm

In this section, we outline a block-coordinate descent approach to solve the training problem (10).

6.1. Updating (W, b)-variables

For fixed X-variables, the problem of updating the W-variables, *i.e.* the weighting matrices $(W_l, b_l)_{l=0}^L$, is parallelizable across both data points and layers. The sub-problem involving updating the weights at a given layer $l = 0, \ldots, L$ takes the form

$$(W_l^+, b_l^+) = \arg\min_{W,b} \lambda_{l+1} D_l(W X_l + b\mathbf{1}^T, X_{l+1}) + \pi_l(W).$$

The above is a convex problem, which can be solved via standard machine learning libraries. Since the divergences are sums across columns (data points), the above problem is indeed parallelizable across data points.

For example, when the activation function at layer l is a ReLU, and the penalty π_l is a squared Frobenius norm, the above problem reads

$$(W_l^+, b_l^+) = \arg\min_{W,b} \lambda_{l+1} \|W X_l + b\mathbf{1}^T - X_{l+1}\|_F^2 + \rho_l \|W\|_F^2$$

which is a standard (matrix) ridge regression problem. Modern sketching techniques for high-dimensional least-squares can be employed, see for example (Woodruff et al., 2014; Pilanci & Wainwright, 2016).

6.2. Updating X-variables

In this step we minimize over the matrices $(X_l)_{l=1}^{L+1}$. The sub-problem reads exactly as (10), with now the (W, b)-variables fixed. By construction of divergences, the problem is decomposable across data points. When JCR conditions hold, the joint convexity of each JC-divergence function allows us update all the X-variables at once, by solving a convex problem. Otherwise, the update must be done cyclically over each layer, in a block-coordinate fashion.

For $l = 1, \ldots, L$, the sub-problem involving X_l, with all the other X-variables X_j, $j \neq l$ fixed, takes the form

$$\begin{aligned}X_l^+ = \arg\min_Z \; & \lambda_{l+1} D_l(W_l Z + b_l \mathbf{1}^T, X_{l+1}) \\ & + \lambda_l D_{l-1}(Z, X_{l-1}^0)\end{aligned} \qquad (13)$$

where $X_{l-1}^0 := W_{l-1} X_{l-1} + b_{l-1} \mathbf{1}^T$. By construction, the above is a convex problem, and is again parallelizable across data points.

Let us detail this approach in the case when the layers $l, l+1$ are both activated by ReLUs. The sub-problem above becomes

$$\begin{aligned}X_l^+ = \arg\min_{Z \geq 0} \; & \lambda_{l+1} \|X_{l+1} - W_l Z - b_l \mathbf{1}^T\|_F^2 + \\ & \lambda_l \|Z - W_{l-1} X_{l-1} - b_{l-1} \mathbf{1}^T\|_F^2\end{aligned}$$

The above is a (matrix) non-negative least-squares, for which many modern methods are available, see (Kim et al., 2007; 2014) and references therein. As before, the problem above is fully parallelizable across data points (columns), where each data point gives rise to a standard (vector) non-linear least-squares. Note that the cost of updating all columns can be reduced by taking into account that all column's updates share the same coefficient matrix W_l.

The case of updating the last matrix X_{L+1} is different, as it involves the output and the loss function $\mathcal{L}$. The update rule for X_{L+1} is indeed

$$X_{L+1}^{+} = \arg\min_{Z} \ \mathcal{L}(Y, Z) + \lambda_{L+1} D_L(X_L^0, Z), \qquad (14)$$

where $X_L^0 := W_L X_L + b_L \mathbf{1}^T$. Again the above is parallelizable across data points.

In the case when the loss function $\mathcal{L}$ is a squared Frobenius norm, and with a ReLU activation, the update rule (14) takes the form

$$X_{L+1} = \arg\min_{Z \geq 0} \ \|Z - Y\|_F^2 + \lambda_L \|Z - X_L^0\|_F^2,$$

which can be solved analytically:

$$X_{L+1}^{+} = \max\left(0, \frac{1}{1+\lambda_{L+1}} Y + \frac{\lambda_{L+1}}{1+\lambda_{L+1}} X_L^0\right).$$

In the case when the loss function is cross-entropy, and the last layer generates a probability distribution via the probability simplex projection, the above takes the form

$$\begin{aligned} X_{L+1} = \arg\min_{Z} \ & -\mathbf{Tr}\, Y^T \log Z + \lambda_{L+1} \|Z - X_L^0\|_F^2 \\ & Z \geq 0, \quad Z^T \mathbf{1} = \mathbf{1} \end{aligned} \qquad (15)$$

where we use the notation $\log$ in a component-wise fashion. The above can be solved as a set of parallel bisection problems. See Appendix A.

7. Numerical Experiments

Although lifted models in their own right can be used for supervised learning tasks, their main success so far has been using them to initialize traditional networks. In this section, we examine this and see if the lifted models can generate good initial guesses for standard networks.

7.1. MNIST

The model described in this paper was compared against a traditional neural network with equivalent architectures on the MNIST dataset (LeCun & Cortes, 2010). 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 as seen in (11). 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 of the same architecture to compare their performance, both in classification and convergence during training. For all models, ReLU activations were used. ℓ_2 regularization was used for all layers and the regularization parameters $\rho = 10^{-3}$ were held constant throughout all training procedures. The λ parameters for the lifted model were selected using Bayesian Optimization. The lifted model was trained using the block-coordinate descent scheme outlined in Section 6. The standard feedforward networks were trained in Tensorflow using a constant learning rate; reasons for this are highlighted in (Wilson et al., 2017). Table 1 summarizes the accuracy rates for the different architectures for 2 different learning rates. Figure 1 illustrates the test set accuracy versus number of epochs for two different architectures.

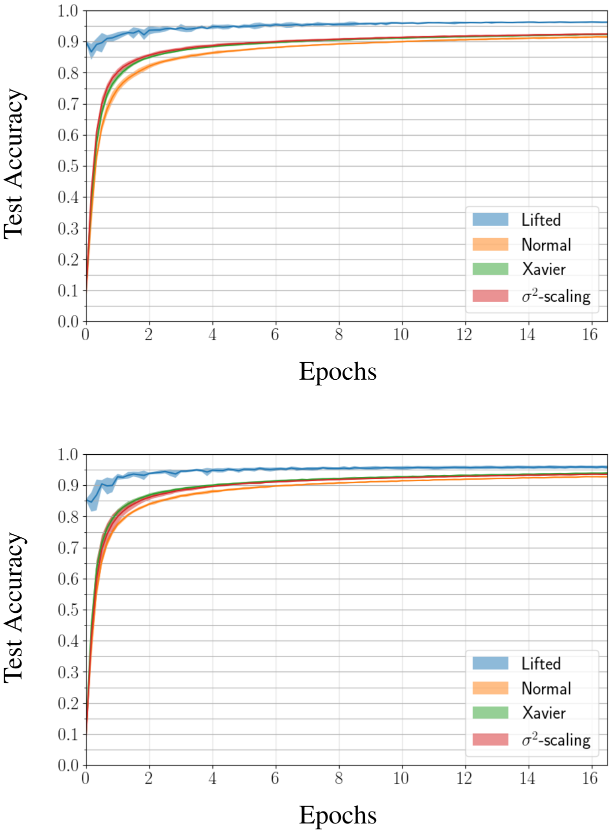

Figure 1. Plot of test accuracy vs number of training epochs on a held-out validation set during training for two different architectures. The shaded area on the plots indicated uncertainty to 2 standard deviations across 5 different experiments. The batch size was fixed at 100 and the learning rate was $\eta = 1\mathrm{e}{-5}$. **Top:** One layer neural network with 300 hidden units and ReLU activation. **Bottom:** Neural network composed of 3 ReLU layers with 500, 200, and 100 hidden units respectively.

Learning rate η =1e-5

Architecture	Our Model	NN[Normal]	NN[Xavier]	NN [σ-scale]	NN [Lifted]
300	0.898 ± 0.005	0.915 ± 0.004	0.9230 ± 0.005	0.924 ± 0.003	$\mathbf{0.962 \pm 0.003}$
$300 - 100$	0.875 ± 0.005	0.919 ± 0.003	0.932 ± 0.003	0.931 ± 0.003	$\mathbf{0.969 \pm 0.004}$
$500 - 150$	0.865 ± 0.005	0.927 ± 0.003	0.936 ± 0.004	0.935 ± 0.005	$\mathbf{0.970 \pm 0.005}$
$500 - 200 - 100$	0.853 ± 0.003	0.927 ± 0.001	0.939 ± 0.005	0.935 ± 0.003	$\mathbf{0.958 \pm 0.008}$
$400 - 200 - 100 - 50$	0.770 ± 0.015	0.919 ± 0.005	$\mathbf{0.938 \pm 0.003}$	0.936 ± 0.006	0.919 ± 0.030

Learning rate η =1e-6

Architecture	Our Model	NN[Normal]	NN[Xavier]	NN [σ^2-scale]	NN [Lifted]
300	0.898 ± 0.005	0.800 ± 0.011	0.836 ± 0.008	0.844 ± 0.011	$\mathbf{0.875 \pm 0.022}$
$300 - 100$	0.875 ± 0.005	0.792 ± 0.013	0.838 ± 0.007	0.842 ± 0.009	$\mathbf{0.899 \pm 0.021}$
$500 - 150$	0.865 ± 0.005	0.824 ± 0.007	0.850 ± 0.004	0.858 ± 0.002	$\mathbf{0.890 \pm 0.018}$
$500 - 200 - 100$	0.853 ± 0.003	0.821 ± 0.017	0.857 ± 0.007	0.848 ± 0.011	$\mathbf{0.926 \pm 0.053}$
$400 - 200 - 100 - 50$	0.770 ± 0.015	0.751 ± 0.045	0.838 ± 0.017	0.815 ± 0.020	$\mathbf{0.959 \pm 0.003}$

Table 1. Accuracy rate on the test set using different networks with the best result in boldface. The architectures indicate the number of hidden layers and the number of hidden units per layer. NN[x] indicates a standard neural network initialized with method x: *Normal* for normally distributed intialization of all weight variables with $\mu = 0$ and $\sigma^2 = 0.1$, *Xavier* for initialization highlighted in (Glorot & Bengio, 2010), σ^2*-scale* for variance scaling initialization and *Lifted* for initializing with the weights and biases learned from a lifted NN. All bias variables were initialized to 0.1 except for the Lifted case in which the bias vectors are optimized during pretraining. The neural networks were trained for 17 epochs using mini-batch gradient descent in Tensorflow (Abadi et al., 2015). The lifted model achieves test accuracy as high as 90 % on MNIST.

Remark 1. *Although our model does not perform as well as the other models on this task, using it as initialization results in increased accuracy for almost all network architectures.*

In particular, in Figure 1 we see that with our initialization, the test accuracy both converges more quickly and to higher values compared with the other initializations: in fact, across all experiments the lifted initlization starts within 90% of its final accuracy. This seems to indicate that the lifted model we train on is a close approximation to a standard feed-forward network and our weights learned are already near optimal for these networks. Although after a few passes of the dataset the other models converge, we usually observed a constant gap between the test set accuracy using our initialization versus the others.

8. Conclusion

In this work we have proposed a novel model for supervised learning. The key idea behind our method is replacing non-smooth activation functions by smooth penalties in the training problem; we have shown how to do this for general monotonic activation functions. This modifies the multi-layer neural networks optimization problem to a similar problem which we called a lifted neural network. We applied this technique to build a model which we later use as initialization on feedforward neural networks with ReLU activations. Experimental results have shown that the weights of our trained model serve as a good initialization for the parameters of classical neural networks, outperforming neural networks with both random and structured initialization.

9. Future Work

Although lifted nets give good results when used as weight initialization for MNIST, they have not extensively been tested on other well known datasets such as CIFAR-10 or other non-image based data sets. The simplest extension of this work will be to apply lifted nets to these different data sets and to different learning tasks such as regression. The lifted framework also easily allows for several extensions and variants that would be very difficult to consider in a standard formulation. This includes handling uncertainty in the data (matrix uncertainty) using principles of robust optimization, optimizing over scale parameters in activation functions, such as the α-parameter in leaky-ReLUs, and adding unitary constraints on the W variables. Speedup in a distributed setting is also a point of interest. Additionally, the lifted model can easily be adapted for both convolutional and recurrent neural network architectures.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Berrada, Leonard, Zisserman, Andrew, and Kumar, M. Pawan. Trusting svm for piecewise linear cnns. *CoRR*, abs/1611.02185, 2016.

Bubeck, Sébastien. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 2015. doi: 10.1561/2200000050. URL http://dx.doi.org/10.1561/2200000050.

Carreira-Perpinan, Miguel and Wang, Weiran. Distributed optimization of deeply nested systems. In Kaski, Samuel and Corander, Jukka (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pp. 10–19, Reykjavik, Iceland, 22–25 Apr 2014. PMLR. URL http://proceedings.mlr.press/v33/carreira-perpinan14.html.

Duchi, John C., Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2011. URL http://dl.acm.org/citation.cfm?id=2021068.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

Kim, Dongmin, Sra, Suvrit, and Dhillon, Inderjit S. Fast Newton-type methods for the least squares nonnegative matrix approximation problem. In *Proceedings of the 2007 SIAM international conference on data mining*, pp. 343–354. SIAM, 2007.

Kim, Jingu, He, Yunlong, and Park, Haesun. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations (ICLR)*, 2015.

LeCun, Yann and Cortes, Corinna. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

LeCun, Yann, Bottou, Léon, Orr, Genevieve B., and Müller, Klaus-Robert. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65311-2. URL http://dl.acm.org/citation.cfm?id=645754.668382.

Maclin, Richard and Shavlik, Jude W. 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. ISBN 1-55860-363-8, 978-1-558-60363-9. URL http://dl.acm.org/citation.cfm?id=1625855.1625924.

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

Seuret, Mathias, Alberti, Michele, Ingold, Rolf, and Liwicki, Marcus. PCA-initialized deep neural networks applied to document image analysis. *CoRR*, abs/1702.00177, 2017. URL http://arxiv.org/abs/1702.00177.

Sutskever, Ilya, Martens, James, Dahl, George, and Hinton, Geoffrey. 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.

Taylor, Gavin, Burmeister, Ryan, Xu, Zheng, Singh, Bharat, Patel, Ankit, and Goldstein, Tom. Training neural networks without gradients: A scalable admm approach. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 2722–2731. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045677.

Wilson, Ashia C., Roelofs, Rebecca, Stern, Mitchell, Srebro, Nati, and Recht, Benjamin. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 4151–4161, 2017.

Woodruff, David P et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

Appendix A. Solving for the last layer with cross entropy loss

In this section, we consider problem (15), which is of the form

$$\min_Z -\mathbf{Tr}\, Y^T \log Z + \lambda \|Z - X^0\|_F^2 \;:\; Z^T\mathbf{1} = \mathbf{1}, \;\; Z \geq 0, \tag{16}$$

where we use the notation $\log$ in a component-wise fashion, and $X^0 \in \mathbb{R}^{p\times m}$ and $Y \in \{0,1\}^{p\times m}$, $Y^T\mathbf{1} = \mathbf{1}$, $\lambda > 0$ are given. The above can be easily solved by dual matrix bisection. Indeed, the problem can be decomposed across columns of Y (that is, across data points). The problem for a single column has the following form:

$$p^* := \min_z - \sum_{i=1}^p y_i \log z_i + \lambda \|z - x^0\|_2^2 \;:\; z \geq 0, \;\; z^T\mathbf{1} = 1,$$

where vectors $y \in \{0,1\}^p$, $y^T\mathbf{1} = 1$ and $x^0 \in \mathbb{R}^p$ are given. Dualizing the equality constraint, we obtain a Lagrangian of the form

$$\mathcal{L}(z,\nu) = 2\nu + \sum_{i=1}^p \left(z_i^2 - \frac{y_i}{\lambda}\log z_i - 2z_i(\nu + x_i^0)\right),$$

where ν is a (scalar) dual variable. At the optimum z^*, we have

$$\forall\, i \;:\; 0 = \frac{1}{2}\frac{\partial \mathcal{L}(z,\nu)}{\partial z_i}(z^*,\nu) = z_i^* - \frac{y_i}{2\lambda z_i^*} - (\nu + x_i^0),$$

leading to the unique non-negative solution

$$z_i^* = \frac{x_i^0 + \nu}{2} + \sqrt{\left(\frac{x_i^0+\nu}{2}\right)^2 + \frac{y_i}{2\lambda}}, \;\; i = 1, \ldots, p,$$

where the dual variable ν is such that $\mathbf{1}^T z^* = 1$. We can locate such a value ν by simple bisection.

The bisection scheme requires initial bounds on ν. For the upper bound, we note that the property $z^* \leq \mathbf{1}$, together with the above optimality condition, implies

$$\nu \leq 1 - \max_{1\leq i\leq p} \left(x_i^0 + \frac{y_i}{2\lambda}\right).$$

For the lower bound, let us first define $\mathcal{I} := \{i \;:\; y_i \neq 0\}$, $k = |\mathcal{I}| \leq p$. At optimum, we have

$$\forall\, i \in \mathcal{I} \;:\; -\log z_i^* \leq - \sum_{j\in\mathcal{I}} y_j \log z_j^* \leq p^* \leq \theta,$$

$$\theta := -\sum_{i\in\mathcal{I}} y_i \log z_i^0 + \lambda \|z^0 - x^0\|_2^2$$

where $z^0 \in \mathbb{R}^p$ is any primal feasible point, for example $z_i^0 = 1/k$ if $i \in \mathcal{I}$, 0 otherwise. We obtain

$$\forall\, i \in \mathcal{I} \;:\; z_i^* \geq z_{\min} := e^{-\theta}.$$

The optimality conditions imply

$$0 = \frac{1}{2}\sum_i \frac{\partial \mathcal{L}(z^*,\nu)}{\partial z_i} = -\frac{1}{2\lambda}\sum_{i\in\mathcal{I}} \frac{y_i}{z_i^*} + 1 - \mathbf{1}^T x^0 - p\nu$$

and therefore:

$$p\nu = 1 - \mathbf{1}^T x^0 - \frac{1}{2\lambda}\sum_{i\in\mathcal{I}} \frac{y_i}{z_i^*} \geq 1 - \mathbf{1}^T x^0 - \frac{\mathbf{1}^T y}{2\lambda} e^{\theta}.$$

To conclude, we have

$$\frac{1}{p}\left(1 - \mathbf{1}^T x^0 - \frac{\mathbf{1}^T y}{2\lambda}e^{\theta}\right) =: \underline{\nu} \leq \nu \leq \overline{\nu} := 1 - \max_{1\leq i\leq p}\left(x_i^0 + \frac{y_i}{2\lambda}\right).$$

To solve the original (matrix) problem (16), we can process all the columns in parallel (matrix) fashion, updating a *vector* $\nu \in \mathbb{R}^m$.