Peeking into the Black Box: Layer-wise Training of Convex Convolutional Neural Networks

Trenton Chang (EE364b)
Stanford University
tchang97@stanford.edu

Raymond Ye Lee
Stanford University
ryelee@stanford.edu

1. Introduction

Supervised deep learning has achieved promising results in image classification (Tan and Le, 2021), object detection, and semantic segmentation, enabling a plethora of downstream applications. Many of these tasks rely on convolutional neural network (CNNs) architectures for feature extraction. Despite these successes, the optimization of deep convolutional neural networks remains poorly understood: how these networks do so well remains a black box.

In contrast to traditional optimization techniques focus on convex problem—which often have provable convergence guarantees or even polynomial-time solutions—deep neural networks are highly non-convex. Deep neural networks are generally constructed by alternating affine transformations with nonlinearities, then interspersing normalization (Ba et al., 2016; Ioffe and Szegedy, 2015), skip/residual-connections (He et al., 2015; Huang et al., 2018), or pooling operators (LeCun et al., 1989; Lecun et al., 1998a) across layers. With layer counts in the hundreds, it is non-trivial to reason rigorously about the dynamics of deep neural networks. This means that neural network optimization often relies on experimentation and heuristics with respect to hyperparameter and architectural choices, which can be time-consuming and come with few rigorous guarantees. This motivates the exploration of more principled approaches for optimizing deep neural networks.

Recent work has used convex optimization principles to enable a more theoretically-grounded understanding of convolutional neural network optimization, including subnetwork training (as in dropout) (Chen and Wang, 2014), convex relaxations of filters (Zhang et al., 2017), and an exact convex formulation (Ergen and Pilanci, 2021). These insights leverage component-wise or layer-wise convexity structures inside deep neural networks. In particular, we build on the work of Ergen and Pilanci (2021), which provides invaluable insight for two- and three-layer convolutional architectures. However, whether this formulation can enable higher-performance deep networks remains an open question.

We take an empirical approach to scaling up convex optimization of convolutional neural networks to deep architectures. Our project seeks to 1) validate the convex formulation of CNNs in Ergen and Pilanci (2021) for deep CNNs via greedy layer-wise training (Belilovsky et al., 2019), and 2) qualitatively understand optimal solutions generated by layer-wise convex formulations of CNNs through filter visualization. In this milestone report, we introduce preliminaries and our methods, then overview progress on our layer-wise training experiments and filter visualization pipeline.

2. Problem Statement

As our project centers around a first attempt at greedy layer-wise training of shallow convex networks, we consider in the first thrust of this project the C-class classification tasks posed by benchmark datasets like CIFAR-10 (Krizhevsky, 2009) and Imagenet (Deng et al., 2009). Specifically, we are interested in (1) the trajectory of training loss as a way of understanding how training dynamics might compare against classical methods, as well as (2) training and test accuracy to get a sense of whether such an approach might be effective in practice. In the second thrust of this project, we seek to investigate how the image features learned by this network formulation might differ from classical formulations. To this end, we visualize and compare the weights learned by both formulations.

In this milestone in particular, we present preliminary results for shallow networks using CIFAR-10 and show that in our implementations, the convex formulation indeed has more favorable training dynamics as well as performance on both training and test sets. We also show that the filters learned by either network are markedly different in natural ways, and provides insight into why the convex network formulation may be effective beyond those provided by inspection of the optimization problem.

3. Technical Approach

Let $\mathcal{X}, \mathcal{Y}$ be the input manifold and vector-valued label space. Following the empirical risk minimization framework (Vapnik, 1991), we learn a predictor function parameterized by $\theta \in \Theta$ defined by $f_\theta : \mathcal{X} \to \mathcal{Y}$ such that $\mathbb{E}_{\theta\sim\Theta,(x,y)\in\mathcal{S}}[\ell(f_\theta(x), y)]$ is minimized over a training set $\mathcal{S} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, where ℓ is convex. Following Ergen and Pilanci (2021), we take ℓ to be the mean-squared loss function.

For our learner function f_θ, we use the convex formulation of a two-layer CNN as detailed in Ergen and Pilanci (2021) with the following layers: Conv2D-ReLU-GlobalAveragePool-Linear. At a high level, a convolutional neural network works by applying filters of area h to image patches of area h via element-wise multiplication; the output is constructed by aggregating the results of the patch-filter multiplications over all patches and filters followed by a ReLU. The average pool operation averages over all spatial dimensions. Decomposing the parameters θ layer-wise, we let $\theta := \{\mathbf{u}_1, \ldots, \mathbf{u}_m, \mathbf{w}\}$ denote the m filter and linear layer weights, respectively. Representing $x \in \mathcal{X}$ as an array of K *patches* $\{X_1, X_2, \ldots, X_K\}$ each of size h (i.e. inputs to each filter), the forward pass of the model is

$$f(x) = \sum_{j=1}^{m} w_j \sum_{k=1}^{K} (X_k \mathbf{u}_j)_+. \tag{1}$$

This outputs a scalar-valued "score," which is generalized to multi-class classification by creating C independent copies of θ and summing loss across classes. Class predictions are obtained by taking the argmax over class scores. The primal objective, which is non-convex, can be written as

$$\text{minimize} \quad \frac{1}{2}\|f(x) - y\|_2^2 + \frac{\beta}{2}\sum_{i=1}^{m}(\|\mathbf{u}_i\|_2^2 + w_i^2) \tag{2}$$

which is provably equivalent by Ergen and Pilanci (2021) to the convex problem

$$\text{minimize} \quad \frac{1}{2}\left\|\sum_{i=1}^{P_{conv}}\sum_{k=1}^{K}\mathbf{D}(S_i^k)X_k\mathbf{v}_i - y\right\|_2^2 + \frac{\beta}{2}\sum_{i=1}^{m}(\|\mathbf{v}_i\|_2^2) \tag{3}$$

$$\text{subject to} \quad (\mathbf{I}_n - 2\mathbf{D}(S_i^k))X_k\mathbf{v}_i \leq 0, \forall i, k$$

in variables $\mathbf{v}_i$, a reparameterization of θ. Note that $\mathbf{D}(S_i^k)$ represents a diagonal sign matrix.[1]

While for this milestone we provide results for a single Conv2D-ReLU-GlobalAveragePool-Linear block, our ultimate goal is to construct layer-wise-trained networks wherein new layers are trained by freezing all past layers, creating a "frozen" neural network. We then use output of the last convolutional layer from the frozen network as input to the new layer, which is a Conv2D-ReLU-GlobalAveragePool-Linear block as before, adapting the strategy from Belilovsky et al. (2019).

[1]Formally: $\mathbf{D}$ is the diagonalization operator that converts a vector to a diagonal matrix with corresponding entires. Furthermore, $S \in \{\pm 1\}^h$, so $\mathbf{D}(S)$ is a diagonal matrix with ± 1 entries on the main diagonal.

4. Dataset

As noted previously, we use the CIFAR-10 dataset (Krizhevsky, 2009) for initial tests, which has dimensionality $32 \times 32 \times 3$. It contains 10 classes with $6,000$ images per class, of which $5,000$ are considered training and the remaining considered test. We will evaluate our model based on two considerations: the trajectory of training loss versus the analogous non-convex formulation, and accuracy, as measured by classification performance, and which allows us to easily make broader comparisons to other models. In particular, potential "models to beat" include AlexNet (Krizhevsky et al., 2012) and VGG-11 (Simonyan and Zisserman, 2015), which achieve accuracies of 89.0% and 92.5%, respectively, on CIFAR-10 using end-to-end training.

5. Results

In our experiments, we use stochastic gradient descent with momentum ($\beta = 0.9$) to optimize both the primal (non-convex) and convex objectives above for identical-sized two-layer CNNs. For both, we train for 100 epochs with batch sizes of 1000, $1,024$ filters of size 6×6 each, a stride of 4, and a learning rate of 0.0001, as suggested by Tolga Ergen.

5.1. Performance

As noted previously, we are especially interested in the training loss, and training and test accuracies for the convex formulation using the non-convex formulation as a baseline. Our results are presented in Table 1.

Metric	Convex	Non-convex
Loss (training)	0.3469	0.3735
Accuracy (training)	0.520	0.478
Accuacy (test)	0.5036	0.4474

Table 1: Training loss, and training and test accuracies between convex and non-convex network formulations.

We see that the convex formulation achieves a lower training loss than the non-convex formulation (0.3469 for the convex formulation versus 0.3735 for the non-convex formulation). Correspondingly, the accuracy achieved by

the convex formulation (0.520) on the training set is also higher than that of the non-convex formulation (0.478). Though these more favorable results on the training set might reflect overfitting, accuracy results on the test set suggest otherwise; the convex formulation outperforms the non-convex formulation by a handy 5.62 percentage points on the test set.

The speed with which the convex formulation converges during training is also notable. Figures 1 and 2 reveal differences in the training dynamics. In particular, we see that the non-convex formulation never breaches 0.5 accuracy on the training set, and that the convex network took only a few hundred steps before breaching 0.4, whereas the primal formulation took over two thousand to achieve the same performance.

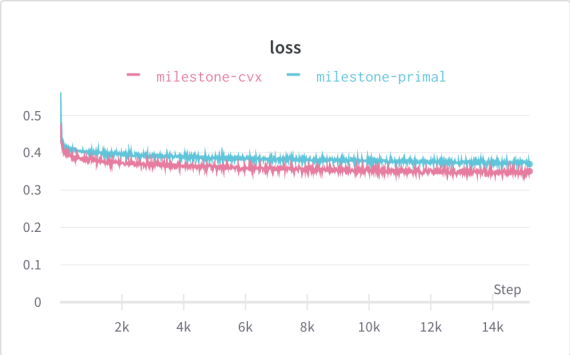


Figure 1: Training loss for primal (non-convex) and convex formulations.

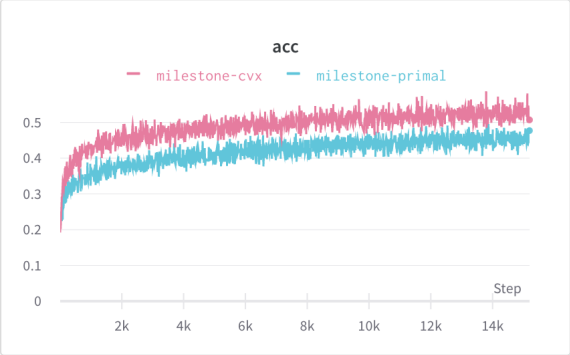


Figure 2: Training accuracy for primal (non-convex) and convex formulations.

5.2. Filter visualization

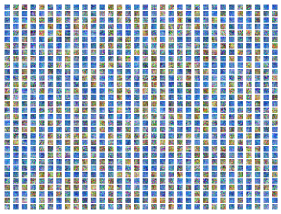
Filter visualization is a common technique for visualizing learned features of a CNN. To visualize filters, we find the image that maximizes the activations of that filter. Formally, let $\mathcal{K}_{i,j}$ be the function mapping image input $\mathbf{X}$ to the ith filter output at layer j. We have

$$\underset{\mathbf{X}}{\text{maximize}} \quad \mathcal{K}_{i,j}(\mathbf{X}) \quad \text{subject to} \quad \rho_{\min}\mathbf{1}_n \preceq \mathbf{X} \preceq \rho_{\max}\mathbf{1}_n \tag{4}$$

where $\rho_{\min}, \rho_{\max}$ are the minimum and maximum allowed pixel values, $\mathbf{1}_n$ is the all-ones matrix, and the constraint is elementwise. Though $\mathcal{K}_{i,j}$ is generally non-convex, this is readily solved using projected gradient descent.

For simplicity, we display filter weights directly for the two-layer CNNs. These can be thought of as visually similar to the gradient-ascent based method for the first convolutional layer.[2] Note that the convex-formulation network as detailed in Ergen and Pilanci (2021) has class-separable weights, so that we have $1,024$ filters for each of the 10 classes in the CIFAR-10 dataset in the convex model (and $1,024$ in total for the non-convex model. Preliminary results in Figure 3 show that the convex-formulation network appears to be indexing on color features for classification.

(a) Filter weights, non-convex model.



(b) Filter weights, convex model, class 0 (Airplane).

Figure 3: Filter weights

For example, we see that for the example of class 0 (Airplane), the class-specific filters in the convex model are dominated by blue colors, which likely corresponds to the fact that many training examples in the CIFAR-10 airplane class feature clear sky backgrounds. Similar color patterns are visible for the filter weights corresponding to the other classes (omitted for length). The weight visualization for the non-convex model is not very interpretable, as no clear patterns have emerged; since the non-convex model does not have class-separable weights, this is unsurprising.

6. Next steps

While this milestone shows heartening results produced by our implementation of the shallow convex network, we

[2]To see this; note that in the first convolutional layer, the maximizing patch under convolution must be a multiple of $\mathbf{u}_j$. We can summarily reconstruct the maximal-activation image from the relevant input patches. Deeper layers require the gradient ascent-based method to recover the maximizing input as outlined above.

still need to polish our implementation of layer-wise training. In particular, our goal is to scale our implementation of these shallow convex network via greedy layer-wise training to match the depth of well-studied deep architectures like the VGG family of models. We also plan to expand on our visualization analysis by incorporating newer methods, including more advanced methods like occlusion mapping (Zeiler and Fergus, 2013) or GradCAM (Selvaraju et al., 2019). Finally, having shown promising results on CIFAR-10, we are in the process of building our pipeline to also produce classification results for other benchmarks like MNIST (Lecun et al., 1998b) and ImageNet (Deng et al., 2009).

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Belilovsky, E., Eickenberg, M., and Oyallon, E. (2019). Greedy Layerwise Learning Can Scale to ImageNet. *arXiv:1812.11446 [cs, stat]*.

Chen, S. and Wang, Y. (2014). Convolutional neural network and convex optimization. *Dept. of Elect. and Comput. Eng., Univ. of California at San Diego, San Diego, CA, USA, Tech. Rep*.

Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Ergen, T. and Pilanci, M. (2021). Implicit Convex Regularizers of CNN Architectures: Convex Optimization of Two- and Three-Layer Networks in Polynomial Time. *arXiv:2006.14798 [cs, stat]*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2018). Densely connected convolutional networks.

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

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. page 60.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. In *NIPS*.

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

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

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2019). Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*.

Tan, M. and Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training.

Vapnik, V. (1991). Principles of risk minimization for learning theory. In *NIPS*.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks.

Zhang, Y., Liang, P., and Wainwright, M. J. (2017). Convexified convolutional neural networks. In *International Conference on Machine Learning*, pages 4044–4053. PMLR.