Fenchel Lifted Networks: A Lagrange Relaxation of Neural Network Training

Fangda Gu* **Armin Askari*** **Laurent El Ghaoui**
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
{gfd18, aaskari, elghaoui}@berkeley.edu

Abstract

Despite the recent successes of deep neural networks, the corresponding training problem remains highly non-convex and difficult to optimize. Classes of models have been proposed that introduce greater structure to the objective function at the cost of lifting the dimension of the problem. However, these lifted methods sometimes perform poorly compared to traditional neural networks. In this paper, we introduce a new class of lifted models, Fenchel lifted networks, that enjoy the same benefits as previous lifted models, without suffering a degradation in performance over classical networks. Our model represents activation functions as equivalent biconvex constraints and uses Lagrange Multipliers to arrive at a rigorous lower bound of the traditional neural network training problem. This model is efficiently trained using block-coordinate descent and is parallelizable across data points and/or layers. We compare our model against standard fully connected and convolutional networks and show that we are able to match or beat their performance. [1]

1 Introduction

Deep neural networks (DNNs) have become the preferred model for supervised learning tasks after their success in various fields of research. However, due to their highly non-convex nature, DNNs pose a difficult problem during training time; the optimization landscape consists of many saddle points and local minima which make the trained model generalize poorly (Chaudhari et al., 2016; Dauphin et al., 2014). This has motivated regularization schemes such as weight decay (Krogh and Hertz, 1992), batch normalization (Ioffe and Szegedy, 2015), and dropout (Srivastava et al., 2014) so that the solutions generalize better to the test data.

In spite of this, backprop used along with stochastic gradient descent (SGD) or similar variants like Adam (Kingma and Ba, 2015) suffer from a variety of problems. One of the most notable problems is the vanishing gradient problem which slows down gradient-based methods during training time. Several approaches have been proposed to deal with the problem; for example, the introduction of rectified linear units (ReLU). However, the problem persists. For a discussion on the limitations of backprop and SGD, we direct the reader to Section 2.1 of Taylor et al. (2016).

One approach to deal with this problem is to introduce auxiliary variables that increase the dimension of the problem. In doing so, the training problem decomposes into multiple, local sub-problems which can be solved efficiently without using SGD or Adam; in particular, the methods of choice have been block coordinate descent (BCD) (Askari et al., 2018; Lau et al., 2018; Zhang and Brand, 2017; Carreira-Perpinan and Wang, 2014) and the alternating direction method of multipliers (ADMM) (Taylor et al., 2016; Zhang et al., 2016). By lifting the dimension of the problem, these models avoid many of the problems DNNs face during training time. They also offer new avenues towards interpretability and robustness of networks by allowing for the penalization of the additional variables.

While these methods, which we refer to as "lifted" models for the remainder of the paper, offer an alter-

[1] A python implementation of our model can be found at https://github.com/beeperman/Fenchel_Lifted_Networks

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

native to the original problem with some added benefits, they have their limitations. Most notably, traditional DNNs are still able to outperform these methods in spite of the difficult optimization landscape. As well, most of the methods are unable to operate in an online manner or adapt to continually changing data sets which is prevalent in most reinforcement learning settings (Sutton and Barto, 1998). Finally, by introducing auxiliary variables, the dimensionality of the problem greatly increases, making these methods very difficult to train with limited computational resources.

1.1 Paper contribution

To address the problems listed above, we propose Fenchel lifted networks, a biconvex formulation for deep learning based on Fenchel's duality theorem that can be optimized using BCD. We show that our method is a rigorous *lower bound* for the learning problem and admits a natural batching scheme to adapt to changing data sets and settings with limited computational power. We compare our method against other lifted models and against traditional fully connected and convolutional neural networks. We show that we are able to outperform the former and that we can compete with or even outperform the latter.

Paper outline. In Section 2, we give a brief overview of related works on lifted models. In Section 3 we introduce the notation for the remainder of the paper. Section 4 introduces Fenchel lifted networks, their variants and discusses how to train these models using BCD. Section 5 compares the proposed method against fully connected and convolutional networks on MNIST and CIFAR-10.

2 Related Work

Lifted methods Related works that lift the dimension of the training problem are primarily optimized using BCD or ADMM. These methods have experienced recent success due to their ability to exploit the structure of the problem by first converting the constrained optimization problem into an unconstrained one and then solving the resulting sub-problems in parallel. They do this by relaxing the network constraints and introducing penalties into the objective function. The two main ways of introducing penalties into the objective function are either using quadratic penalties (Sutskever et al., 2013; Taylor et al., 2016; Lau et al., 2018) or using equivalent representations of the activation functions (Askari et al., 2018; Zhang and Brand, 2017).

As a result, these formulations have many advantages over the traditional training problem, giving superior performance in some specific network structures (Carreira-Perpinan and Wang, 2014; Zhang and Brand, 2017). These methods also enjoy great potential for parallelization as shown by Taylor et al. (2016); the authors parallelize training over different cores and show a linear scaling between reduction in training time and the number of cores used. However, there has been little evidence showing that these methods can compete with traditional DNNs which shadows the nice structure these formulations bring about.

An early example of auxiliary variables being introduced into the training problem is the method of auxiliary coordinates (MAC) by Carreira-Perpinan and Wang (2014) which uses quadratic penalties to enforce network constraints. They test their method on auto encoders and show that their method is able to outperform SGD. Followup work by Carreira-Perpinán and Alizadeh (2016); Taylor et al. (2016) demonstrate the huge potential for parallelizing these methods. Lau et al. (2018) gives some convergence guarantee on a modified problem.

Another class of models that lift the dimension of the problem do so by representing activation functions in equivalent formulations. Negiar et al. (2017); Askari et al. (2018); Zhang and Brand (2017); Li et al. (2019) explore the structure of activation functions and use arg min maps to represent activation functions. In particular, Askari et al. (2018) show how a strictly monotone activation function can be seen as the arg min of a specific optimization problem. Just as with quadratic penalties, this formulation of the problem still performs poorly compared to traditional neural networks. In an independent and concurrent work by Li et al. (2019), the authors arrive at a lifted formulation of the training problem via the use of proximal operators. While the approach in aforementioned work appears unrelated to the work presented here, they are in fact deeply related (for a more complete discussion, see Appendix D).

3 Background and Notation

Feedforward neural networks. We are given an input data matrix of m data points $X = [x_1, x_2, ..., x_m] \in \mathbb{R}^{n\times m}$ and a response matrix $Y \in \mathbb{R}^{p\times m}$. We consider the supervised learning problem involving a neural network with $L \geq 1$ hidden layers. The neural network produces a prediction $\hat{Y} \in \mathbb{R}^{p\times m}$ with the feed forward recursion $\hat{Y} = W_L X_L + b_L \mathbf{1}^\top$ given below

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}^\top), \quad l = 0, \ldots, L-1. \qquad (1)$$

where $\phi_l, l = 0, \ldots, L$ are the activation functions that act column-wise on a matrix input, $\mathbf{1} \in \mathbb{R}^m$ is a vector of ones, and $W_l \in \mathbb{R}^{p_{l+1}\times p_l}$ and $b_l \in \mathbb{R}^{p_{l+1}}$ are the weight matrices and bias vectors respectively. Here p_l is the number of output values for a single data point (i.e., hidden nodes) at layer l with $p_0 = n$ and $p_{L+1} = p$. Without loss of generality, we can remove $b_l\mathbf{1}^\top$ by adding an extra column to W_l and a row of ones to X_l. Then (1) simplifies to

$$X_{l+1} = \phi_l(W_l X_l), \quad l = 0, \ldots, L-1. \tag{2}$$

In the case of fully connected networks, ϕ_l is typically sigmoidal activation functions or ReLUs. In the case of Convolutional Neural Networks (CNNs), the recursion can accommodate convolutions and pooling operations in conjunction with an activation. For classification tasks, we typically apply a softmax function after applying an affine transformation to X_L.

The initial value for the recursion is $X_0 = X$ and $X_l \in \mathbb{R}^{p_l\times m}$, $l = 0, \ldots, L$. We refer to the collections $(W_l)_{l=0}^L$ and $(X_l)_{l=1}^L$ as the W and X-variables respectively.

The weights are obtained by solving the following constrained optimization problem

$$\begin{aligned} \min_{(W_l)_{l=0}^L,(X_l)_{l=1}^L} \quad & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ \text{s.t.} \quad & X_{l+1} = \phi_l(W_l X_l),\ l = 0, \ldots, L-1 \\ & X_0 = X \end{aligned} \tag{3}$$

Here, $\mathcal{L}$ is a loss function, $\rho \in \mathbb{R}_+^{L+1}$ is a hyperparameter vector, and π_l's are penalty functions used for regularizing weights, controlling network structures, etc. In (3), optimizing over the X-variables is trivial; we simply apply the recursion (2) and solve the resulting unconstrained problem using SGD or Adam. After optimizing over the weights and biases, we obtain a prediction $\hat{Y}$ for the test data X by passing X through the recursion (2) one layer at a time.

Our model. We develop a family of models where we approximate the recursion constraints (2) via penalties. We use the arg min maps from Askari et al. (2018) to create a biconvex formulation that can be trained efficiently using BCD and show that our model is a lower bound of (3). Furthermore, we show how our method can naturally be batched to ease computational requirements and improve the performance.

4 Fenchel lifted networks

In this section, we introduce Fenchel lifted networks. We begin by showing that for a certain class of activation functions, we can equivalently represent them as biconvex constraints. We then dualize these constraints and construct a lower bound for the original training problem. We show how our lower bound can naturally be batched and how it can be trained efficiently using BCD.

4.1 Activations as bi-convex constraints

In this section, we show how to convert the equality constraints of (3) into inequalities which we dualize to arrive at a *relaxation* (lower bound) of the problem. In particular, this lower bound is biconvex in the W-variables and X-variables. We make the following assumption on the activation functions ϕ_l.

> **BC Condition** The activation function $\phi : \mathbb{R}^p \to \mathbb{R}^q$ satisfies the BC condition if there exists a *biconvex* function $B_\phi : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$, such that
>
> $$v = \phi(u) \Longleftrightarrow B_\phi(v, u) \le 0.$$

We now state and prove a result that is at the crux of Fenchel lifted networks.

Theorem 1. *Assume $\phi : \mathbb{R} \to \mathbb{R}$ is continuous, strictly monotone and that $0 \in$ range(ϕ) or $0 \in$ domain(ϕ). Then ϕ satisfies the BC condition.*

Proof. Without loss of generality, ϕ is strictly increasing. Thus it is invertible and there exists ϕ^{-1} such that $u = \phi^{-1}(v)$ for $v \in \text{range}(\phi)$ which implies $v = \phi(u)$. Now, define $F : \mathbb{R}^p \to \mathbb{R}$ as

$$F(v) := \int_z^v \phi^{-1}(\xi)\, d\xi$$

where $z \in \text{range}(\phi)$ and is either 0 or satisfies $\phi^{-1}(z) = 0$. Then we have

$$\begin{aligned} F^*(u) &= \int_{\phi^{-1}(z)}^u \phi(\eta)\, d\eta \\ B(v, u) &= F(v) + F^*(u) - uv \end{aligned} \tag{4}$$

where F^* is the Fenchel conjugate of F. By the Fenchel-Young inequality, $B(v, u) \ge 0$ with equality if and only if

$$v^* = \arg\max_v \; uv - F(v) \;:\; v \in \text{range}(\phi)$$

By construction, $v^* = \phi(u)$. Note furthermore since ϕ is continuous and strictly increasing, so is ϕ^{-1} on its domain, and thus F, F^* are convex. It follows that $B(v, u)$ is a biconvex function of (u, v).

We simply need to prove that $F^*(u)$ above is indeed the Fenchel conjugate of F. By definition of the Fenchel conjugate we have that

$$F^*(u) = \max_v \; uv - F(v) \;:\; v \in \text{range}(\phi)$$

It is easy to see that $v^* = \phi(u)$. Thus

$$\begin{aligned} F^*(u) &= u\phi(u) - F(\phi(u)) \\ &= u\phi(u) - \int_z^{\phi(u)} \phi^{-1}(\xi)\, d\xi \\ &= \int_z^{\phi(u)} \xi \frac{d}{d\xi}\phi^{-1}(\xi)\, d\xi \\ &= \int_{\phi^{-1}(z)}^{u} \phi(\eta)\, d\eta \end{aligned}$$

where the third equality is a consequence of integration by parts, and the fourth equality we make the subsitution $\eta = \phi^{-1}(\xi)$ □

Note that Theorem 1 implies that activation functions such as sigmoid and tanh can be equivalently written as a biconvex constraint. Although the ReLU is not strictly monotone, we can simply restrict the inverse to the domain $\mathbb{R}_+$; specifically, for $\phi(x) = \max(0, x)$ define

$$\phi^{-1}(z) = \begin{cases} +\infty & \text{if } z < 0, \\ z & \text{if } z \geq 0, \end{cases}$$

Then, we can rewrite the ReLU function as the equivalent set of biconvex constraint

$$v = \max(0, u) \iff \begin{cases} \frac{1}{2}v^2 + \frac{1}{2}u_+^2 - uv \leq 0 \\ v \geq 0 \end{cases}$$

where $u_+ = \max(0, u)$. This implies

$$B_\phi(v, u) = \begin{cases} \frac{1}{2}v^2 + \frac{1}{2}u_+^2 - uv & \text{if } v \geq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{5}$$

Despite the non-smoothness of u_+, for fixed u or fixed v, (5) belongs in C^1 – that is, it has continuous first derivative and can be optimized using first order methods. We can trivially extend the result of Theorem 1 for matrix inputs: for matrices $U, V \in \mathbb{R}^{p\times q}$, we have

$$B_\phi(V, U) = \sum_{i,j} B_\phi(V_{ij}, U_{ij}).$$

4.2 Lifted Fenchel model

Assuming the activation functions of (3) satisfy the hypothesis of Theorem 1, we can reformulate the learning problem equivalently as

$$\begin{aligned} \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \; & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ \text{s.t. } \; & B_l(X_{l+1}, W_l X_l) \leq 0, \; l = 0, \ldots, L-1 \\ & X_0 = X, \end{aligned} \tag{6}$$

where B_l is the short-hand notation of B_{ϕ_l}. We now dualize the inequality constraints and obtain the lower bound of the standard problem (3) via Lagrange relaxation

$$\begin{aligned} G(\lambda) := \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \; & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ & + \sum_{l=0}^{L-1} \lambda_l B_l(X_{l+1}, W_l X_l) \\ \text{s.t. } X_0 = X, & \end{aligned} \tag{7}$$

where $\lambda_l \geq 0$ are the Lagrange multipliers. The maximum lower bound can be achieved by solving the dual problem

$$p^* \geq d^* = \max_{\lambda \geq 0} G(\lambda) \tag{8}$$

where p^* is the optimal value of (3). Note if all our activation functions are ReLUs, we must also include the constraint $X_l \geq 0$ in the training problem as a consequence of (5). Although the new model introduces L new parameters (the Lagrange multipliers), we can show that using variable scaling we can reduce this to only *one* hyperparameter (for details, see Appendix A). The learning problem then becomes

$$\begin{aligned} G(\lambda) := \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \; & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ & + \lambda \sum_{l=0}^{L-1} B_l(X_{l+1}, W_l X_l) \\ \text{s.t. } X_0 = X. & \end{aligned} \tag{9}$$

In a regression setting where the data is generated by a one layer network, we are able to provide global convergence guarantees of the above model (for details, see Appendix B).

Comparison with other methods. For ReLU activations, $B(v, u)$ as in (5) differs from the penalty terms introduced in previous works. In Askari et al. (2018); Zhang and Brand (2017) they set $B(v, u) = \|v - u\|_2^2$ and in Taylor et al. (2016); Carreira-Perpinan and Wang (2014) they set $B(v, u) = \|v - u_+\|_2^2$. Note that $B(v, u)$ in the latter is not biconvex. While the $B(v, u)$ in the former is biconvex, it does not perform well at test time. Li et al. (2019) set $B(v, u)$ based on a proximal operator that is similar to the BC condition.

Convolutional model. Our model can naturally accommodate average pooling and convolution operations found in CNNs, since they are linear operations. We can rewrite $W_l X_l$ as $W_l * X_l$ where $*$ denotes the convolution operator and write $\text{Pool}(X)$ to denote the average pooling operator on X. Then, for example, the sequence Conv → Activation can be represented via the constraint

$$B_l(X_{l+1}, W_l * X_l) \leq 0, \quad (10)$$

while the sequence Pool → Conv → Activation can be represented as

$$B_l(X_{l+1}, W_l * \text{Pool}(X_l)) \leq 0. \quad (11)$$

Note that the pooling operation changes the dimension of the matrix.

4.3 Prediction rule.

In previous works that reinterpret activation functions as arg min maps (Askari et al., 2018; Zhang and Brand, 2017), the prediction at test time is defined as the solution to the optimization problem below

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

where x_0 is test data point, $\hat{y}$ is the predicted value, and x_l, $l = 1, \ldots, L$ are the intermediate representations we optimize over. Note if $\mathcal{L}$ is a mean squared error, applying the traditional feed-forward rule gives an optimal solution to (12). We find empirically that applying the standard feed-forward rule works well, even with a cross-entropy loss.

4.4 Batched model

The models discussed in the introduction usually require the entire data set to be loaded into memory which may be infeasible for very large data sets or for data sets that are continually changing. We can circumvent this issue by batching the model. By sequentially loading a part of the data set into memory and optimizing the network parameters, we are able to train the network with limited computational resources. Formally, the batched model is

$$\begin{aligned} \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \ & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ + \lambda & \sum_{l=0}^{L-1} B_l(W_l X_l, X_{l+1}) + \sum_{l=0}^{L} \gamma_l \|W_l - W_l^0\|_F^2 \\ & \text{s.t. } X_0 = X, \end{aligned} \quad (13)$$

where X_0 contains only a batch of data points instead of the complete data set. The additional term in the objective $\gamma_l \|W_l - W_l^0\|_F^2, \quad l = 0, \ldots, L$ is introduced to moderate the change of the W-variables between subsequent batches; here W_l^0 represents the optimal W variables from the previous batch and $\gamma \in \mathbb{R}_+^{L+1}$ is a hyperparameter vector. The X-variables are reinitialized each batch by feeding the new batch forward through the equivalent standard neural network.

4.5 Block-coordinate descent algorithm

The model (9) satisfies the following properties:

- For fixed W-variables, and fixed variables $(X_j)_{j \neq l}$, the problem is convex in X_l, and is decomposable across data points.
- For fixed X-variables, the problem is convex in the W-variables, and is decomposable across layers, and data points.

The non-batched and batched Fenchel lifted network are trained using block coordinate descent algorithms highlighted in Algorithms 1 and 2. By exploiting the biconvexity of the problem, we can alternate over updating the X-variables and W-variables to train the network.

Note Algorithm 2 is different from Algorithm 1 in three ways. First, re-initialization is required for the X-variables each time a new batch of data points are loaded. Second, the sub-problems for updating W-variables are different as shown in Section 4.5.2. Lastly, an additional parameter K is introduced to specify the number of training alternations for each batch. Typically, we set $K = 1$.

4.5.1 Updating X-variables

For fixed W-variables, the problem of updating X-variables can be solved by cyclically optimizing $X_l, \quad l = 1, \ldots, L$, with $(X_j)_{j \neq l}$ fixed. We initialize our X-variables by feeding forward through the equivalent neural network and update the X_l's backward from X_L to X_1 in the spirit of backpropagation.

We can derive the sub-problem for $X_l, \ l = 1, \ldots, L-1$ with $(X_j)_{j \neq l}$ fixed from (6). The sub-problem writes

$$X_l^+ = \arg\min_Z B_l(X_{l+1}, W_l Z) + B_{l-1}(Z, X_{l-1}^0) \quad (14)$$

where $X_{l-1}^0 := W_{l-1} X_{l-1}$. By construction, the sub-problem (14) is convex and parallelizable across data points. Note in particular when our activation is a ReLU, the objective function in (14) is in fact strongly convex and has a continuous first derivative.

Algorithm 1 Non-batched BCD Algorithm

Initialize $(W_l)_{l=0}^{L}$.
Initialize X_0 with input matrix X.
Initialize $X_1, \dots, X_L$ with neural network feed forward rule.
repeat
  $X_L \leftarrow \arg\min_Z \mathcal{L}(Y, W_L Z) + \lambda B_{L-1}(Z, X_{L-1}^0)$
  for $l = L-1, \dots, 1$ **do**
    $X_l \leftarrow \arg\min_Z B_l(X_{l+1}, W_l Z) + B_{l-1}(Z, X_{l-1}^0)$
  end for
  $W_L \leftarrow \arg\min_W \mathcal{L}(Y, W X_L) + \rho_l \pi_l(W)$
  for $l = L-1, \dots, 0$ **do**
    $W_l \leftarrow \arg\min_W \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W)$
  end for
until convergence

Algorithm 2 Batched BCD Algorithm

Initialize $(W_l)_{l=0}^{L}$.
repeat
  $(W_l^0)_{l=0}^{L} \leftarrow (W_l)_{l=0}^{L}$
  Re-initialize X_0 with a batch sampled from input matrix X.
  Re-initialize $X_1, \dots, X_L$ with neural network feed forward rule.
  for alternation$= 1, \dots, K$ **do**
    $X_L \leftarrow \arg\min_Z \mathcal{L}(Y, W_L Z) + \lambda B_{L-1}(Z, X_{L-1}^0)$
    for $l = L-1, \dots, 1$ **do**
      $X_l \leftarrow \arg\min_Z \lambda B_l(X_{l+1}, W_l Z) + \lambda B_{l-1}(Z, X_{l-1}^0)$
    end for
    $W_L \leftarrow \arg\min_W \mathcal{L}(Y, W X_L) + \rho_L \pi_L(W) + \gamma_l \|W - W_L^0\|_F^2$
    for $l = L-1, \dots, 0$ **do**
      $W_l \leftarrow \arg\min_W \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W) + \gamma_l \|W - W_l^0\|_F^2$
    end for
  end for
until convergence

For the last layer (i.e., $l = L$), the sub-problem derived from (6) writes differently

$$X_L^+ = \arg\min_Z \mathcal{L}(Y, W_L Z) + \lambda B_{L-1}(Z, X_{L-1}^0) \quad (15)$$

where $X_{L-1}^0 := W_{L-1} X_{L-1}$. For common losses such as mean square error (MSE) and cross-entropy, the subproblem is convex and parallelizable across data points. Specifically, when the loss is MSE and we use a ReLU activation at the layer before the output layer, (15) becomes

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

where $X_{L-1}^0 := W_{L-1} X_{L-1}$ and we use the fact that X_{L-1}^0 is a constant to equivalently replace B_{L-1} as in (5) by a squared Frobenius term. The sub-problem is a non-negative least squares for which specialized methods exist Kim et al. (2014).

For a cross-entropy loss and when the second-to-last layer is a ReLU activation, the sub-problem for the last layer takes the convex form

$$X_L^+ = \arg\min_{Z \geq 0} - \operatorname{Tr} Y^\top \log s(W_L Z) + \frac{\lambda}{2} ||Z - X_{L-1}^0||_F^2, \quad (16)$$

where $s(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ is the softmax function and log is the element-wise logarithm. Askari et al. (2018) show how to solve the above problem using bisection.

4.5.2 Updating W-variables

With fixed X-variables, the problem of updating the W-variables can be solved in parallel across layers.

Sub-problems for non-batched model. The problem of updating W_l at intermediate layers becomes

$$W_l = \arg\min_{W,} \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W). \quad (17)$$

Again, by construction, the sub-problem (17) is convex and parallelizable across data points. Also, since there is no coupling in the W-variables between layers, the sub-problem (17) is parallelizable across layers.

For the last layer, the sub-problem becomes

$$W_L = \arg\min_W \mathcal{L}(Y, W X_L) + \rho_L \pi_L(W). \quad (18)$$

Sub-problems for batched model. As shown in Section 4.4, the introduction of regularization terms between W and values from a previous batch require the sub-problems (17, 18) be modified. (17) now becomes

$$W_l = \arg\min_W \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W) + \gamma_l \|W - W_l^0\|_F^2, \quad (19)$$

while (18) becomes

$$W_L = \arg\min_W \mathcal{L}(Y, W X_L) + \rho_L \pi_L(W) + \gamma_L \|W - W_L^0\|_F^2. \quad (20)$$

Note that these sub-problems in the case of a ReLU activation are strongly convex and parallelizable across layers.

5 Numerical Experiments

In this section, we compare Fenchel lifted networks against other lifted models discussed in the introduction and against traditional neural networks. In particular, we compare our model against the models proposed by Taylor et al. (2016), Lau et al. (2018) and Askari et al. (2018) on MNIST. Then we compare Fenchel lifted networks against a fully connected neural network and LeNet-5 (LeCun et al., 1998) on MNIST. Finally, we compare Fenchel lifted networks against LeNet-5 on CIFAR-10. For a discussion on hyperparameters and how model paramters were selected, see Appendix Appendix C.

5.1 Fenchel lifted networks vs. lifted models

Here, we compare the non-batched Fenchel lifted network against the models proposed by Taylor et al. (2016)[2], Lau et al. (2018)[3] and Askari et al. (2018). The former model is trained using ADMM and the latter ones using the BCD algorithms proposed in the respective papers. In Figure 1, we compare these models on MNIST with a 784-300-10 architecture (inspired by LeCun et al. (1998)) using a mean square error (MSE) loss.

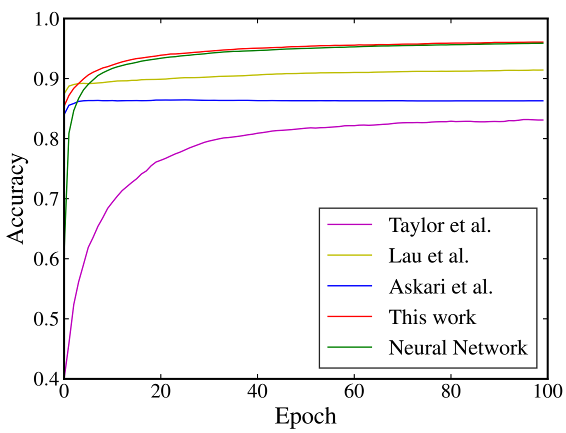

Figure 1: Test set performance of different lifted methods with a 784-300-10 network architecture on MNIST with a MSE loss. Final test set performances: **Taylor et al.** 0.834, **Lau et al.** 0.914, **Askari et al.** 0.863, **Neural Network** 0.957, **This work** 0.961.

After multiple iterations of hyperparameter search with little improvement over the base model, we chose to keep the hyperparameters for Taylor et al. (2016) and Lau et al. (2018) as given in the code. The hyperparameters for Askari et al. (2018) were tuned using cross validation on a hold-out set during training. Our model used these same parameters and cross validated the remaining hyperparameters. The neural network model was trained using SGD. The resulting curve of the neural network is smoothed in Figure 1 for visual clarity. From Figure 1 it is clear that Fenchel lifted networks vastly outperform other lifted models and achieve a test set accuracy on par with traditional networks.

5.2 Fenchel lifted networks vs. neural networks on MNIST

For the same 784-300-10 architecture as the previous section, we compare the batched Fenchel lifted networks against traditional neural networks trained using first order methods. We use a cross entropy loss in the final layer for both models. The hyperparameters for our model are tuned using cross validation. Figure 2 shows the results.

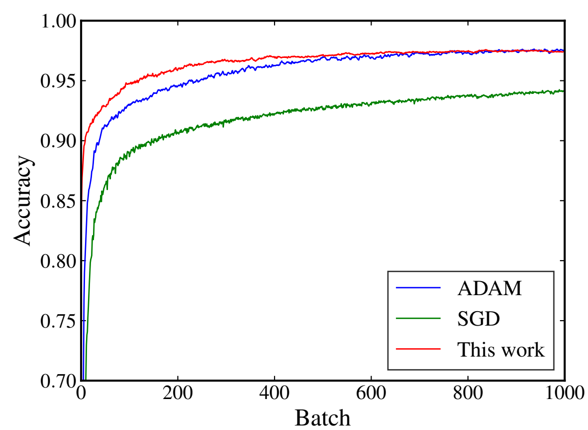

Figure 2: Test set performance of Fenchel lifted networks and fully connected networks trained using Adam and SGD on a 784-300-10 network architecture on MNIST with cross entropy loss. Total training time was 10 epochs. Final test set performances: **SGD** 0.943, **Adam** 0.976, **This work** 0.976.

As shown in Figure 2, Fenchel lifted networks learn faster than traditional networks as shown by the red curve being consistently above the blue and green curve. Although not shown, between batch 600 and 1000, the accuracy on a training batch would consistently hit 100% accuracy. The advantage of the Fenchel lifted networks is clear in the early stages of training, while towards the end the test set accuracy and the accuracy of an Adam-trained network converge to the same values.

[2]Code available in https://github.com/PotatoThanh/ADMM-NeuralNetworks

[3]Code available in https://github.com/deeplearning-math/bcd_dnn

We also compare Fenchel lifted networks against a LeNet-5 convolutional neural network on MNIST. The network architecture is 2 convolutional layers followed by 3 fully-connected layers and a cross entropy loss on the last layer. We use ReLU activations and average pooling in our implementation. Figure 3 plots the test set accuracy for the different models.

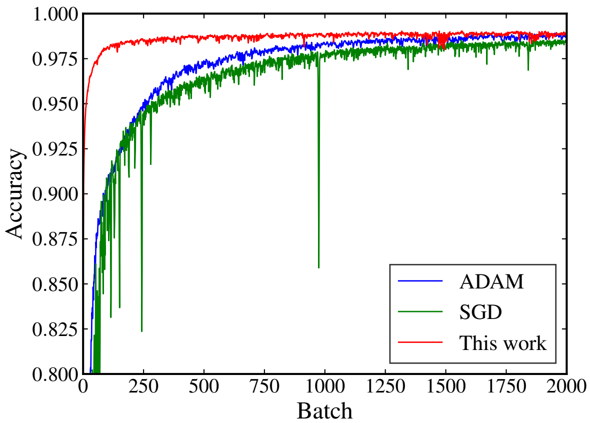

Figure 3: Test set performance of Fenchel lifted networks and LeNet-5 trained using Adam and SGD on MNIST with a cross entropy loss. Total training time was 20 epochs. Final test set performances: **SGD** 0.986, **Adam** 0.989, **This work** 0.990.

In Figure 3, our method is able to nearly converge to its final test set accuracy after only 2 epochs while Adam and SGD need the full 20 epochs to converge. Furthermore, after the first few batches, our model is attaining over 90% accuracy on the test set while the other methods are only at 80%, indicating that our model is doing something different (in a positive way) compared to traditional networks, giving them a clear advantage in test set accuracy.

5.3 Fenchel lifted networks vs CNN on CIFAR-10

In this section, we compare the LeNet-5 architechture and with Fenchel lifted networks on CIFAR-10. Figure 4 compares the accuracies of the different models.

In this case, the Fenchel lifted network still outperforms the SGD trained network and only slightly under performs compared to the Adam trained network. The larger variability in the accuracy per batch for our model can be attributed to the fact that in this experiment, when updating the W-variables, we would only take one gradient step instead of solving (19) and (20) to completion. We did this because we found empirically solving those respective sub-problems to completion would lead to poor performance at test time.

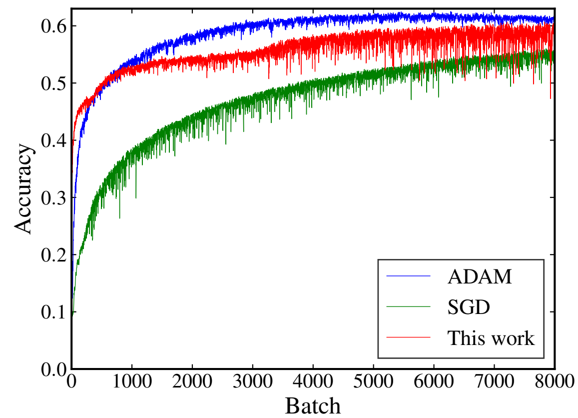

Figure 4: Test set performance of Fenchel lifted networks and LeNet-5 trained using Adam and SGD on CIFAR-10 with a cross entropy loss. Total training time was 80 epochs. Final test set performance: **SGD** 0.565, **Adam** 0.625, **This work** 0.606

6 Conclusion and Future Work

In this paper we propose Fenchel lifted networks, a family of models that provide a rigorous lower bound of the traditional neural network training problem. Fenchel lifted networks are similar to other methods that lift the dimension of the training problem, and thus exhibit many desirable properties in terms of scalability and the parallel structure of its sub-problems. As a result, we show that our family of models can be trained efficiently using block coordinate descent where the sub-problems can be parallelized across data points and/or layers. Unlike other similar lifted methods, Fenchel lifted networks are able to compete with traditional fully connected and convolutional neural networks on standard classification data sets, and in some cases are able to outperform them.

Future work will look at extending the ideas presented here to Recurrent Neural Networks, as well as exploring how to use the class of models described in the paper to train deeper networks.

Acknowledgements

The authors would like to acknowledge support from Berkeley Artificial Intelligence Research (BAIR) and Tsinghua-Berkeley-Shenzhen Institute (TBSI).

References

Askari, A., Negiar, G., Sambharya, R., and El Ghaoui, L. (2018). Lifted neural networks. *arXiv preprint arXiv:1805.01532*.

Carreira-Perpinan, M. and Wang, W. (2014). Distributed optimization of deeply nested systems. In Kaski, S. and Corander, J., editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 10–19, Reykjavik, Iceland. PMLR.

Carreira-Perpinán, M. A. and Alizadeh, M. (2016). Parmac: distributed optimisation of nested functions, with application to learning binary autoencoders. *arXiv preprint arXiv:1605.09114*.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2016). Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

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

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957.

Lau, T. T.-K., Zeng, J., Wu, B., and Yao, Y. (2018). A proximal block coordinate descent algorithm for deep neural network training. *Workshop track - International Conference on Learning Representations*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, J., Fang, C., and Lin, Z. (2019). Lifted proximal operator machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4181–4188.

Negiar, G., Askari, A., and Ghaoui, L. E. (2017). OPTML 2017 : Lifted Neural Networks for Weight Initialization.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). 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, pages III–1139–III–1147. JMLR.org.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. (2016). Training neural networks without gradients: A scalable admm approach. In *International Conference on Machine Learning*, pages 2722–2731.

Zhang, Z. and Brand, M. (2017). Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1721–1730.

Zhang, Z., Chen, Y., and Saligrama, V. (2016). Efficient training of very deep neural networks for supervised hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1487–1495.