Abstract

We present a novel layerwise optimization algorithm for the learning objective of Piecewise-Linear Convolutional Neural Networks (PL-CNNs), a large class of convolutional neural networks. Specifically, PL-CNNs employ piecewise linear non-linearities such as the commonly used ReLU and max-pool, and an SVM classifier as the final layer. The key observation of our approach is that the problem corresponding to the parameter estimation of a layer can be formulated as a difference-of-convex (DC) program, which happens to be a latent structured SVM problem. We optimize the DC program using the concave-convex procedure in conjunction with the block-coordinate Frank-Wolfe algorithm. Using publicly available data sets, we show that our approach outperforms the state of the art variants of backpropagation for learning PL-CNNs.

1 Introduction

Convolutional neural networks (CNNs) can automatically learn powerful discriminative representations of visual data, thereby providing vast improvements for several structured prediction tasks in computer vision such as object recognition, semantic segmentation and pose estimation. Given a training data set, the parameters of a CNN need to be estimated by solving a challenging large-scale non-convex optimization problem. Typically, this problem is solved using the backpropagation algorithm [18] whose success relies heavily on the values of its hyperparameters such as the learning rate and momentum. Recently, several methods have been proposed to fix the learning rate of backpropagation, including Adam [9], Adagrad [5], and Adadelta [20]. However, since backpropagation is a form of stochastic subgradient descent, none of these methods provide a monotonic decrease in the learning objective. Furthermore, they do not offer any guarantee of convergence and still require the tuning of an initial learning rate for good performance.

The CNN framework is in stark contrast to structured support vector machines (SVM), which formulate any structured prediction task as a convex optimization problem [8]. Their parameters can then be efficiently estimated with customized solvers such as the cutting plane algorithms [7] or the block-coordinate Frank-Wolfe (BCFW) method [11].

In this work, we propose a new optimization algorithm that combines the best of both worlds: the accuracy and power of a CNN, with the convex formulation of a structured SVM. Specifically, we consider a CNN that employs piecewise linear non-linearities such as ReLU and max-pool, which are commonly used in computer vision. We refer to this class of networks as Piecewise-Linear CNN (PL-CNN). Our algorithm performs a layerwise optimization. That is, it iteratively updates the parameters of one layer while keeping the parameters of all other layers fixed. The key observation of our approach is that the problem of estimating the parameters of a single layer can be formulated as a difference-of-convex (DC) program, which happens to be a latent SVM-type problem. We solve it using the iterative concave-convex procedure (CCCP) [19] in combination with an improved BCFW algorithm.
Using standard network architectures, and the widely used MNIST and CIFAR-10 data sets, we show that our algorithm outperforms the state of the art backpropagation variants for learning PL-CNNs.

This manuscript only provides a brief overview of the approach, a more detailed description can be found at [https://arxiv.org/abs/1611.02185](https://arxiv.org/abs/1611.02185)

1.1 Related Work

While some of the early successful approaches to deep neural networks optimization relied on greedy layer-wise training [2][6], most currently used methods are variants of backpropagation [18] with adaptive learning rates, as discussed above in the introduction.

Second-order and natural gradient optimization methods have also been a subject of attention [4][12][13][14]. Though providing more informative updates and solid theoretical support than SGD-based approaches, these methods do not take into account the structure of the problem offered by the commonly used non-linear operations.

Recently, work has been proposed to exploit the structure of CNNs and use layer-wise strategies: the authors of [17] suggest to use ADMM for massive distribution of computation, and in [21], the class of CNNs is restricted, allowing for convexification of the problem. Like these approaches, our work exploits the properties of the CNN for a tailored optimization algorithm. This enables our method to guarantee convergence to a critical point as well as a monotonic decrease in the objective function.

2 Structure of Convolutional Neural Networks

We denote the parameters of a CNN by \( W = \{W^l; l = 1, \ldots, L\} \) where \( W^l \) are the parameters of the \( l \)-th layer and \( L \) is the total number of layers. The CNN defines a composite function, that is, the output of layer \( l-1 \), denoted by \( z^{l-1} \), is the input to the layer \( l \). Given the input \( z^{l-1} \) to layer \( l \), the output is computed as \( z^l = a'(W^l \cdot z^{l-1}) \), where “\( \cdot \)” is either a convolution or a matrix multiplication, and \( a' \) is a non-linear activation. The input to the first layer is a data sample such as an image \( x \), that is, \( z^1 = x \). The output of the final layer provides a concise representation of \( x \), which we denote by \( z^L = \Phi(x; W) \).

We consider a training data set \( D = \{(x_i, y_i)\}_{i \in [n]} \) where \( x_i \) is the input and \( y_i \) is the ground-truth output for sample \( i \). Then for each \( x_i \), the CNN outputs a feature \( \Phi(x_i; W) \) fed to the SVM loss layer with parameters \( W^{svm} \), which computes the hinge loss for the objective function (1). The goal of this work is to learn the parameters \( W \) and \( W^{svm} \) given the training data \( D \). This involves minimizing the following non-convex objective function:

\[
\min_{W, W^{svm}} \lambda \frac{1}{2} \sum_{l \in [1:L]} \|W^l\|_F^2 + \frac{1}{n} \sum_{i=1}^{n} \max_{y_i \in \mathcal{Y}} \left( \Delta(\bar{y}_i, y_i) + (W^{svm}_{\bar{y}_i} - W^{svm}_{y_i})^T \Phi(x_i; W) \right) \tag{1}
\]

In (1), \( \Delta(y_i, \bar{y}_i) \) is the zero-one classification loss for the ground truth \( y_i \) and a label \( \bar{y}_i \) in the set of possible labels \( \mathcal{Y} \). We have also used the notation \( W^{svm}_{\bar{y}_i} \) to represent the block of coordinates corresponding to the label \( \bar{y}_i \) in \( W^{svm} \). Although we focus on classification in this work, the flexibility of the \( \Delta \) loss function in structured SVMs allows our framework to be extended to many other problems in computer vision or machine learning.

We require the non-linear activations \( a^l \) to be piecewise-linear, which includes the commonly used Rectified Linear Units (ReLU) and max-pooling. As a composition of these and of linear layers, \( \Phi(x, W) \) is piecewise linear in any of its parameters \( W^l \). Finally, the hinge loss of the SVM layer is also a piecewise linear mapping. The objective function is therefore the sum of a quadratic regularization term and a piecewise linear term. This is a specific structure which we exploit in our optimization algorithm.

3 Optimization

We present here the main steps of the approach:

**Layer-wise Optimization** We perform layer-wise optimization to take advantage of the piecewise linearity of the hinge loss. In the space of parameters, optimizing one layer corresponds to optimizing
one block of coordinates. Say we want to optimize layer \( l \). For the objective function, this means that \( W_l \) is seen as a variable while all \( W_{l'} \) for \( l' \neq l \) are considered as fixed parameters.

**Difference of Convex Programming** We identify the optimization problem of layer \( l \) as a Difference of Convex (DC) problem. An explicit DC decomposition further reveals that the problem is a latent SVM one, where the latent variables are the non-linear activations of the network. To solve this problem, we use the Concave Convex Procedure (CCCP), which iteratively linearizes the concave part and solves the resulting approximate convex problem. This guarantees to have the objective function decrease at every step, and converge at a critical point (15, 16).

**Block-Coordinate Frank Wolfe Algorithm** In order to minimize the convex problem in the inner iteration of the CCCP, we employ the BCFW algorithm, which iteratively computes the conditional gradient of the dual problem. The BCFW algorithm features a monotonic increase in the dual objective, a closed-form optimal step size for the dual as well as convergence certificates. We improve the BCFW in three significant ways: (i) we include a trust-region for the parameters, which allows us to use the previous parameters as an initialization; (ii) we reduce the memory requirement of BCFW by potentially several orders of magnitude for the dense layers; and (iii) we observe that, empirically, the optimal solution of the structured SVM problem can be obtained efficiently by solving a subproblem which contains only a small fraction of the constraints. These improvements are further described in the appendix.

### 4 Results

**Datasets & Architectures** In order to ensure convergence of both our method and the baselines, we use the standard MNIST and CIFAR-10 data sets. These data sets allow us to run the solvers for a substantial number of epochs to get sufficiently close to convergence. The MNIST data set consists of 60,000 gray scale images of size 28 \( \times \) 28 for handwritten digits classification (50,000 for training and validation, and 10,000 for testing). There are 10 classes, one for every digit from zero to nine. The architecture used for MNIST is given in Table 1. The CIFAR-10 data set is a collection of 60,000 RGB natural images of size 32 \( \times \) 32 [10]. There are 50,000 images for training and 10,000 images for testing, with 10 possible classes. For this data set, we use the architecture in Table 2.

<table>
<thead>
<tr>
<th>Conv + ReLU</th>
<th>Max-Pooling</th>
<th>Conv + ReLU</th>
<th>Max-Pooling</th>
<th>FC + ReLU</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 filters</td>
<td>patches: 2 ( \times ) 2</td>
<td>12 filters</td>
<td>patches: 5 ( \times ) 5</td>
<td>2 ( \times ) 2</td>
<td>256</td>
</tr>
<tr>
<td>5 ( \times ) 5</td>
<td></td>
<td>2 ( \times ) 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conv</th>
<th>Conv</th>
<th>Max-Pooling</th>
<th>Max-Pooling</th>
<th>FC</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ ReLU</td>
<td>+ ReLU</td>
<td>+ ReLU</td>
<td>+ ReLU</td>
<td>+ ReLU</td>
<td>+ ReLU</td>
</tr>
<tr>
<td>12 filters</td>
<td>12 filters</td>
<td>patches: 24 filters</td>
<td>24 filters</td>
<td>patches: 512</td>
<td>10</td>
</tr>
<tr>
<td>3 ( \times ) 3</td>
<td>3 ( \times ) 3</td>
<td>2 ( \times ) 2</td>
<td>3 ( \times ) 3</td>
<td>2 ( \times ) 2</td>
<td></td>
</tr>
</tbody>
</table>

**Methods** Our experiments are designed to assess the ability of LW-SVM (Layer-Wise SVM, our method) and the SGD baselines to optimize Problem (1). To compare LW-SVM with the state-of-the-art variants of backpropagation, we look at not only the training and testing accuracies but also the training objective value. As the objective function needs to be comparable across the methods, we use the same PL non-linearities for all the methods, without introducing batch-normalization. Nevertheless, we point out that batch normalization is a difference-of-convex function, and could therefore be introduced in our framework with the same convergence and monotonic decrease guarantees - using another convex solver than BCFW, since batch-normalization is not piecewise linear. Moreover, unlike dropout, which effectively learns an ensemble model, we learn a single model using each baseline optimization algorithm. Note however that we could apply our method to learning ensembles of models and thus use it with dropout. All experiments are conducted on a GPU (Nvidia GTX Titan X), and use Theano [1, 3]. We compare LW-SVM against Adagrad, Adadelta and Adam, in the conditions presented below.
On MNIST, we run Adam and Adadelta for 100 epochs, and Adagrad on 200 epochs before running LW-SVM. Results are presented in Table 3. On CIFAR-10, Adam and Adadelta are trained for 500 epochs and Adagrad for 1000 epochs before using LW-SVM. We also compare these performances with LW-SVM trained from scratch for CIFAR-10. The results are shown in Table 4.

The layer-wise schedule of our algorithm is as follows: as long as the validation accuracy increases, we perform passes from the end of the network (SVM) to the beginning (conv1). At each pass, each layer is optimized with one iteration of the CCCP until the dual objective function does not increase anymore (by more than 1%).

**Hyperparameters** The regularization hyper-parameter $\lambda$ is chosen by cross-validation and is set to 0.001 for all algorithms and data sets. The proximal weight $\mu$ is set to $10\lambda$, i.e. 0.001. The initial learning rates are chosen by cross-validation. For Adagrad, Adadelta and Adam, they are respectively set to 0.01, 1 and 0.001 on MNIST, and to 0.001, 1 and 0.0001 on CIFAR-10.

**Results** It can be seen in Tables 3 and 4 that (i) LW-SVM always improves over the solution of the SGD algorithm, for example on CIFAR-10, decreasing the objective value of Adam from 0.08 to 0.06, or improving the test accuracy of Adadelta from 68.1% to 70.9%, and (ii) LW-SVM can reach similar accuracies on the test data set when applied from scratch. Due to the regularization of the proximal term, the evolution of our algorithm leads to a better generalization, although it takes longer and has a higher training objective.

### Table 3: Results on MNIST

<table>
<thead>
<tr>
<th>Solver (epochs)</th>
<th>Training Objective</th>
<th>Training Accuracy</th>
<th>Time (s)</th>
<th>Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adagrad</td>
<td>0.028</td>
<td>99.94%</td>
<td>362</td>
<td>99.13%</td>
</tr>
<tr>
<td>Adagrad + LW-SVM</td>
<td>0.025</td>
<td>99.95%</td>
<td>362+233</td>
<td>99.17%</td>
</tr>
<tr>
<td>Adadelta</td>
<td>0.051</td>
<td>99.41%</td>
<td>190</td>
<td>99.13%</td>
</tr>
<tr>
<td>Adadelta + LW-SVM</td>
<td>0.036</td>
<td>99.84%</td>
<td>190+136</td>
<td>99.32%</td>
</tr>
<tr>
<td>Adam</td>
<td>0.040</td>
<td>99.68%</td>
<td>183</td>
<td>99.34%</td>
</tr>
<tr>
<td>Adam + LW-SVM</td>
<td>0.031</td>
<td>99.87%</td>
<td>183+171</td>
<td>99.36%</td>
</tr>
</tbody>
</table>

### Table 4: Results on CIFAR-10

<table>
<thead>
<tr>
<th>Solver (epochs)</th>
<th>Training Objective</th>
<th>Training Accuracy</th>
<th>Time (s)</th>
<th>Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adagrad</td>
<td>0.136</td>
<td>99.79%</td>
<td>2692</td>
<td>62.62%</td>
</tr>
<tr>
<td>Adagrad + LW-SVM</td>
<td>0.104</td>
<td>99.86%</td>
<td>2692+1554</td>
<td>62.83%</td>
</tr>
<tr>
<td>Adadelta</td>
<td>0.350</td>
<td>96.59%</td>
<td>1613</td>
<td>68.11%</td>
</tr>
<tr>
<td>Adadelta + LW-SVM</td>
<td>0.287</td>
<td>100.00%</td>
<td>1613+870</td>
<td>70.90%</td>
</tr>
<tr>
<td>Adam</td>
<td>0.080</td>
<td>99.91%</td>
<td>1550</td>
<td>66.00%</td>
</tr>
<tr>
<td>Adam + LW-SVM</td>
<td>0.060</td>
<td>100.00%</td>
<td>1550+637</td>
<td>66.60%</td>
</tr>
<tr>
<td>LW-SVM</td>
<td>0.5</td>
<td>86.46%</td>
<td>5500</td>
<td>68.4%</td>
</tr>
</tbody>
</table>

### 5 Discussion

We presented a novel layerwise optimization algorithm for a large and useful class of convolutional neural networks. Our key observation is that the optimization of the parameters of one layer is equivalent to solving a latent structured SVM problem. As the problem is a DC program, it naturally lends itself to the iterative CCCP approach, which optimizes a convex structured SVM objective at each iteration. This allows us to leverage the advancements made in structured SVM optimization over the past decade to design a computationally feasible approach for learning such CNNs. Specifically, we use the BCFW algorithm and extend it to improve its initialization, memory requirements and time complexity. In particular, this allows our method to not require the tuning of any learning rate. Using the publicly available MNIST and CIFAR-10 data sets, we show that our approach outperforms the state of the art variants of backpropagation for learning PL-CNNs.
References


A Difference of Convex Decomposition

DC Decomposition in a Layer The DC decomposition for any layer can be broken down to decomposing linear operations (convolution or matrix multiplication) and non-linear activations:

For linear operations, we use the fact that a non-negative linear combinations of convex functions is still convex. We separate the weights of a linear layer in $W = W^+ - W^-$, where both $W^+$ and $W^-$ are of the same size of $W$ and have exclusively non-negative coefficients. Suppose we have a DC decomposition of the input $x$: $x = x^{cvx} - x^{ccv}$, where $x^{cvx}$ and $x^{ccv}$ are both convex functions. Then we can decompose the output $y$ of the linear layer by noting that:

$$y = W \cdot x = (W^+ + W^-) \cdot (x^{cvx} - x^{ccv})$$

The two terms are convex because they are a non-negative linear combination of convex functions.

For non-linearities, we note that both ReLUs, max-pooling and the hinge loss can be written in the following form: $z = \max_{k \in K}(y_k)$. Then given a DC decomposition of the inputs $y_k = y_k^{cvx} - y_k^{ccv}$, we write the explicit DC decomposition:

$$z = \max_{k \in K}(y_k) = \max_{k \in K}(y_k^{cvx} - y_k^{ccv}) = \max_{k \in K} \left( y_k^{cvx} + \sum_{j \in K \setminus \{k\}} y_j^{ccv} \right) - \sum_{j \in K} y_j^{ccv}$$

Resulting Decomposition By successively applying the above method until the output of the SVM, we obtain a difference-of-convex decomposition for the hinge loss, thereby we can obtain an explicit difference-of-convex program. We summarize the decomposition in the algorithm below:

Algorithm 1 DC Decomposition for layer $l$

1: Get $x^l$ input of layer $l$ (standard forward pass until layer $l$)
2: $x^{cvx} \leftarrow W^l \cdot x^l$
3: $x^{ccv} \leftarrow 0$
4: for each layer from $l + 1$ to SVM do
5:   $x^{cvx}, x^{ccv} \leftarrow$ DC_Decomposition($x^{cvx}, x^{ccv}$) {Use (2) and (3)}
6: end for

We easily check that the two terms of the initial decomposition $W^l \cdot x^l$ and 0 are convex functions of $W^l$. Then by induction, using at each layer equation (2) for the linear operation and (3) for the activation, the final decomposition $x^{cvx} - x^{ccv}$ is an explicit difference-of-convex functions of $W^l$. We remark that the operations are the same as the ones used in a standard forward pass. Therefore we have obtained a difference-of-convex decomposition with approximately the same time complexity as a standard forward pass.

Moreover we can easily get gradients by a backward pass, for example using the automatic differentiation tools of the standard deep learning libraries.
B SVM Formulation & Dual Derivation

Multi-Class SVM Suppose we are given a data set of \( N \) samples, for which every sample \( i \) has a feature vector \( \phi_i \in \mathbb{R}^d \) and a ground truth label \( y_i \in \mathcal{Y} \). For every possible label \( \overline{y}_i \in \mathcal{Y} \), we introduce the augmented feature vector \( \psi_i(\overline{y}_i) \in \mathbb{R}^{|\mathcal{Y}| \times d} \) containing \( \phi_i \) at index \( \overline{y}_i \), \( -\phi_i \) at index \( y_i \), and zeros everywhere else (then \( \psi_i(y_i) \) is just a vector of zeros). We also define \( \Delta(\overline{y}_i, y_i) \) as the loss by choosing the output \( \overline{y}_i \) instead of the ground truth \( y_i \) in our task. For classification, this is the zero-one loss for example.

The SVM optimization problem is formulated as:

\[
\min_{w, \xi} \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{i=1}^{N} \xi_i \\
\text{subject to: } \forall i \in [N], \forall \overline{y}_i \in \mathcal{Y}, \xi_i \geq w^T \psi_i(\overline{y}_i) + \Delta(y_i, \overline{y}_i)
\]

Where \( \lambda \) is the regularization hyperparameter. We now add a proximal term to a given starting point \( w_0 \):

\[
\min_{w, \xi} \frac{\lambda}{2} \|w\|^2 + \frac{\mu}{2} \|w - w_0\|^2 + \frac{1}{N} \sum_{i=1}^{N} \xi_i \\
\text{subject to: } \forall i \in [N], \forall \overline{y}_i \in \mathcal{Y}, \xi_i \geq w^T \psi_i(\overline{y}_i) + \Delta(y_i, \overline{y}_i)
\]

Factorizing the second-order polynomial in \( w \), we obtain the equivalent problem (changed by a constant):

\[
\min_{w, \xi} \frac{\lambda + \mu}{2} \|w - \frac{\mu}{\lambda + \mu} w_0\|^2 + \frac{1}{N} \sum_{i=1}^{N} \xi_i \\
\text{subject to: } \forall i \in [N], \forall \overline{y}_i \in \mathcal{Y}, \xi_i \geq w^T \psi_i(\overline{y}_i) + \Delta(y_i, \overline{y}_i)
\]

For simplicity, we introduce the ratio \( \rho = \frac{\mu}{\lambda + \mu} \).

**Dual Objective Function** The primal problem is:

\[
\min_{w, \xi} \frac{\lambda + \mu}{2} \|w - \rho w_0\|^2 + \frac{1}{N} \sum_{i=1}^{N} \xi_i \\
\text{subject to: } \forall i \in [N], \forall \overline{y}_i \in \mathcal{Y}, \xi_i \geq w^T \psi_i(\overline{y}_i) + \Delta(y_i, \overline{y}_i)
\]

The dual problem can be written as:

\[
\max_{\alpha \geq 0} \min_{w, \xi} \frac{\lambda + \mu}{2} \|w - \rho w_0\|^2 + \frac{1}{N} \sum_{i=1}^{N} \xi_i + \frac{1}{N} \sum_{i=1}^{N} \sum_{y_i \in \mathcal{Y}} \alpha_i(\overline{y}_i) \left( \Delta(y_i, \overline{y}_i) + w^T \psi_i(\overline{y}_i) - \xi_i \right)
\]

Then we obtain the following KKT conditions:

\[
\forall i \in [N], \frac{\partial}{\partial \xi_i} = 0 \rightarrow \sum_{\overline{y}_i \in \mathcal{Y}} \alpha_i(\overline{y}_i) = 1 \\
\frac{\partial}{\partial w} = 0 \rightarrow w = \rho w_0 - \frac{1}{N} \frac{1}{\lambda + \mu} \sum_{i=1}^{N} \sum_{y_i \in \mathcal{Y}} \alpha_i(\overline{y}_i) \psi_i(\overline{y}_i)
\]
We inject back and simplify to:

\[
\max_{\alpha \in P_n(\mathcal{Y})} \frac{-(\lambda + \mu)}{2} \|A\alpha\|^2 + \mu w_0^T (A\alpha) + \alpha^T b
\]

Finally:

\[
\min_{\alpha \in P_n(\mathcal{Y})} f(\alpha)
\]

Where:

\[
f(\alpha) \triangleq \frac{\lambda + \mu}{2} \|A\alpha\|^2 - \mu w_0^T (A\alpha) - \alpha^T b
\]

**BCFW Derivation** We write \( \nabla_{(i)} f \) the the gradient of \( f \) w.r.t. the block \( (i) \) of variables in \( \alpha \), padded with zeros on blocks \( (j) \) for \( j \neq i \). Similarly, \( A_{(i)} \) and \( b_{(i)} \) contain the rows of \( A \) and the elements of \( b \) for the block of coordinates \( (i) \) and zeros elsewhere. We can write:

\[
\nabla_{(i)} f(\alpha) = (\lambda + \mu) A_{(i)}^T A \alpha - \mu A_{(i)} w_0 - b_{(i)}
\]

Then the search corner for the block of coordinates \( (i) \) is given by:

\[
s_i = \arg\min_{s'_i} \left( <s'_i, \nabla_{(i)} f(\alpha) > \right)
\]

\[
= \arg\min_{s'_i} \left( (\lambda + \mu) A^T A_{(i)} s'_i - \mu w_0^T A_{(i)} s'_i - b_{(i)}' s'_i \right)
\]

We replace:

\[
A\alpha = \rho w_0 - w
\]

\[
A_{(i)} s_i' = \frac{1}{N} \frac{1}{\lambda + \mu} \sum_{y_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i)
\]

\[
b_{(i)}^T s_i' = \frac{1}{N} \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \Delta(\bar{y}_i, y_i)
\]

We then obtain:

\[
s_i = \arg\min_{s'_i} \left( -(w - \rho w_0)^T \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) - w_0^T \rho \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) - \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \Delta(\bar{y}_i, y_i) \right)
\]

\[
= \arg\max_{s'_i} \left( w^T \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) + \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \Delta(\bar{y}_i, y_i) \right)
\]

As expected, this maximum is obtained by setting \( s_i \) to one at \( y_i^* = \arg\max_{\bar{y}_i \in \mathcal{Y}} \left( w^T \psi_i(\bar{y}_i) + \Delta(\bar{y}_i, y_i) \right) \) and zeros elsewhere. We introduce the notation:

\[
w_i = -A_{(i)} \alpha_{(i)}
\]

\[
l_i = b_{(i)}^T \alpha_{(i)}
\]

\[
w_s = -A_{(i)} s_i
\]

\[
l_s = b_{(i)}^T s_i
\]
Then we have:

\[
    w_s = -\frac{1}{N} \frac{1}{\lambda + \mu} \psi(y_i^*) = -\frac{1}{N} \frac{1}{\lambda + \mu} \frac{\partial H_i(y_i^*)}{\partial w} \\
    l_s = \frac{1}{N} \Delta(y_i, y_i^*)
\]

The optimal step size in the direction of the block of coordinates \((i)\) is given by:

\[
    \gamma^* = \arg \min_{\gamma} f(\alpha + \gamma(s_i - \alpha_i))
\]

The optimal step-size is given by:

\[
    \gamma^* = \frac{\langle \nabla_i f(\alpha), s_i - \alpha_i \rangle}{(\lambda + \mu)\|A(s_i - \alpha_i)\|^2}
\]

We introduce \(w_d = -A\alpha = w - \rho w_0\). Then we obtain:

\[
    \gamma^* = \frac{(w_i - w_s)^T(w - \rho w_0) + \rho w_0^T(w_i - w_s) - \frac{1}{\lambda + \mu}(l_i - l_s)}{\|w_i - w_s\|^2}
\]

\[
    = \frac{(w_i - w_s)^T w - \frac{1}{\lambda + \mu}(l_i - l_s)}{\|w_i - w_s\|^2}
\]

And the updates are the same as in standard BCFW:

**Algorithm 2** BCFW with warm start

1: Let \(w_i^{(0)} = w_0, \ \forall i \in [N], \ w_s^{(0)} = 0\)
2: Let \(l_i^{(0)} = 0, \ \forall i \in [N], \ l_s^{(0)} = 0\)
3: for \(k=0...K\) do
4: Pick \(i\) randomly in \(\{1, ..., n\}\)
5: Get \(y_i^* = \arg \max_{y_i \in Y} H_i(\bar{y}_i, w^{(k)})\) and \(w_s = -\frac{1}{N} \frac{1}{\lambda + \mu} \frac{\partial H_i(y_i^*, w^{(k)})}{\partial w}\)
6: \(l_s = \frac{1}{N} \Delta(y_i^*, y_i)\)
7: \(\gamma = \frac{(w_i - w_s)^T w - \frac{1}{\lambda + \mu}(l_i - l_s)}{\|w_i - w_s\|^2}\) clipped to \([0, 1]\)
8: \(w_i^{(k+1)} = (1 - \gamma)w_i^{(k)} + \gamma w_s\)
9: \(l_i^{(k+1)} = (1 - \gamma)\Delta^{(k)} + \gamma l_s\)
10: \(w_s^{(k+1)} = w_s^{(k)} + \gamma l_i^{(k+1)} - w_i^{(k)} = w_i^{(k)} + \gamma (w_s^{(k)} - w_i^{(k)})\)
11: \(l^{(k+1)} = l^{(k)} + l_i^{(k+1)} - l_i^{(k)}\)
12: end for

**Feature Compression for Dense Layers** The BCFW algorithm requires us to store a linear combination of the feature vectors for each mini-batch. While this requirement is not too stringent for convolutional and multi-class SVM layers, where the dimensionality of the feature vectors is small, it becomes prohibitively expensive for dense layers. However this issue is avoided with the following remark: when optimizing dense layer \(l\), if \(W^l \in \mathbb{R}^{p \times q}\), we can compress the joint feature vectors \(\Psi(x, y, h)\) to a vector of size \(p\). This is in contrast to the naïve approach that requires them to be of size \(p \times q\).

To see that, we point out that the feature vectors are subgradients of the hinge loss function, which we loosely denote by \(\eta\) here. Then by the chain rule: \(\frac{\partial \eta}{\partial v^T} = \frac{\partial \eta}{\partial z^T} \cdot \frac{\partial v^T}{\partial z^T} \cdot (z^{l-1})^T\). Noting that \(z^{l-1} \in \mathbb{R}^q\) is a forward pass up until layer \(l\) (independent of \(W^l\)), we can store only \(\frac{\partial \eta}{\partial z^T} \in \mathbb{R}^p\) and still reconstruct the full feature vector \(\frac{\partial \eta}{\partial W^l}\) by a forward pass and an outer product.

9
Reducing the Number of Constraints  In order to reduce the amount of time required for the BCFW algorithm to converge, we use the structure of $\mathcal{H}$ to simplify the full latent SVM problem to a much simpler problem. We have mentioned that the latent space $\mathcal{H}$ represents the activations of the network. Therefore it has a natural decomposition over the layers: $\mathcal{H} = \mathcal{H}_1 \times \ldots \times \mathcal{H}_L$.

Subsequently, the full latent problem can be approximately solved by optimizing the dual problem on increasingly large search spaces. In other words, we start with constraints of $\mathcal{Y}$, followed by $\mathcal{Y} \times \mathcal{H}_L$, then $\mathcal{Y} \times \mathcal{H}_L \times \mathcal{H}_{L-1}$ and so on. The algorithm converges when the primal-dual gap is below tolerance. This is effective in practice because at optimality, the activations selected by the convex and the concave part are similar. Subsequently, the solutions of the convex part (i.e. the chosen activations for the convex part) often lie in a small space.