Convexifying Neural Networks

Sreevatsank Kadaveru
Electrical Computer Engineering
University of California San Diego
skadaveru@ucsd.edu

I. Introduction

The fascinating possibility of modelling the highly non-convex problem of training a neural networks as a convex formulation is explored in [1]. In this work the authors develop a equivalent convex formulation for the non-convex two-layer rectified linear unit (ReLU) neural network training problem and demonstrated that the solution to the convex problem can us used to construct the optimal neural network. The authors further provided bounds on the computational complexity of solving the convex equivalent. [2] extends this work further by leveraging the convex equivalent to obtain the set of all optimal neural networks. They also characterise the invariance properties of the solution set.

This formulation transforms the intractable highly non-convex neural network training problem into an analytical and tractable form. This enables analytical treatment of the problem yielding valuable insights into the functionality if the neural networks and their unreasonable modelling efficiency. Importantly, unlike other recent works that approximate the finite neural network with an infinite width neural network and make connections to kernel methods, this work attempts to analyse the finite dimensional networks as is.

A. Problem Setup

Consider a two-layer ReLU neural network. $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ with m neurons. Let $X \in \mathbb{R}^{n \cdot d}$ be the input training data and $y \in \mathbb{R}^n$ be the label vector, and for analytical simplicity consider the square loss objective and an l_2 norm regulariser.

$$L_\beta = \frac{1}{2} * \left|\left| \sum_{i=1}^{m} (Xu_i)_+ \alpha_i - y \right|\right|_2^2 + \frac{\beta}{2} \sum_{i=1}^{m} (||u_i||_2^2 + \alpha_i^2) \quad (1)$$

$$p^* = \min_{\{\alpha_i, u_i\}_{i=1}^m} L_\beta \quad (2)$$

Eq 2 is the objective function used to train the neural network. The objective function is highly non-convex due to the use of the non-linear activation functions (ReLU here) and parameter multiplication. l_2 regularisation is use here as it is convex. l_2 regulariser in practice and is used to prevent over fitting in neural networks.

II. Convex Equivalent

The authors in [1] using the bidual of 2 show that there exists a m^* such that for $m \geq m^*$ strong duality holds.

Further, the authors show that the solution to Eq 3 subject to 4 constrains can be used to construct a solution to the original non-convex problem 2. Here P is the number of partitions in the $\mathbb{R}^d$ space created by the $x_i^T z = 0$ hyper planes, where $X = [x_i^T]$. The D_i matrix is defined as $D_i = diag[(Xu_i)_+]$.

$$\min_{\{\alpha_i, u_i\}_{i=1}^m} \frac{1}{2} * \left|\left| \sum_{i=1}^{P} D_i X (v_i - w_i) - y \right|\right|_2^2 + \beta \sum_{i=1}^{P} (||v_i||_2 + ||w_i||_2) \quad (3)$$

Given,

$$(2D_i - I_n) X v_i \geq 0, (2D_i - I_n) X w_i \geq 0, \quad (4)$$

The non-convex problem in eq 2 is converted to the convex problem 3 by exploiting the the piece-wise linearity of the ReLU activation function. As the ReLU is linear for positive operands, we decompose the non-linear ReLU operation into the sum of two linear regions of operation. This is achieved by multiplying the sign pattern/partition matrix D_i.

Each partition $C_i = \{u | diag[(Xu)_+] = D_i\}$ is a cone, and given the problem constraints 4 formulation each of the v_i are restricted to their respective cone C_i. The problem 3 is over parameterised with v_i and $-w_i$ to account for the passivity and negativity of the objective in the partition (accounts for the sign of α_i).

A. Solving the Convex Problem

The above problem 3 is a second order cone program and can be solved using standard interior point solvers. In this report we make use of the CVXPY framework [3] and the ECOS solver [4].

The solution to the convex problem is guaranteed to be a global minima and the minimiser is optimal weight vector of the neural network. This guarantee only holds for $m \geq m^*$.

The computational complexity is proportional to the number of variables and the constraints in the convex optimisation program, and we have $2dP$ variables and $2nP$ linear constraints. The worst case complexity is given by $O(d^3 r^3 (\frac{n}{r})^{3r}$. Here $r = Rank(X)$ which is of the order of d. We observe that the complexity is polynomial in n but is exponential in the rank r.

III. Partitions

P is the number of partitions in the $\mathbb{R}^d$ space created by the $x_i^T z = 0$ hyper planes, where $X = [x_i^T]$. The D_i matrix is defined as $D_i = diag[(Xu_i)_+]$. Each partition $C_i = \{u | diag[(Xu)_+] = D_i\}$ is a convex cone, and represents the region of operation where all the ReLU units are in their linear operating region (i.e. in y=0 state or y=x).

The D_i matrices are linear elements that perform row selection operation similar to the of ReLU, they emulate the nonlinear behaviour of ReLU. D_i matrices are fixed matrices and are linear and eliminate the need for the non linearity.

IV. Extension to Convolutional Neural Networks (CNNs)

The main results above can be directly extended to CNNs by interpreting the m as the number of filters in the CNN and d being the filter dimension.

Consider the $X_k \in \mathbb{R}^{n \cdot d}$ be the patch matrix, where $k \in [1, K]$. The objective function becomes 6

$$L_\beta = \frac{1}{2} * \left|\left| \sum_{i=1}^{m} \sum_{k=1}^{K} (X_k u_i)_+ \alpha_i k - y \right|\right|_2^2 + \frac{\beta}{2} \sum_{i=1}^{m} (||u_i||_2^2 + \alpha_i^2) \tag{5}$$

$$p^* = \min_{\{\alpha_i, u_i\}_{i=1}^m} L_\beta \tag{6}$$

Assuming a linear activation function, the equivalent convex objective becomes 7.

$$\min_{z_k \in \mathbb{R}^d} \frac{1}{2} * \left|\left| \sum_{k=1}^{K} X_k z_k - y \right|\right|_2^2 + \beta \, ||Z||_* \tag{7}$$

Where $||Z||_*$ is the nuclear norm.

V. Additional Guarantees provided by [2]

In [2], the authors demonstrate that by the process of splitting they are able generate the entire solution set of optimum neural networks beginning from a single optimal solution. This is an extension of the work in [1] where only one optimal solution could be constructed.

The splitting of a optimal neuron w^* is performed by constructing (u_j, α_j) as below

$$u_j = \sqrt{\gamma_j} \frac{w^*}{\sqrt{||w^*||_2}} \alpha_j \qquad = \sqrt{\gamma_j} \sqrt{||w^*||_2} \tag{8}$$

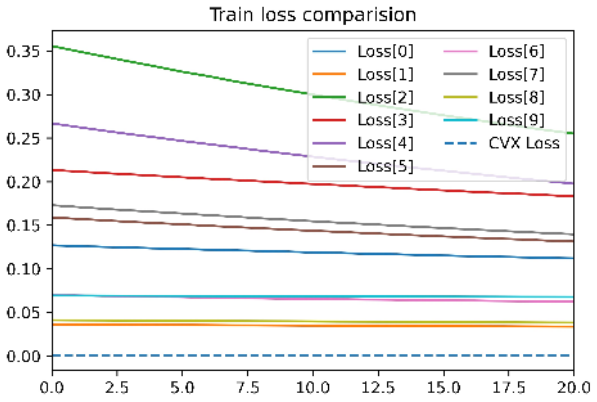
where $\gamma_j \geq 0$ and
$gamma_j \in \mathbb{R}$, and $\sum \gamma_j = 1$.

VI. Experiments

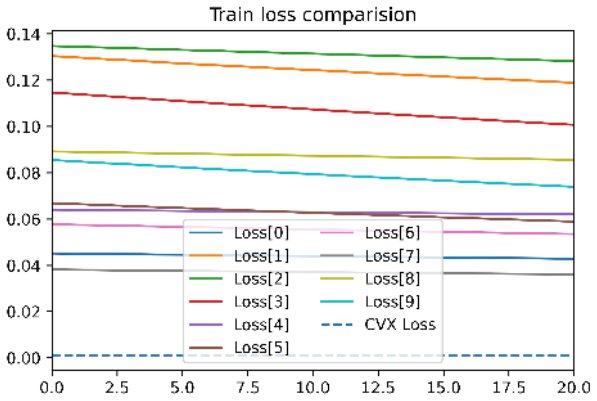
A. Comparison between CVX solution and Stochastic Gradient Descent, for Small N

B. Comparison between CVX solution and Stochastic Gradient Descent, for Large N

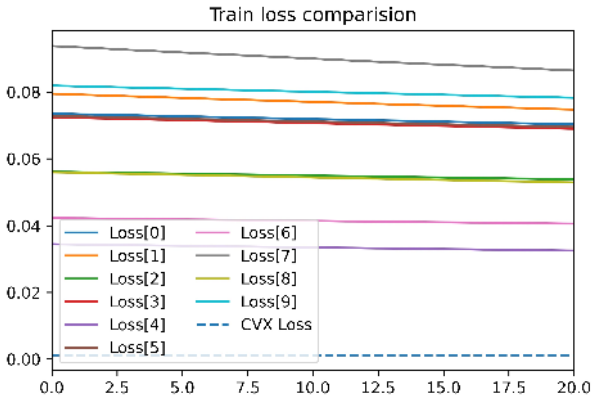
Figure 1, 2, and Figure 3, 4 compare the training and validation loss between different realisation of SDG and the convex solution for small N and large N.



(a) m=5

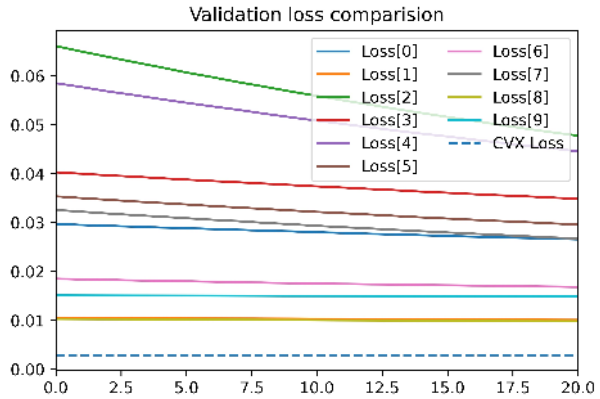


(b) m=15

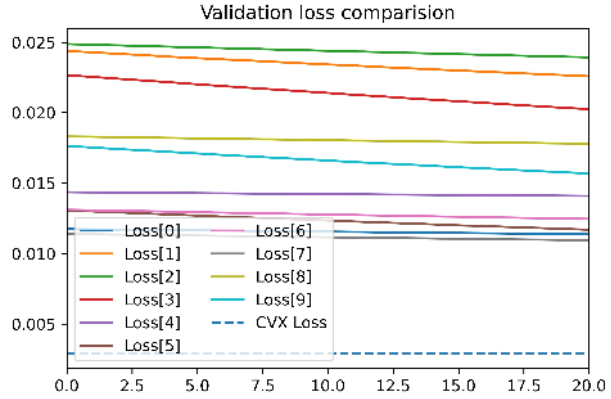


(c) m=50

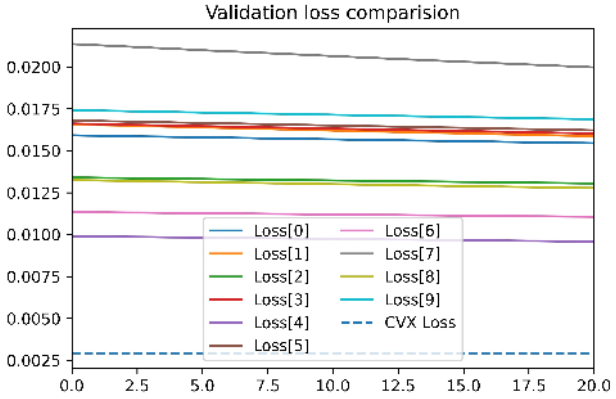
Fig. 1: Training Loss, N=10,m={5,15,50}



(a) m=5

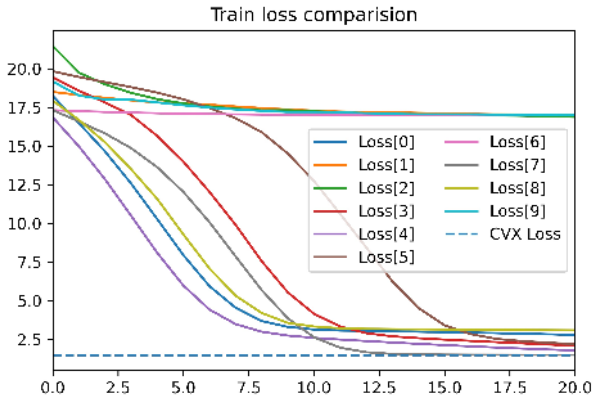


(b) m=15

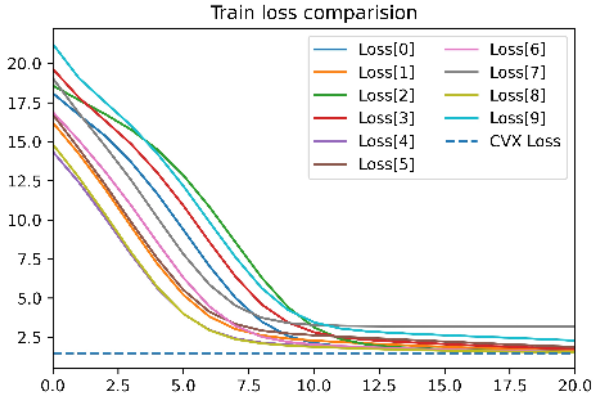


(c) m=50

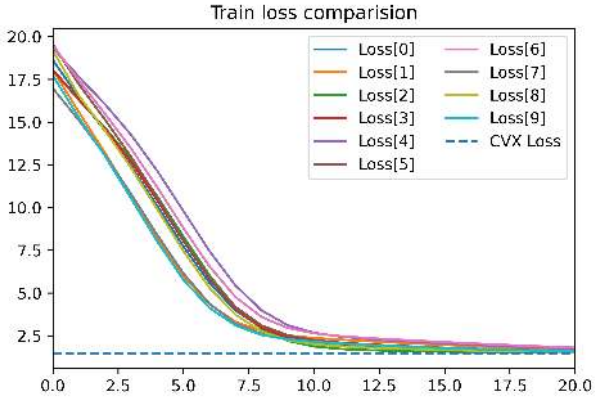
Fig. 2: Validation Loss, N=10,m={5,15,50}



(a) m=5

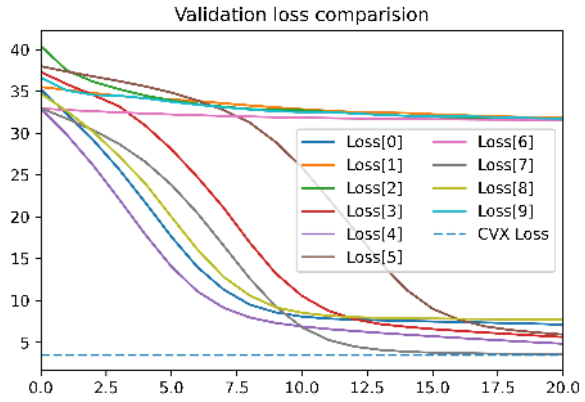


(b) m=15

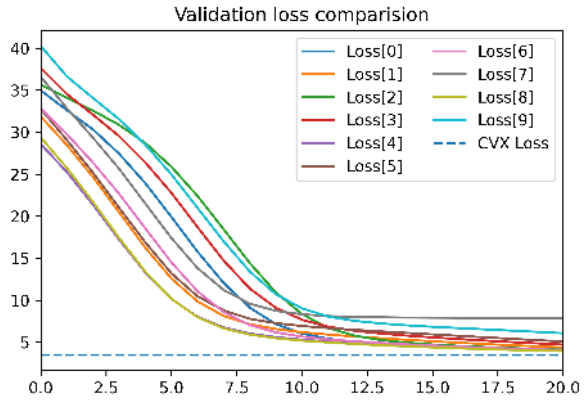


(c) m=50

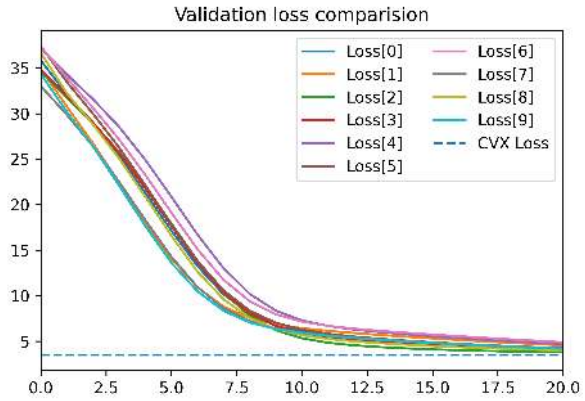
Fig. 3: Training Loss, N=100,m={5,15,50}



(a) m=5

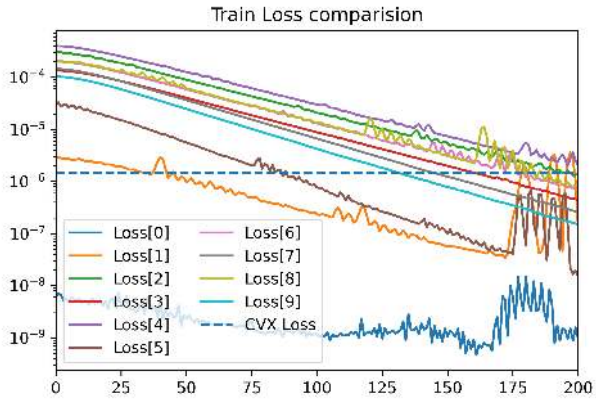


(b) m=15

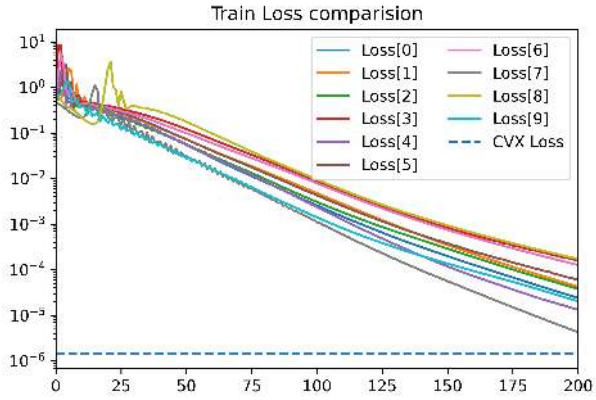


(c) m=50

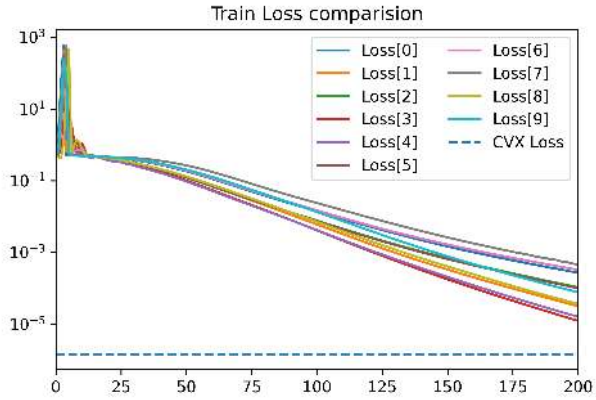
Fig. 4: Validation Loss, N=100,m={5,15,50}



(a) m=50



(b) m=500

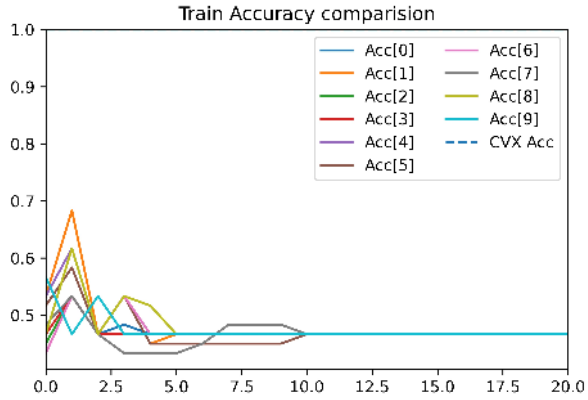


(c) m=50000

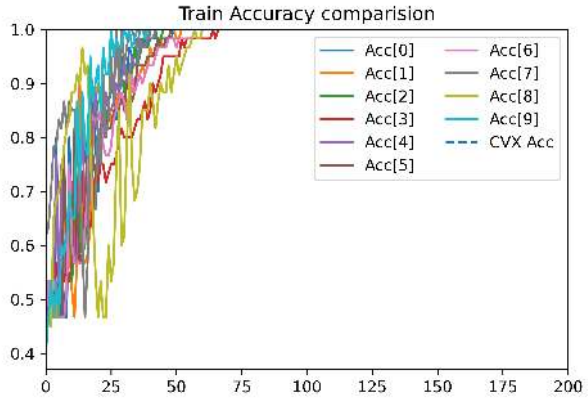
Fig. 5: Training Loss, CIFAR10 N=100,m={50,500,50000}

C. Comparison between CVX solution and Stochastic Gradient Descent, for reduced CIFAR10 Dataset

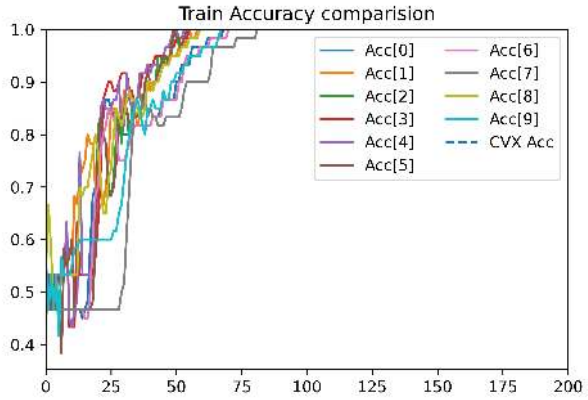
Figure 5, compare the training and validation loss between different realisation of SDG and the convex solution on the CIFAR10 Dataset. Simnularly, figure 6, 7 illustrates the training and validation classification accuracy.



(a) m=50

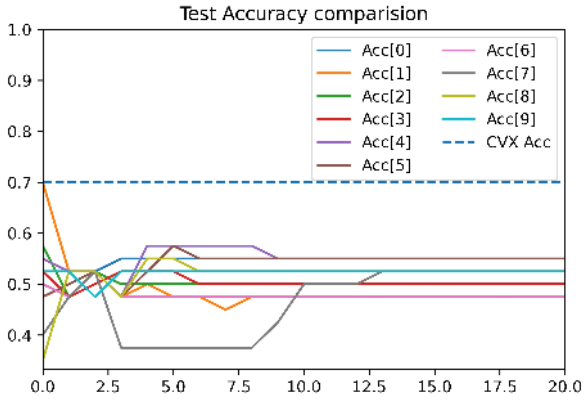


(b) m=500

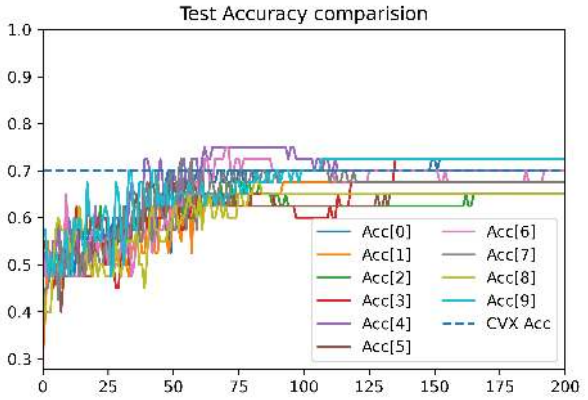


(c) m=50000

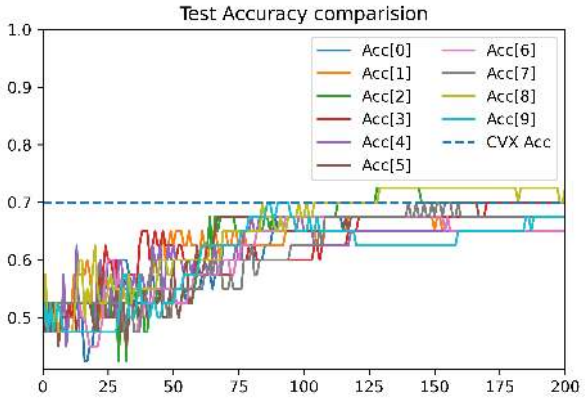
Fig. 6: Training Accuracy, CIFAR10 N=100,m={50,500,50000}



(a) m=50



(b) m=500



(c) m=50000

Fig. 7: Validation Accuracy, CIFAR10 N=100,m={50,500,50000}

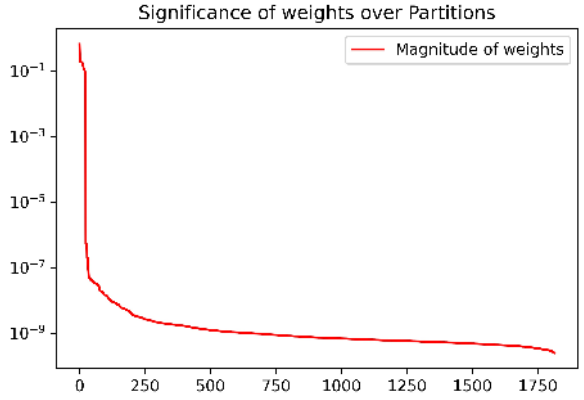


Fig. 8: Significant components in the Partition space

We observe that in certain instances the neural network trained using SGD is performing better than the convex optimal solution. This is because the optimal algorithm is actually the approximate down-sampled This is a consequence of small number of partition samples used

VII. Sparsity in the Partition Space

We observe that the obtained weights for the optimal neural network are sparse in the partition space. Figure 8 show the number of significant partitions in the partition space.

VIII. Conclusion

The highly non-convex neural network training problem can be cast into a convex problem. This convex treatment of the network help us obtain global optimality guarantees and the provides us with the ability to use convex solvers to train the network reliabily with convergence guarantees. We need not rely on on heuristic methods such as gradient decent.

This project has exposed me to the novel ideas in study of neural networks and has helped develop an intuitive understanding for the functionality of neural networks. It is fascinating to see the applications of convex optimisation in the highly non-convex neural networks. It provides and interesting interpretation of the role of the ReLU activation function.

References

[1] M. Pilanci and T. 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.

[2] Y. Wang, J. Lacotte, and M. Pilanci, "The hidden convex optimization landscape of two-layer relu neural networks: an exact characterization of the optimal solutions," *arXiv preprint arXiv:2006.05900*, 2020.

[3] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[4] A. Domahidi, E. Chu, and S. Boyd, "Ecos: An socp solver for embedded systems," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 3071–3076.