Convexifying Neural Networks

Aaron Luo
ECE273
UCSD
aaluo@ucsd.edu

***Abstract*—This project seeks to introduce theoretical guarantees into the field of neural networks by demonstrate the equivalency of solving a non-convex objective function and its corresponding convex formulation. In situations of weak duality, it looks to evaluate the performance differences between the two methods, and to show the advantages of using convex solvers over traditional backpropagation methods.**

I. Introduction

Despite the popularity of Deep Neural Networks (DNNs), understanding of their performance is hampered by its lack of theoretical guarantees. In this project, we introduce theoretical optimality on non-convex minimization problems using convex duality, which will have the same solution as the original non-convex problem due to its strong duality. We verify this claim empirically by solving the non-convex problem using a two-layer neural network with ReLU activation and Stochastic Gradient Descent (SGD). We then compare this result against the solution derived from solving the convex dual of the non-convex problem.

II. Background

This paper attempts to replicate the results established by Pilanci and Ergen (2020). Consider a two-layer neural network $f: \mathbb{R}^d \rightarrow \mathbb{R}$ with ReLU activation and m hidden neurons. This can be written as:

$$f(x) = \sum_{j=1}^{m} \phi\left(\mathrm{x}^{\mathrm{T}} u_j\right) \alpha_j \quad (1)$$

where $x \in \mathbb{R}^d$ is the input data, $u_j \in \mathbb{R}^d$ is the first layer of hidden weights, and $\alpha_j \in \mathbb{R}$ is the second layer of output weights. $\phi: \mathbb{R} \rightarrow \mathbb{R}$ represents the ReLU activation function, and can be written as:

$$\phi(t) = \max(0, t) \quad (2)$$

Now we want to minimize our error such that we can update our weights and improve the performance of our network on a given dataset. Given an input data matrix $X \in \mathbb{R}^{n \times d}$ and a binary label vector $y \in \mathbb{R}^n$, consider the minimization problem:

$$p^* = \min_{\{\alpha_j, u_j\}_{j=1}^m} \frac{1}{2} \left\| \sum_{j=1}^{m} \phi(X u_j) \alpha_j - y \right\|_2^2 + \frac{\beta}{2} \sum_{j=1}^{m} \left(\left\| u_j \right\|_2^2 + \alpha_j^2 \right) \quad (3)$$

where $\beta > 0$ is the regularization parameter. This operation adds the squared L2-norm of the network error function with the sum of squared L2 parameter norms, the latter of which functions as a reguarlization term. The optimization problem (3) is described as highly non-convex due to the non-linearity of its ReLU activation functions as well as the multiplication between the hidden weights μ_j and the output weights α_j. Convexifying this function involves finding its Lagrange dual, which is theoretically guaranteed to be convex regardless of the convexity of the original function. Furthermore, the solution of the primal is bounded by the solution to the dual by weak duality, i.e., $p^* \geq d^*$. Given strong duality, the solution to the dual is equivalent to the solution of the primal, i.e., $p^* = d^*$. This project takes advantage of these facts to develop an optimization method with stronger theoretical guarantees than gradient descent.

A. Deriving the Dual of a 2-Layer ReLU Network

We find the convex dual of the non-convex objective function (3) as derived in the reference paper (Pilanci & Ergen, 2020). Consider the L1-penalized representation of the objective function:

$$p^* = \min_{\substack{\|u\|_2 \leq 1 \\ \forall j \in [m]}} \min_{\{\alpha_j\}_{j=1}^m} \frac{1}{2} \left\| \sum_{j=1}^{m} \phi(X \mu_j) \alpha_j - y \right\|_2^2 + \beta \sum_{j=1}^{m} |\alpha_j| \quad (4)$$

This function is equivalent to the objective function (3). Finding the Lagrange dual of the inner minimization function gives:

$$p^* = \min_{\substack{\|u\|_2 \leq 1 \\ \forall j \in [m]}} \max_{\substack{v \in \mathbb{R}^n \ s.t. \\ |v^T \phi(X u_j)| \leq \beta, \forall j \in [m]}} -\frac{1}{2} \|y - v\|_2^2 + \frac{1}{2} \|y\|_2^2 \quad (5)$$

Using the principle of weak duality, we find the lower bound of the optimal:

$$p^* \geq d^* = \max_{\substack{v \in \mathbb{R}^n \ s.t. \\ |v^T \phi(X u)| \leq \beta, \forall u \in \mathcal{B}_2}} -\frac{1}{2} \|y - v\|_2^2 + \frac{1}{2} \|y\|_2^2 \quad (6)$$

where $\mathcal{B}_n \in \mathbb{R}^n$ is a unit ball function. This function is called a convex semi-infinite optimization problem with n variables and infinitely many constraints. It is shown in Pilanci and Ergen (2020) that if the number of hidden neurons m satisfies the condition $m \geq m^*$, where $m^* \in \mathbb{N}, 1 \leq m^* \leq n$, strong duality holds, and $p^* = d^*$.

B. The Finite Dimensional Convex Problem

We can convert the non-convex objective function to a finite dimensional convex program without using the Lagrange dual approach. Consider the non-convex ReLU activation function. Given that this function has the effect of introducing partitions in the parameter space by masking certain values that fall under a threshold, we can recreate this using convex linear methods. Let $D_1, \ldots, D_P$ be a set of diagonal matrices with binary entries that represent the set of all possible hyperplanes in $\mathbb{R}^d$ space that pass through the origin and linearly separate the points in the dataset X. Let P denote the number of unique partitions that are formed from these hyperplanes. We can bound the value of P by:

$$P \leq 2\sum_{k=0}^{r-1}\binom{n-1}{k} \leq 2r\left(\frac{e(n-1)}{r}\right)^r \quad (7)$$

where $r \leq n$ and r is the rank of the data matrix X. We can additionally represent diagonal matrices $D_i \in \{0,1\}^{n\times n}$ as $Diag(1[Xu_i \geq 0])$, where $1[\cdot] \in \{0,1\}^n$ is an indicator function and $u_i \in \mathbb{R}^d$ is arbitrary. Consider the convex problem:

$$\min_{\{v_i,w_i\}_{i=1}^P} \frac{1}{2}\left|\left|\sum_{i=1}^P D_iX(v_i - w_i) - y\right|\right|_2^2 + \beta\sum_{i=1}^P\left(||v_i||_2 + ||w_2||\right) \quad (8)$$

$$s.t. (2D_i - I_n)Xv_i \geq 0, (2D_i - I_n)Xw_i \geq 0, \forall i$$

The L1 group regularization term of this problem is convex and enforces the sparsity introduced by the ReLU activation function, which in problem (8) has been written as:

$$\phi(Xu_i) = D_iXu_i$$

The optimal solution to the non-convex problem (3) where $m \geq m^*$ and the convex problem (8) are identical (Pilanci & Ergen, 2020). The optimal solution (u_j^*, α_j^*) to the non-convex problem (3) can be written in terms of the optimal solution $\{v_i^*, w_i^*\}$ of the convex problem (8):

$$(u_{j1}^*, \alpha_{j1}^*) = \left(\frac{v_i^*}{\sqrt{||v_i^*||_2}}, \sqrt{||v_i^*||_2}\right) \qquad if \;\; v_i^* \neq 0$$

$$(u_{j2}^*, \alpha_{j2}^*) = \left(\frac{w_i^*}{\sqrt{||w_i^*||_2}}, -\sqrt{||w_i^*||_2}\right) \quad if \; w_i^* \neq 0$$

Using this relationship, we can deduce that $m^* = 2P$. Solving the formulation (8) can provide one globally optimal solution of the corresponding non-convex objective function. Other papers, namely Wang, et al (2022), suggest methods that are able to provide the set of all global optimums of the non-convex objective functions; however, we will be focusing on the single solution version for this project.

C. *Convex Optimization with Convolution Neural Networks*

For Convolutional Neural Networks (CNN), the input data matrix $X \in \mathbb{R}^{n\times d}$ can be partitioned into patch matrices $X_k \in \mathbb{R}^{n\times h}$, where $k = 1, \ldots, K$ and the rank of each patch matrix is very small. Consider the convolutional training problem:

$$\min_{\{\alpha_j,u_j\}_{j=1}^m} \frac{1}{2}\sum_{k=1}^K\left|\left|\sum_{j=1}^m \phi(X_ku_j)\alpha_j - y_k\right|\right|_2^2 + \frac{\beta}{2}\sum_{j=1}^m\left(||u_j||_2^2 + \alpha_j^2\right) \quad (9)$$

Generally, convex solvers perform worse on very high dimensional datasets. Due to the low rank of these patch matrices, CNNs can guarantee that feature complexity remains low, allowing for the convex program (9) to be solved in $O\left(d^3r^3\left(\frac{n}{r}\right)^{3r}\right)$, or polynomial time.

D. *Datasets*

For this project, most experimentation will be done using random generated data $X \sim N(0, I_d)$. Our final set of tests will be done using a small sample of the CIFAR-10 dataset [3], which is an image dataset consisting of 60,000 32x32 color images in 10 classes. For our purposes we will only be using two for binary classification.

III. Methods

A. *Training the Non-Convex Problem*

To solve the non-convex problem (3), we use Stochastic Gradient Descent (SGD) with a two-layer ReLU network, m hidden neurons and 200 iterations. The theoretical guarantee of this method is that it will converge to an optimal $p^* = d^*$, if $m \geq m^*$ and $1 \leq m^* \leq n$. The computational complexity of this method is $O(2^m n^{dm})$, since we will be using naïve backpropagation to update our weights.

For our first experiment, we solve the non-convex problem using a 5×3 randomly generated matrix and $m = 5$ hidden neurons. We then experiment with larger sizes of m, namely $m = 15, 44,$ and 50. $m = 44$ is chosen as P for a full rank rectangular matrix of the given dimensions is 22, as determined by Equation (7). Since $m^* = 2P$, we experiment with $m = 44$.

We repeat this experiment with a larger n, namely $n = 10$. Since $P = 92$, we scale m up accordingly to $m = 184$. The idea here is to test what happens when m is far greater than n but also greater than or equal to $2P$. We also try a some previously used values of $m = [5,15]$ so that we can make a direct comparison to the first experiment.

Out last experiment involves performing binary classification of the CIFAR-10 dataset. We sample approximately 5% of the dataset from each label group to train on, totaling to around 100 binarily labeled data. The data was then divided by 1000 to scale down values but keep proportional differences. This step was included as using lower values in the CVXPY convex solver seemed to dramatically improve performance. We additionally perform dimensionality reduction on the sampled data using PCA, as while gradient descent isn't as affected by datasets with high numbers of features, convex optimization methods see notably high impacts in performance when the dataset is too complex. In this case, we reduce the number of dimensions to 100. In terms of the number of hidden neurons m, we reference Pilanci and Ergen (2020) and set $m = 12$, as they also do in a similar experiment described in the paper.

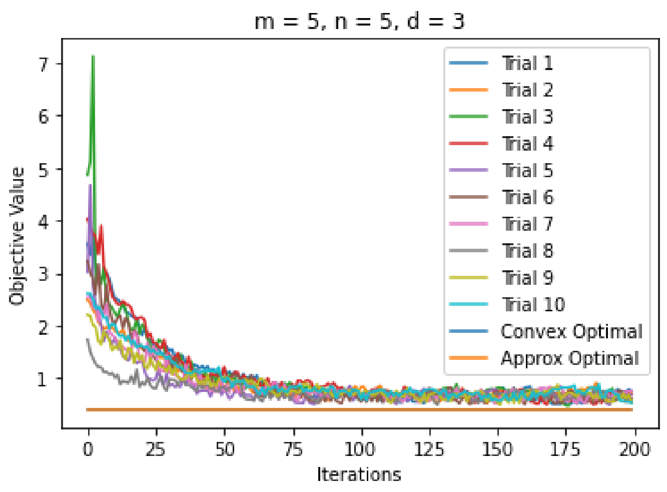
B. *Solving the Convex Problem*

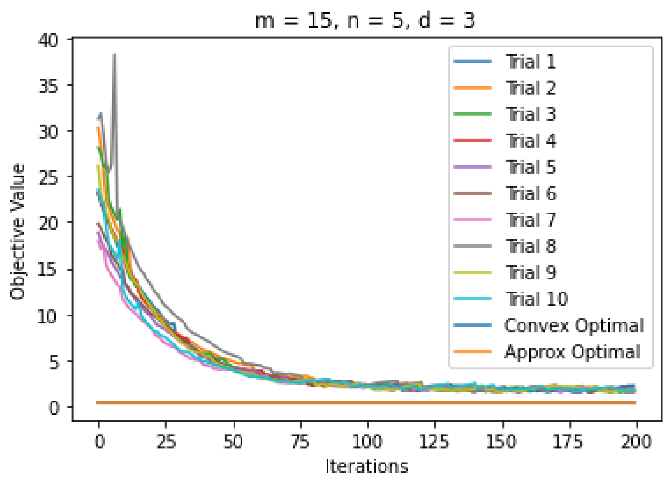
The convex problem (8) was used to perform our experiments. The Python library CVXPY was utilized to help solve our convex problem. Generating the diagonal matrices D_i is done in two ways. The first is an exact method, that is, the number of diagonal matrices produced are equal to P, the theoretical maximal number of partitions given a dataset X. To do this, we iterate through all possible combinations of datapoints, with each combination having a size range of $[2, r]$, where $r = rank(X)$. Then, each set of points is used to create a hyperplane that runs through all the chosen points and linearly divides the dataset into two groups. Since every hyperplane needs to pass through the origin, a range of $[1, r - 1]$ datapoints is chosen for each set. If a set does not have enough points to create a hyperplane, additional points that are not equal to any of the points in the original dataset are randomly sampled to compensate.

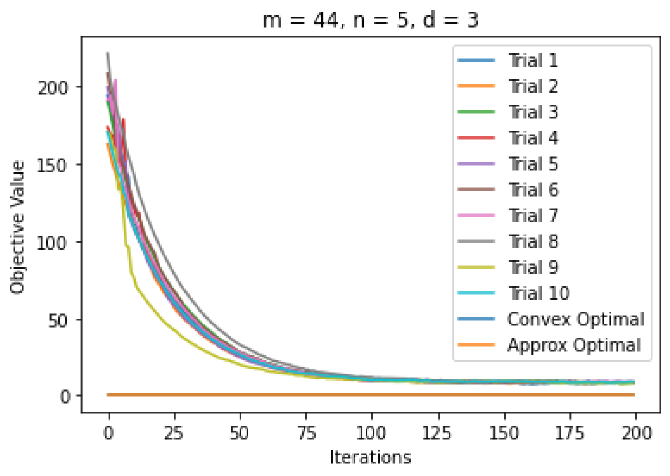
Note that each hyperplane created this way can generate up to four diagonal matrices D_i . Given the half space $\{x_i | Xu \leq 0\}$, we can generate two diagonal matrices by simply flipping the inequality. Removing the equality sign for each inequality generates an additional two diagonal matrices.

The second method we use to generate diagonal matrices is an approximate one. When P approaches extremely high values due to increases in data size or complexity, generating exactly P diagonal matrices become unreasonable and computationally inefficient. Using an approximal method allows the dimensions of X to increase while keeping computational costs reasonable. To generate an approximate number of diagonal matrices, $\bar{P}$ hyperplanes $u \sim N(0, I_d)$ are randomly sampled and used to create diagonal matrices. Using this method of approximation produces a convex optimum similar to the one that would be produced by method 1 (Pilanci & Ergen, 2020). For the purposes of this project, $\bar{P}$ is calculated using the expression: $\bar{P} = \min(P, 50)$.

IV. Results







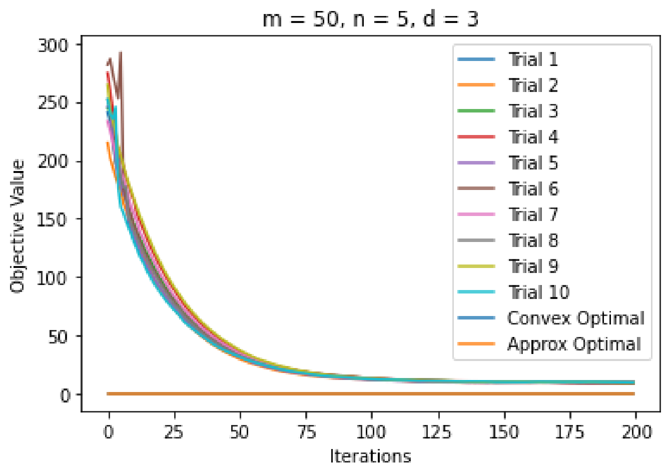


Figure 1: Experiment 1

Figure 1 contains the results of our first experiment, where we had a randomly generated input dataset of $X \in \mathbb{R}^{5 \times 3}$ and hidden neurons $m = [5, 15, 44, 50]$. Generally, SGD seems to converge towards the lower bound set by the convex program, but never seems to reach it. These results seem to corroborate the theoretical understanding of weak duality, that is $p^* \geq d^*$.

As the values of m increase, the higher the gradient seems to spike at the beginning of training, which results in slower convergence over time.

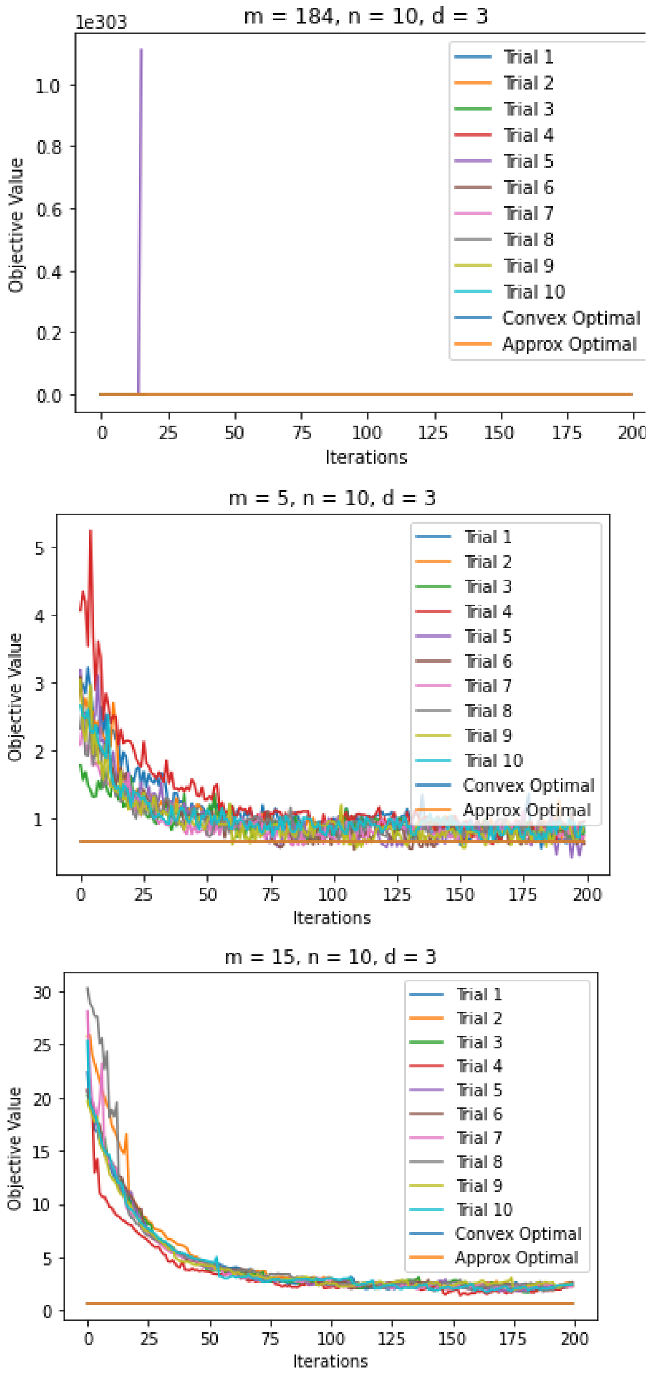


Figure 2: Experiment 2

Figure 2 shows the results of our second experiment, where we experimented with a larger value of $n = 10$. The first set of trials where $m = 2P = 184$ encountered the exploding gradient problem, which sent weight values to infinity. Lower m values did not appear to run into the same issue. When comparing these results where $m = [5,15]$, $n = 10$ to the earlier results where $n = 5$, it seems that they both convergence to some value close to but greater than the convex optimal. One noticeable difference is that the set with the larger sample size seem to have a somewhat noiser convergence than the set with the smaller sample size.

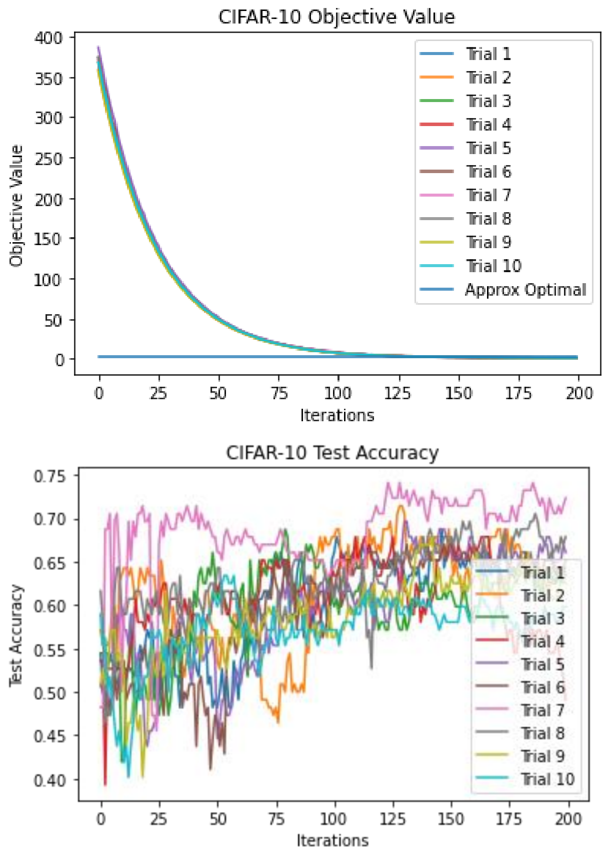


Figure 3: Experiment 3

Figure 3 shows the results of our last experiment, where we perform optimization on a small sample of the CIFAR-10 dataset. Like in our previous experiments, the non-convex problem seems to converge towards the convex lower bound over time. Also like in our previous experiments, the higher sample size of $n \approx 100$ spikes the gradient at the beginning of training. Each trial seems to have a very similar trajectory to each other, which could maybe be attributed to the higher complexity of the dataset. It could, however, also just be attributed to the fact that the graph contains a larger range of values.

It appears that our test accuracy seems to generally improve over time as well. The trajectories are very noisy, however, and possibly suggest some instability with the gradient. Some trajectories also seem to drop in accuracy towards the end, particularly Trial 4.

V. Discussion

From the results, one important thing to discuss is the theoretical understanding that $m \geq m^* = 2P$, along with the idea that $1 \leq m^* \leq n$. Both reference papers (Pilanci & Ergen, 2020; Wang et al, 2022), seem to corroborate this idea.

However, frequently in the datasets used in our experiments, these two expressions were contradictory. In fact, given an $n \times d$ matrix, these two expressions can only be true at the same time when $n \geq 2$ and $r = 1$, in which case $P = 1$. This conflict is somewhat supported by experimentation; when setting a value of $m > n$ such that $m \geq 2P$, there was a reasonable likelihood that the dataset would encounter the exploding gradient problem. In fact, this likelihood did not seem to be correlated with the difference between n and m. Increasing the value of m at all seemed to increase the probability of an exploding gradient along with it.

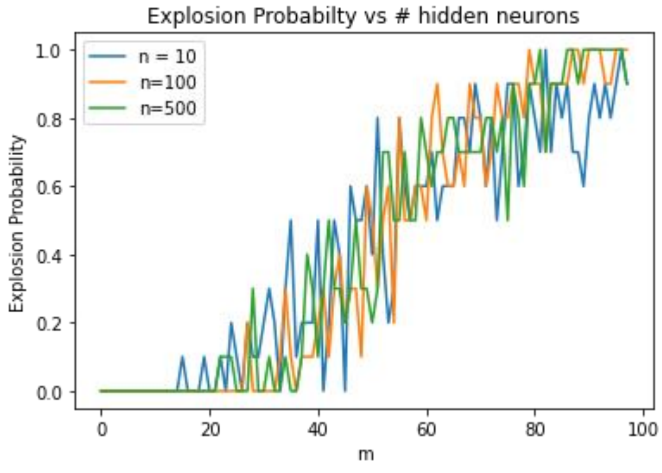


Figure 4

Figure 4 depicts the probability of gradient explosion as m approaches 100. In this experiment, $m \in [2,100)$, and is tested 10 times with new data and weight initialization at each iteration. The dataset X$\in \mathbb{R}^{n \times 3}$, and is randomly sampled from a normal distribution. The values of n are set to $n = [10,100,500]$. These values were chosen to test three things: 1. Explosion probability as m becomes much bigger than n over time, 2. Explosion probability as m converges to n, 3. Explosion probability when m is much lower than n. All three of these cases have a very similar probability trajectory, so it appears that the value of n doesn't affect the explosion probability.

The main purpose of this discussion is to point out that the theoretical optimal m seems to go to very high values as the size and rank of the dataset increases. However, the probability that the non-convex optimal goes to infinity seems to go up along with it. This, along with the fact that $2P \leq m^* \leq n$ most of the time seems to, at the very least, question our understanding of this paper. Further research will need to be done to answer this.

Another thing worth discussing is the behavior of the graphs in the CIFAR-10 experiment. First, we answer the question brought up in the results about the trajectory behavior of the non-convex optimization graph.

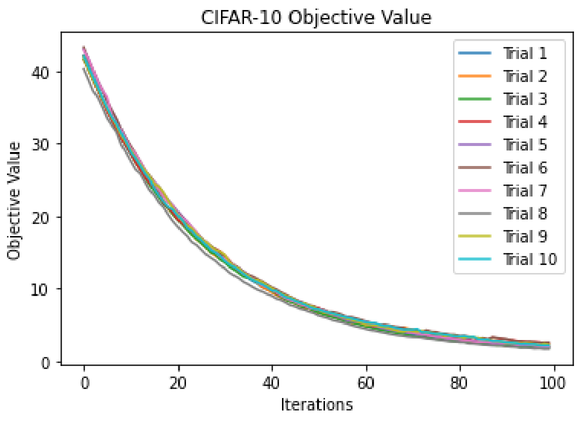


Figure 5: Zoomed In Objective Graph

Figure 5 is a subsection of the graph objective function graph over time in figure 3. All the trajectories appear to be very similar to each other, even after scaling down the graph. This could be attributed to the preprocessing of the input data. In the other experiments, all the data was randomly sampled from a normal distribution. In this third experiment, not only was the data not randomly sampled, it was also scaled down by a factor of 1000, making all the datapoints more similar to each other. Further experimentation would need to be done to verify if this is the case; unfortunately keeping the data at the same scale leads to very poor performance.

The second thing that should be discussed regarding the CIFAR dataset is the behavior of the test accuracy graph. In the results section, we can see that the test accuracy goes up as the objective function converges to an optimal. However, this isn't always the case

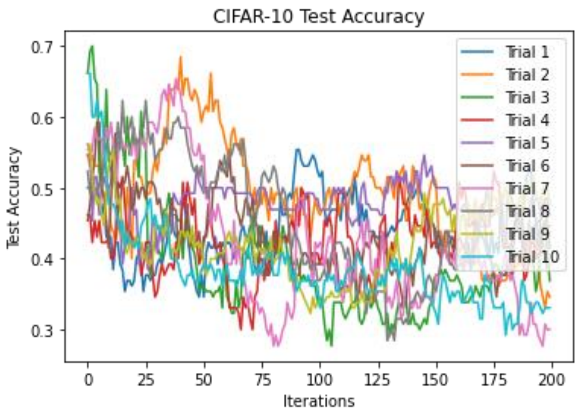


Figure 6: Downward Trending Accuracy

As can be seen in Figure 6, sometimes the accuracy goes down over time. This is technically not a bad thing, as since this is a binary classification problem, we can simply flip the graph over and the accuracy is now trending upwards. However, sometimes the test accuracy does not seem to move at all.

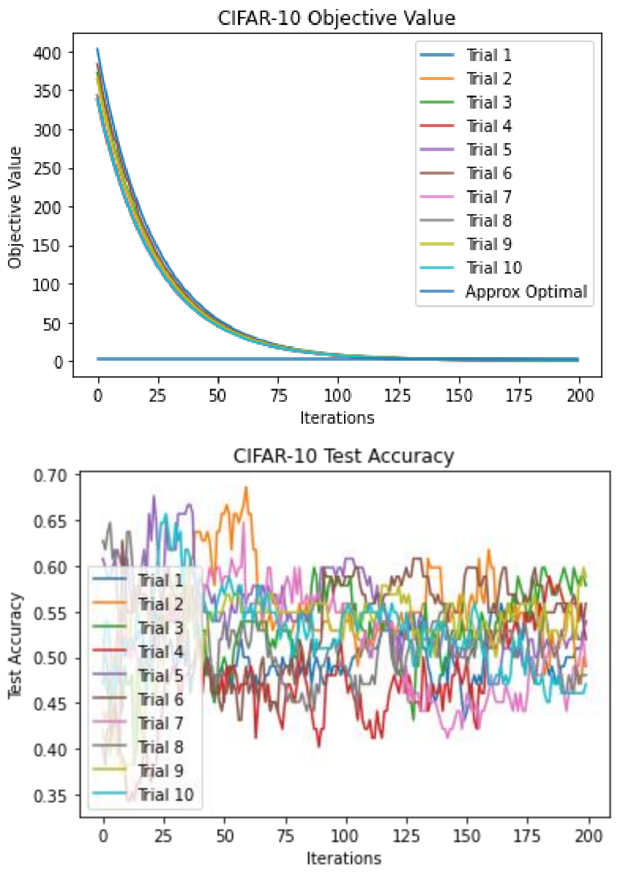


Figure 7: No Trending Testing Accuracy

As can be seen in Figure 7, this accuracy doesn’t seem to move at all, even though its corresponding objective function very clearly is able to converge towards the convex optimum. It is possible that the weights happened to be close to the optimal when it was initialized, so the accuracy doesn’t move much over time as it has already converged. However, it is uncertain robust this hypothesis is, given the steep convergence trajectory of the objective function over time. Further research would be needed to verify this phenomenon.

Overall, I learned that neural networks could be made convex along with when it is advantageous to do so. Based on the results of this project, it seems that gradient descent is better on large, highly complex datasets performance-wise. While using convex solvers seems to provide a better optimal on all datasets, it seemed to take much longer to reach said optimal when compared to SGD when used on the larger datasets. I would, however, like to look further into this, as my experiences with this project could potentially be attributed to the library used. Maybe I could write a more efficient version. Overall, optimization of the convex method seems like it would be worth experimenting further with.

VI. References

1. Pilanci, M. and Ergen, T. (2020).
Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks.
2. Wang, Y., Lacotte, J., and Pilanci, M. (2021).
The hidden convex optimization landscape of regularized two-layer
relu networks: an exact characterization of optimal solutions.
3. Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
http://www.cs.toronto.edu/~kriz/cifar.html