Robust Single Layer Neural Network

Group 8 - CS777 Investigatory Project Report

Varun Khare
150793
Department of CSE
IIT Kanpur
varun@iitk.ac.in

Harshvardhan
150283
Department of EE
IIT Kanpur
harshv@iitk.ac.in

Abstract

This work aims to derive a non-trivial breakdown point for an algorithm for training a single hidden-layer Neural Network. In pursuit of this goal, we propose two algorithms for training a network with ReLU activations. The first approach utilizes the partitioning property of the ReLU function while the second approach utilizes the convexity of the activation function.

1 Introduction

The topics for our Investigatory and Reading Project are the same.

Neural Networks have been observed to be robust to even adversarial corruptions. However, only a handful of theoretical results exist which guarantee robustness of NNs. We have made an attempt to address this, using results and techniques from robust statistics.

To measure the robustness of an algorithm, we want to derive its breakdown point which is the largest number of adversarially corrupted points an algorithm can handle and still guarantee recovery. The presence of breakdown point results for simpler learners like linear regression was a primary motivation to pursue this project using robust statistics.

Although we were not able to achieve all our objectives, but we have analysed two approaches for NN training and have implemented one of them. The rest of this report is structured in the following way : Section 2 contains a brief overview of robustness guarantees of other regression techniques and convergence results for neural networks. Section 3 describes all the details of the exact problem statement which we have worked on with a subsection describing all notations. Section 4 explains our two approaches. Section 5 describes the experiments we performed for our methods while the next two sections deal with the challenges encountered and the possible extensions of our work.

2 Related Works

There has been considerable work in robust statistics and results for breakdown point exist for the cases of linear regression[3][1]. Iterative Hard Thresholding algorithm proposes an AltOpt treatment of the problem by identifying

the corrupted samples and then solving the linear regression on the remaining samples. Works on robust logistic regression like [4] change the loss function into a form more amenable to robustness analysis. These works deal with corruption in the responses. Corruption in the covariates has been analysed in the work of [2] for the special case of sparse regression. The authors define a new robust inner product to simplify their analyses.

3 Problem Statement

The problem statement formulated for this Investigatory project is to design and analyse a robust training algorithm for a neural network. To simplify the problem, we consider the case of regression only single hidden layer neural networks with ReLU activation on each node and an output node with no activation with squared loss function. The corruptions are assumed to be present in the responses only and can be modeled by the following equation.

$$y_j = h\left(\mathbf{x}_j, \theta\right) + \eta_j,\ y \in \mathbb{R}$$

Here h is the regressor representing the neural network and η_j represents the corruption for a noiseless training set.We assume that the number of corruptions are unbounded in magnitude but are present in only a fraction of the training dataset.

3.1 Notation

- Training points $(\mathbf{x}, y) \in \mathcal{D}, y \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^d$
- K hidden layer nodes.
- $W = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K], W \in \mathbb{R}^{d \times K}$ where each $\mathbf{w}_i$ represents the weight vector from the input to the i^{th} node.
- $\mathbf{v} = [v_1, v_2, \ldots, v_k]^T, \mathbf{v} \in \mathbb{R}^K$. Each v_i represents the weight of the edge from the i^{th} hidden node to the output node.
- The predicted output for the neural network can be computed as -

$$\hat{y} = \sum_{i=1}^{K} v_i \cdot max\left\{\langle \mathbf{w}_i, \mathbf{x} \rangle, 0\right\}$$

- The loss function value is

$$\mathcal{L}\left(\mathbf{x}, y\right) = \left(\sum_{i=1}^{K} v_i \cdot max\left\{\langle \mathbf{w}_i, \mathbf{x} \rangle, 0\right\} - y\right)^2$$

4 The Proposed Approaches

4.1 Approach 1 : ReLU Partitions

4.1.1 Partitions

This approach utilizes the prpoerty of ReLU to divide the space of the input $\mathbb{R}^d$ into 2 half-spaces, one where the ReLU is active and the other where it is 0.

$$max\left\{\langle \mathbf{w}_i, \mathbf{x} \rangle, 0\right\} = \begin{cases} \langle \mathbf{w}_i, \mathbf{x} \rangle & \text{if } \langle \mathbf{w}_i, \mathbf{x} \rangle > 0 \\ 0 & \langle \mathbf{w}_i, \mathbf{x} \rangle \leq 0 \end{cases}$$

Since $\langle \mathbf{w}_i, \mathbf{x} \rangle = 0$ represents a plane in $\mathbb{R}^d$ which divides the space into two half-subspaces. Each node can define a different plane corresponding to it's $\mathbf{w}$. Now, to analyse the final output, considering no relation between number of

nodes K and input space dimension d, there can be a maximum of 2^K regions. We can label each region according to the nodes which are active in it. We will have one region having 0 nodes active and 1 region having all nodes active.

Our training algorithm for such a network becomes simple, first separate the training set into these regions and corresponding to each region, learn a linear classifier. The robustness results for linear regression can be applied to make this step robust. After obtaining the new weight vectors, we re-partition the training set.

This algorithm easily becomes intractable and ill-posed. Firstly, for accurate estimation of the linear classifier in each partition, we require sufficient number of training points in that partition. Since there are 2^K partitions and these partitions are changing with each iteration, we cannot guarantee sufficient training points in each partition at every iteration. This will make the problem ill-posed in partitions having scarcity of points. However, there is some structure in this problem which we have not exploited. These partitions are created by only K weight vectors. This promotes the idea of sharing information from datapoint-rich partitions to datapoint-poor partitions.

Suppose we have 2 nodes $\mathbf{w}_1$ and $\mathbf{w}_2$ then in the scenario when no points lie in the region where only one of them is active, we will be able to estimate the complete term $v_1\mathbf{w}_1 + v_2\mathbf{w}_2$ and not its individual components. This is also a defect of this partitioning approach. But, more importantly, it can be converted to a condition on the dataset which for this 2 node case is , for every possible pair of $\mathbf{w}_1, \mathbf{w}_2$, there should be sufficient datapoints in atleast 2 regions where atleast 1 node is active. Extending this to the K-node case gives us the condition of sufficient datapoints in that many regions, whose linear combination can give us an estimate of each individual $\mathbf{w}$. For the K-node case, the minimum number of such regions is K. We have not delved deeper into obtaining a proper definition or analysis for the above-mentioned condition. The precise details of the infer operation also need to worked out.

Since this algorithm tries to utilise the structure of the ReLU function, it is not applicable to networks having other activation functions. Also, this algorithm will scale very poorly with increase in number of layers in the network.

Algorithm 1: ReLU Partition-based Training

Input: $S = (\mathbf{x}_j, y_j)_{j=1}^n, K$
Output: $W, \mathbf{v}$

$W^0, \mathbf{v}^0 \leftarrow INIT,$
for $t = 1, 2, \ldots, T$ **do**
  $S_1, \ldots, S_{2^K} \leftarrow \{\}$
  for $i = 1, 2, \ldots, n$ **do**
    $S_{\sum_{j=1}^K \mathbb{I}\{\langle \mathbf{w}_j, \mathbf{x}\rangle > 0\} 2^{j-1}}.append\left(\mathbf{x}_i\right)$
  end for
  for $r = 1, 2, \ldots, 2^K$ **do**
    $W_r = \underset{W_r \in \mathbb{R}^d}{argmin} \sum_{j \in S_r} \left(\sum_{i=1}^K v_i \cdot max\left\{\langle \mathbf{w}_i, \mathbf{x}_j\rangle, 0\right\} - y\right)^2$
  end for
  $W^t, \mathbf{v}^t \leftarrow INFER\left(W_1, W_2, \ldots W_{2^K}\right)$
end for
return $W^T, \mathbf{s}^T$

4.2 Approach 2 : Alternating Optimization

4.2.1 Method

$$\begin{aligned}
\mathcal{L}(\mathbf{X}, y) &= \sum_{j=1}^{n} \left(\sum_{i=1}^{K} v_i \cdot \max\left\{ \langle \mathbf{w}_i, \mathbf{x} \rangle, 0 \right\} - y \right)^2 \\
&= \sum_{j=1}^{n} \left(\sum_{i=1}^{K} sign(v_i) |v_i| \cdot \max\left\{ \langle \mathbf{w}_i, \mathbf{x} \rangle, 0 \right\} - y \right)^2 \\
&= \sum_{j=1}^{n} \left(\sum_{i=1}^{K} sign(v_i) \cdot \max\left\{ \left\langle |v_i|^2 \mathbf{w}_i, \mathbf{x}_j \right\rangle, 0 \right\} - y_j \right)^2 \\
&= \sum_{j=1}^{n} \left(f(\mathbf{x}_j, W_P) - g(\mathbf{x}_j, W_N) - y_j \right)^2
\end{aligned}$$

where

$$f(\mathbf{x}_j, W_P) = \sum_{sign(v_i)=1} \cdot \max\left\{ \left\langle |v_i|^2 \mathbf{w}_i, \mathbf{x}_j \right\rangle, 0 \right\}$$

and

$$g(\mathbf{x}, W_N) = \sum_{sign(v_i)=-1} \cdot \max\left(\left\langle |v_i|^2 \mathbf{w}_i, \mathbf{x}_j \right\rangle, 0 \right)$$

The vector of $sign(v_i)$ is denoted as $sign(\mathbf{v})$. Thus, $sign(\mathbf{v}) \in \{-1, 1\}^K$. For a given value of $sign(\mathbf{v})$, the functions f and g are convex in W_P and W_N respectively (as max is always a convex function). Now, fixing both $sign(\mathbf{v})$ and W_N (equivalent to fixing $sign(\mathbf{v})$ and $g(\mathbf{x}, W_N)$), the loss function is square of a convex function ($f(\mathbf{x}, W_P)$), thus it is convex.

$$\sum_{j=1}^{n} \left(f(\mathbf{x}_j, W_P) - r \right)^2$$

.(r is a constant)

Solving a convex optimization problem is simple and we can ensure convergence to the minima using Gradient-descent style algorithms. Similarly, fixing $sign(\mathbf{v})$ and W_P, gives us another convex problem where ensuring convergence is easy. Solving the convex optimization on f translates to updating W_P while doing the same on g results in updating W_N. Since W_P and W_N consist of mutually exclusive hidden nodes' weight vectors $\mathbf{w}$ so optimizing f and g is equivalent to optimizing weight vectors of certain nodes. The complete W matrix has the weight vectors for each node, thus optimizing f and g is same as performing block-of-coordinate-wise updates on the W matrix where the coordinates represent weight vectors of different nodes. This perspective turns out to be of great use when proving convergence for this algorithm.

Fixing f and g,

$$\mathcal{L}(X, \mathbf{y}) = \|A\mathbf{s} - \mathbf{b}\|_2^2$$

,

where $A_{j,i} = \max\left(\left\langle |v_i|^2 \mathbf{w}_i, \mathbf{x}_j \right\rangle, 0 \right)$ and $\mathbf{s} = [sign(v_1), sign(v_2), \ldots, sign(v_K)]^T \in \{-1, 1\}^K, \mathbf{b} = \mathbf{y}$.

The loss function resembles the sparse recovery objective except the vectors $\mathbf{s} \in \{-1, 1\}^K$. To convert this to a sparse format we perform the following computation.

$$\mathbf{s} = \frac{1}{2}(\mathbf{p} - \mathbf{1})$$

$\mathbf{p}$ vector is K-sparse and its non-zero entries are always 1. Projection onto the vector space of $\mathbf{p}$ vectors(let it be $\mathcal{C}$) is also thresholding. For a vector $\mathbf{s} \in \mathbb{R}^K$, to compute projection of $\mathbf{s}$ on $\mathcal{C}$, we minimize

$$\mathbf{p} = \underset{\mathbf{p} \in \mathcal{C}}{argmin} \|\mathbf{s} - \mathbf{p}\|_2^2$$

Solving this for each coordinate, $(\mathbf{s}_i - \mathbf{p}_i)^2$ needs to be minimized where $\mathbf{p}_i = 0, 1$. This is minimized if $\mathbf{s}_i > \frac{1}{2} \implies \mathbf{p}_i = 1$ else $\mathbf{p}_i = 0$. Thus, we get the projection rule onto set $\mathcal{C}$.

Theorem 0.1. Projection of vector $\mathbf{s}$ on the set of binary vectors $\mathcal{C} \in \mathbb{R}^K$ is given by -

$$\mathbf{p}_i = \begin{cases} 1 & \text{if } \mathbf{s}_i > \frac{1}{2} \\ 0 & \text{if } \mathbf{s}_i \leq \frac{1}{2} \end{cases} \forall i \in [K]$$

Since the Projection step is easy, we can solve this sparse-recovery problem using Projected Gradient Descent.

These update equations can be performed sequentially to obtain an Alternating Optimization algorithm for minimizing the complete loss function. Moreover, each subproblem is easy to solve and can be shown to converge. The complete algorithm is presented in Algorithm 2.

Algorithm 2: Alt-Opt-NN

Input: $S = \{(\mathbf{x}_j, y_j)\}_{j=1}^n, K$
Output: $W, , \mathbf{s}$
1: $W^0, \mathbf{s}^T \leftarrow INIT,$
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $\quad W_P^t = \underset{W_P}{\arg\min} \sum_{j=1}^n \left(f\left(\mathbf{x}, W_P\right) - r\right)^2$
4: $\quad W_N^t = \underset{W_N}{\arg\min} \sum_{j=1}^n \left(g\left(\mathbf{x}, W_N\right) - r^-\right)^2$
5: $\quad \mathbf{s}^t = \underset{\mathbf{s} \in \{-1,1\}^K}{\arg\min} \|A\mathbf{s} - \mathbf{b}\|_2^2$
6: **end for**
7: **return** $W^T, \mathbf{s}^T$

4.2.2 Analysis

Alternating algorithms can easily get stuck, so we will need to show non-zero decrease in the loss function value in each iteration. We will assume complete minimisation of convex optimization problems with f and g and Projected Gradient Descent for solving the sparse-recovery problem.

We will make the following assumptions to make analysis simple-

- The complete objective function $h\left(W, \mathbf{s}\right) = \mathcal{L}\left(X, \mathbf{y}\right)$ is β-smooth with respect to the Frobenius norm of W. Thus,
$$h\left(W_1, \mathbf{s}\right) \leq h\left(W_0, \mathbf{s}\right) + \langle \nabla_W h\left(W_0, \mathbf{s}\right), W_1 - W_0 \rangle + \frac{\beta}{2} \|W_1 - W_0\|_F^2$$
- We add a regularizer with respect to $\mathbf{s}$ to the objective function to make it α-RSC over the set of binary vectors $\mathcal{C}$. Thus, $h\left(W, \mathbf{s}\right) = \mathcal{L}\left(X, \mathbf{y}\right) + \frac{\alpha}{2} \|\mathbf{s}\|_2^2$
- To enable convergence in sparse recovery, we need $\nabla_{\mathbf{s}} h\left(\mathbf{s}^*\right) = 0$

Complete minimization with respect to f gives us -

$$h\left(W_P^*, W_N, \mathbf{s}\right) \leq h\left(W_P, W_N, \mathbf{s}\right) + \langle \nabla_{W_P} h\left(W_P, W_N, \mathbf{s}\right), W_P^* - W_P \rangle + \frac{\beta}{2} \|W_P^* - W_P\|_F^2$$

Since $W_P^* = W_P - \lambda$, where λ is a matrix.

$$h\left(W_P^*, W_N, \mathbf{s}\right) \leq min_\lambda \left[h\left(W_P, W_N, \mathbf{s}\right) + \langle \nabla_{W_P} h\left(W_P, W_N, \mathbf{s}\right), \lambda \rangle + \frac{\beta}{2} \left\|\lambda\right\|_F^2 \right]$$

$$h\left(W_P^*, W_N, \mathbf{s}\right) \leq h\left(W_P, W_N, \mathbf{s}\right) - \frac{1}{2\beta} \left\|\nabla_{W_P} h\left(W_P, W_N, \mathbf{s}\right)\right\|_F^2$$

Note that this analysis is similar to that done for coordinate descent.

We obtain a similar equation after optimizing for g -

$$h\left(W_P^*, W_N^*, \mathbf{s}\right) \leq h\left(W_P^*, W_N, \mathbf{s}\right) - \frac{1}{2\beta} \left\|\nabla_{W_N} h\left(W_P^*, W_N, \mathbf{s}\right)\right\|_F^2$$

$$\implies h\left(W_P^*, W_N^*, \mathbf{s}\right) \leq h\left(W_P, W_N, \mathbf{s}\right) - \frac{1}{2\beta} \left\|\nabla_{W_P} h\left(W_P, W_N, \mathbf{s}\right)\right\|_F^2 - \frac{1}{2\beta} \left\|\nabla_{W_N} h\left(W_P^*, W_N, \mathbf{s}\right)\right\|_F^2$$

If we run PGD for sparse recovery until convergence then,

$$\implies h\left(W_P^*, W_N^*, \mathbf{s}^*\right) \leq h\left(W_P, W_N, \mathbf{s}\right) - \frac{1}{2\beta} \left\|\nabla_{W_P} h\left(W_P, W_N, \mathbf{s}\right)\right\|_F^2 - \frac{1}{2\beta} \left\|\nabla_{W_N} h\left(W_P^*, W_N, \mathbf{s}\right)\right\|_F^2 - \frac{\alpha}{2} \left\|\mathbf{s}^* - \mathbf{s}\right\|_2^2$$

(By α-RSC) Thus, there is decrease in the objective function value in each iteration.

5 Experiments

We encountered certain errors while implementing the algorithm because of problems with Xavier initialization due to the ReLU function. The update equations due to renormalization of each hidden layer output made the gradients for backpropagation difficult to compute. Due to paucity of time, we were not able to implement this algorithm for other activation functions like sigmoid or Leaky ReLU.

6 Future Works

1. Saddle Point : Although this technique won't get stuck , it can still land up at a saddle point since the decrease is proportional to the norm of gradient. For this, we need to further investigate the nature of the problem and add constraints.
2. Robustness: Making the algorithm robust can now be broken down into making each subproblem robust and then ensuring that each iteration of the AltOpt is also robust. The latter should be the more difficult task. Coming up with
3. Ensuring β-RSS and α-RSC for the sparse recovery problem by putting conditions on the A matrix. Since the A matrix is recomputed at each iteration, we will have to come up with constraints on the data distribution and the weight vectors to ensure these properties.
4. $\beta - SS$ wrt the Frobenius norm is difficult to ensure for the ReLU function. Thus, we will have to use Fenchel duality or Nesterov's Theorem for approximate smoothness. Also, we can use a different smooth and convex activation function for Method-2 since the only constraint for the algorithm to work is for the activation to be convex.
5. Extension of the rule proposed for Method-1 to connect the vertices of the lattice for information-sharing.

7 Acknowledgments

We thank our mentor Dr. Purushottam Kar for his patient guidance during our project. We also thank our classmates of CS777 for the insightful discussions both inside and outside the classroom.

References

[1] BHATIA, K., JAIN, P., AND KAR, P. Robust regression via hard thresholding. *CoRR abs/1506.02428* (2015).

[2] CHEN, Y., CARAMANIS, C., AND MANNOR, S. Robust sparse regression under adversarial corruption. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, Georgia, USA, 17–19 Jun 2013), S. Dasgupta and D. McAllester, Eds., vol. 28 of *Proceedings of Machine Learning Research*, PMLR, pp. 774–782.

[3] JAIN, P., AND KAR, P. Non-convex optimization for machine learning. *Found. Trends Mach. Learn. 10*, 3-4 (Dec. 2017), 142–336.

[4] LI, Y., AND YUAN, Y. Convergence analysis of two-layer neural networks with relu activation. *CoRR abs/1705.09886* (2017).