CS230

Adapting Convex Techniques for Greedy Layerwise Learning

Shyamoli Sanghi
Department of Computer Science
Stanford University
shyamoli@stanford.edu

Elijah Spiegel
Department of Philosophy
Stanford University
spiegele@stanford.edu

Abstract

Convex duals of nonconvex optimization problems allow us to find optimal solutions quickly and efficiently. Recent work by [Pilanci & Ergen 2020][7] has found nonconvex duals for the task of training shallow, fully-connected ReLU networks. In this paper, we explore extending convex training techniques to deeper networks by using a greedy layerwise training approach (introduced in the paper Belilovsky et al. 2019)[3]). We find that adding layers in this way often allows us to increase test accuracy. Further, column normalization is required to make layerwise learning effective in this context. We achieved final test accuracy of 50.1% on the CIFAR-10 dataset compared to 45.9% with single-layer architectures.

1 Introduction

High computational costs and difficulty of achieving global optimality make it difficult to develop and deploy neural networks in high-stakes, time-constrained or resource-limited contexts. Recent work[7] has shown that it is often possible to train the nonconvex duals of convex optimization problems, achieving global optimality faster. This work has so far focused on shallow networks. We combine convex approaches with greedy layerwise learning to evaluate utilizing convex techniques in developing deep neural networks. We use the CIFAR-10 task as a benchmark to develop this architecture. Our architecture consists of a stack of convex layers (which we describe) that yields as output a classification of the input image among ten possible class labels.

In this paper, we explore approaches to performing greedy layerwise training on CIFAR-10 using convex techniques, and seek to determine whether we can gain better results in terms of accuracy and training speed, than corresponding non-convex techniques. This problem involves trying to avoid the pitfalls of greedy learning while preserving the global optimality advantage of the convex approach.

The input to our problem is a set of images. Our model uses stacked convex layers, that are duals of two-layer fully connected network, to output labels, that classify the images into different categories.

(This is a cross-project with CS 231N. In this paper, we explore the basic combination of convex training and greedy layerwise training. In our 231N paper, we explore the combination of these techniques with the beta-lasso algorithm put forward by (Neyshabur et al. 2020)[6] and explore a second dataset.)

2 Related work

The following two papers inform the basis our project. They provide, respectively, the convex training approach and greedy layerwise training approach that inform our paper.

CS230: Deep Learning, Spring 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

1. 'Neural Networks are Convex Regularizers' [Pilanci & Ergen 2020][7]
2. 'Greedy Layerwise Learning Can Scale to Imagenet' [Belilovsky et al. 2019][3]

[Pilanci & Ergen 2020] explains how we can use convex techniques to find optima for two-layer FC ReLU networks. We will explain in more detail how this works in the Methods section. The drawback of the paper is that the convex dual developed only applies to two-layer networks. To get better results, we would like to be able to apply its techniques to deeper networks, but it is not immediately clear how we might do this. This is where we turn to (Belilovsky et al. 2019). That paper explains how to build deep networks by solving iterated subproblems of training two-layer networks. This paper could be further improved by applying convex techniques to the subproblems in order to allow faster training with a guarantee of optimality.

Several other papers have previously explored global optimization training. We looked at

1. 'Understanding deep neural networks with rectified linear units' [Arora et al. 2016][1]
2. 'Breaking the curse of dimensionality with convex neural networks' [Bach 2017][2]
3. 'Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods' [Janzamin et al. 2015][4]

The [Pilanci & Ergen 2020] paper is state of the art among the global optimization training papers since its convex dual problem is an exact solution whose complexity is polynomial in the number of training examples, input dimensions, and hidden neurons, whereas the other papers either present algorithms that are inexact, that scale exponentially with problem parameters, or fail to present a fully developed training algorithm. Thus, we use that paper as a starting point.

3 Dataset and Features

We trained and tested our model on the CIFAR-10 dataset with 49000 training examples and 1000 validation example images of dimensions 32*32*3. Thus, the images are of low resolution (32*32) and the features are the pixels of the images.

We did not need to pre-process this dataset but did use mean subtraction across the entire dataset to normalize the images.

The CIFAR-10 dataset contains images from 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. An example from each class is shown below.

The CIFAR-10 dataset was collected in (Krizhevsky et al. 2009)[5] and downloaded from http://www.cs.toronto.edu/ kriz/cifar.html

4 Methods

The backbone of our project is the convex dual result of [Pilanci & Ergen 2020]. [7] The standard, nonconvex problem of training a fully-connected two-layer ReLU network with L_2 regularization is given by

$$\min_{\{\alpha_j, u_j\}_{j=1}^m} \| \sum_{j=1}^m (Xu_j)_+ \alpha_j - y\|_2^2 + \frac{\beta}{2} \sum_{j=1}^m (\|u_j\|_2^2 + \alpha_j^2) \tag{1}$$

[Pilanci & Ergen 2020] show that the convex dual of this problem is given by

$$\min_{\{v_i, w_i\}_{i=1}^P} \frac{1}{2} \| \sum_{i=1}^P D_i X(v_i - w_i) - y\|_2^2 + \beta \sum_{i=1}^P (\|v_i\|_2 + \|w_i\|_2) \tag{2}$$

s.t. $\forall i : (2D_i - I_n)Xv_i \geq 0,\ (2D_i - I_n)Xw_i \geq 0$
in which the D's are diagonal matrices exhausting the matrices the form Diag($[1[x_1^T u \geq 0], \ldots, 1[x_n^T u \geq 0]]$ for arbitrary $u \in \mathbb{R}^d$.

Note that this assumes a scalar output, i.e. binary classification. In order to extend this to the multiclass setting, we train a binary classifier for each class label and we perform multiclass classification by predicting the label with maximum output. The learned V and W vectors of equation 2 are not themselves the weights of the optimal U and α of equation 1, but the optimal weights of the nonconvex problem can be recovered from them. In addition, since the number of D matrices grows exponentially, the function is estimated by randomly sampling possible D matrices. Instead of solving this problem exactly, we use gradient descent methods to minimize the expression, along with a straightforward translation of the inequality constraints into loss terms, i.e. adding them to the loss, weighted by a hyperparameter ρ.

For details of why the convex dual recovers the optimal solution to the nonconvex optimization problem, see [Pilanci & Ergen 2020]. In short, the convex dual converts the highly complex loss landscape of the nonconvex problem into one that can be optimized very quickly, and for which convergence to a value near the global optimum can be guaranteed. The layerwise training approach of [Belilovsky et al. 2019][3] proceeds by training two-layers of a convolutional network on the desired classification task, using as inputs the outputs of the (frozen) previously trained layers. [Belilovsky et al. 2019] then discard the second layer and freeze the first layer on top of the previously trained layers and use the outputs of this network as the inputs for training the next layer.

In our approach, we do not discard layers in this way. Instead, we train subsequent layers on top of the unsummed outputs of the previous layers. For example, when training a second layer, we take the $DX(V - W)$ output of the first layer, which in the binary classification case is a tensor of shape (Num_samples, P) and we use this as the input to our second layer. Let A_0 be the original image inputs to the network. Then $A_1 = DX(V_1 - W_1)$ would be the output of the first layer of the convex architecture. (Note that this corresponds to two layers with an intervening ReLU nonlinearity in the nonconvex domain.) Then the training problem of the second convex layer would be $\min_{\{v_i, w_i\}_{i=1}^P} \frac{1}{2} \| \sum_{i=1}^P D_i A_1 (v_i - w_i) - y \|_2^2 + \beta \sum_{i=1}^P (\|v_i\|_2 + \|w_i\|_2)$, subject to the previously described inequality constraints. Note that this applies to the binary classification case. The unsummed outputs in the multiclass setting are 3-dimensional tensors which include a dimension for the number of classes. This requires a pooling operation before being fed into the next layer, as we will discuss in the next section.

5 Experiments/Results/Discussion

After conducting several experiments and tuning our model's hyperparameters, the optimal results we obtained were:

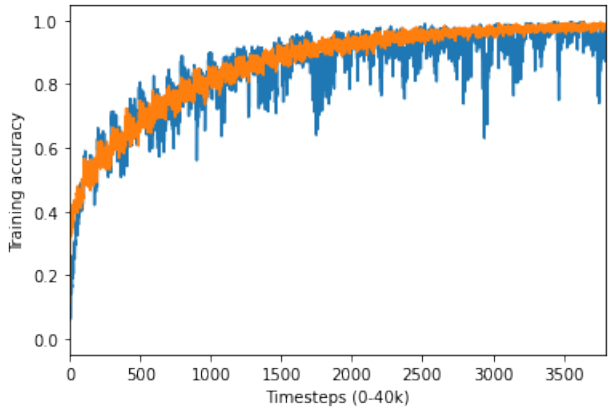
Training Loss (two layers)	0.116
Training Loss (baseline, one layer)	0.116
Training Accuracy (two layers)	97.2%
Training Accuracy (baseline, one layer)	98.4%
Test Loss (two layers)	0.377
Test Loss (baseline, one layer)	0.494
Test Accuracy (two layers)	**50.1%**
Test Accuracy (baseline, one layer)	45.9%

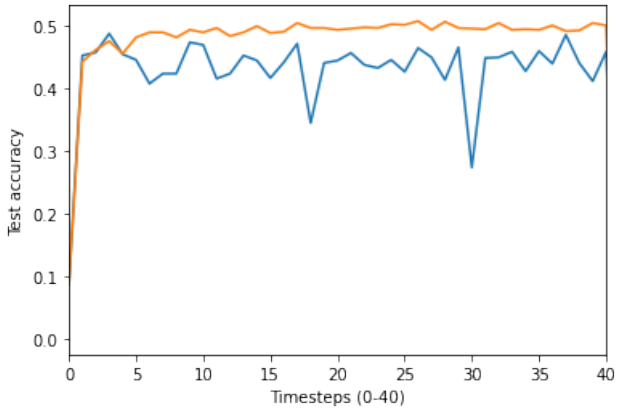
Our baseline is the performance of the first layer output of the layered model. This comparison allows us to show that the convex architecture improves when going from one to multiple layers.

We tuned the following hyperparameters in our model. learning rate, beta (regularization strength), rho, number of epochs and number of neurons, and optimization algorithm. Below are the values of these hyperparameters that obtained optimal results:

Layers	**Learning Rates**	L_2 **Normalization Weight** (β)	**Inequality Constraint Weight** (ρ)	**Epochs**	**Neurons**	**Optimization Algorithm**
2	1e-7 (First layer), 1e-6 (Second layer)	1e-4	1e-6	40	500	ADAM

The graphs that demonstrate these results are displayed below. Throughout the rest of the paper, we use the convention of plotting the first layer's results in blue and the second layer's results in orange. Note how the second layer's accuracies are much more stable over time compared to the first layer.





The confusion matrix and precision, recall, support values for our model are shown below:

Predicted class \ True class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	53	5	3	3	0	2	2	3	14	2
1	3	79	2	3	3	0	2	5	10	12
2	7	1	25	2	17	5	11	6	4	1
3	7	4	12	36	9	23	10	4	0	7
4	3	2	13	5	29	4	11	6	3	2
5	1	3	11	15	8	38	6	11	5	0
6	1	6	9	9	12	10	50	4	0	1
7	8	4	6	7	8	4	6	54	1	6
8	10	6	1	4	1	3	1	3	81	3
9	6	19	0	5	0	8	4	1	6	56

	precision	recall	f1-score	support
0	0.535	0.609	0.570	87
1	0.612	0.664	0.637	119
2	0.294	0.316	0.305	79
3	0.404	0.321	0.358	112
4	0.333	0.372	0.352	78
5	0.392	0.388	0.390	98
6	0.485	0.490	0.488	102
7	0.557	0.505	0.529	107
8	**0.653**	**0.717**	**0.684**	113
9	0.622	0.533	0.574	105
accuracy			0.501	1000
macro avg	0.489	0.492	0.489	1000
weighted avg	0.501	0.501	0.499	1000

Thus, we can see that the highest precision, recall and F1 score values are obtained by class 8 of the CIFAR-10 dataset, that is, ships. Thus, we can conclude that our model classifies ships most accurately.

Now, we will describe and analyze all the experiments we conducted in order to achieve the above results:

1. Pooling functions: As described in the Methods section, for the multiclass setting the unsummed output of our network is a three dimensional tensor with dimensions for number of examples, hidden dimension, and number of classes. In order to use this as input to the next layer, we have to eliminate the third dimension. We experimented with using max-pooling and average-pooling across the class dimension in order to down-sample the input before it enters the next layer. We found that both max-pooling and average-pooling resulted in good performance. Max-pooling achieved the optimal results described above, while average-pooling achieved training and test accuracies of 99.2% and 50% respectively, and training and test losses of 0.107 and 0.368 respectively.

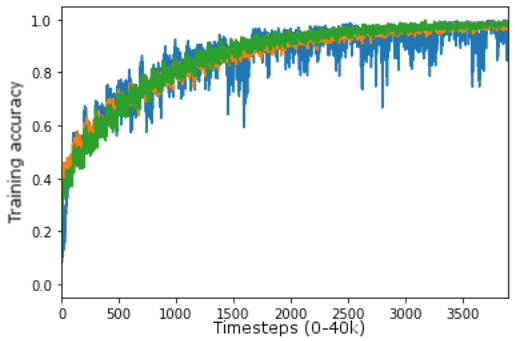
2. Initialization: We first used zero initialization (without column normalization) for our model's parameters at the beginning of both layers. However, with this setup, the model stopped learning at around 40 percent training accuracy. Thus, we then used Xavier initialization at the beginning of the first layer of the network - but this resulted in high training and test losses. After this, we tried using Xavier initialization at the start of the second layer of the model, without column normalization.

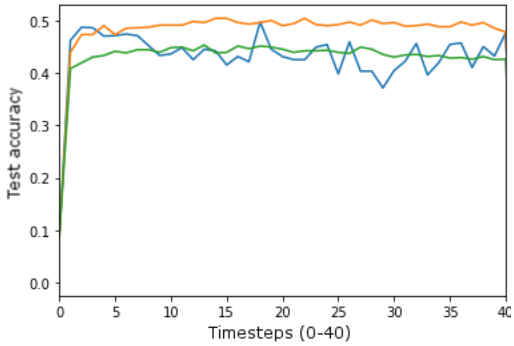
While this definitely improved training accuracy, which reached upto 60 percent, it then started to go down significantly. Next, we added column normalization to this setup. However, the losses became huge. Finally, we switched back to zero initialization with column normalization, and achieved the most optimal results (reported above). The performance of the model without column normalization can be seen below. A plot showing the performance of the model with zero initialization on the first layer and Xavier initialization on the second layer can be found in the appendix.

3. Normalization: In order to mitigate overfitting, we tried using Batch normalization, but it made our model's performance suffer, making us realize that convex learning will be at a disadvantage with a highly variable technique such as batch normalization, that must keep a track and alter running means and variances often. However, in order to normalize data in between layers, we used the much more dependable (less variable) technique of column normalization between layers. This significantly improved performance. A plot showing the performance of models without column normalization can be found in the appendix.

Column normalization was critical to making layerwise learning work in the convex domain. Column normalization's centering allows the stacked layers to distinguish between differences in activation relative to a neuron's typical activation. This removes an obstacle to a stacked layer's learning to "interpret" the neuron's activation profile. Column normalization's scaling effect levels the playing field of neurons' contributions to the next layer's inputs. This allows all neuron's to be treated equally when learning, which improves the probability that the layer will learn the correct neurons to rely on.

4. Adding layers: We tried to improve the model's performance by adding a third layer to the network. We observed that the best training and test accuracies obtained were lower than what the two-layer architecture accomplished. The plots for the 3 layer training and test accuracies can be seen below. The third layer's results are shown in green, and the first and second layers in blue and orange as before.





We believe that the third layer performs worse than the second layer because going from two layers to three in the convex domain corresponds to going from four layers to six in the non-convex domain. Thus, the third layer suffers from the same diminishing returns we expect from stacking fully-connected layers too deeply.

6 Conclusion/Future Work

We found that greedy layerwise training of networks composed of convex duals of two-layer fully-connected ReLU networks was able to decrease output variance. Stacked layers also experienced much more stable training and overall smoother convergence. Column normalization of prior layer outputs was found to be critical to successful layerwise training. Particulars of architecture choice, such as choice of pooling function, were found to have less effect on performance. Future work should likely focus on reducing the sizable remaining variance in the model through other techniques. Further, this approach could be applied to rapidly deploy models for image classification tasks to demonstrate its benefits in speeding up development cycles on time-sensitive tasks. If this approach can scale satisfactorily, it will have the benefits of speeding up deployment timelines on time-sensitive tasks, reducing the environmental footprint of training deep networks, and democratizing deep learning for those who would not otherwise have the computational resources to train deep networks.

7 Contributions

All written and coding work was done together.We worked on this project with the guidance of our mentor, Tolga Ergen (ergen@stanford.edu), affiliated with the Stanford University Department of Electrical Engineering.

References

[1] Raman Arora et al. "Understanding deep neural networks with rectified linear units". In: *arXiv preprint arXiv:1611.01491* (2016).

[2] Francis Bach. "Breaking the curse of dimensionality with convex neural networks". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 629–681.

[3] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. "Greedy layerwise learning can scale to imagenet". In: *International conference on machine learning*. PMLR. 2019, pp. 583–593.

[4] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. "Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods". In: *arXiv preprint arXiv:1506.08473* (2015).

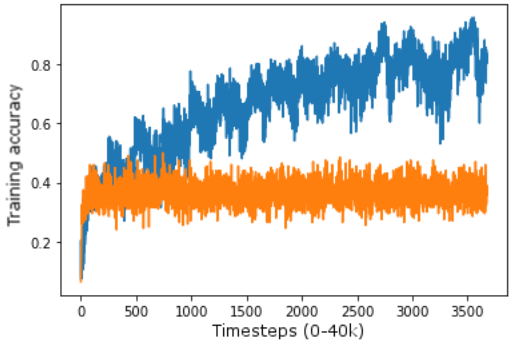
[5] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[6] Behnam Neyshabur. "Towards learning convolutions from scratch". In: *arXiv preprint arXiv:2007.13657* (2020).

[7] Mert Pilanci and Tolga Ergen. "Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7695–7705.

Appendix

The following plot depicts a network trained without column norm and with zero initialization for both layers.



The following plot depicts a network trained without column norm, with zero initialization for the first layer and Xavier initialization for the second layer.