Incremental Training of a Two Layer Neural Network

Group 10

Gurpreet Singh
150259
guggu@iitk.ac.in

Jaivardhan Kapoor
150300
jkapoor@iitk.ac.in

Abstract

Gradient boosting for convex objectives has had a rich history and literature with provable guarantees for many years now. The same cannot be said for the workings of a neural network, while the class of neural networks is a set of incredibly powerful models, which can approximate complex function mappings. In this project, we make an attempt to combine the two approaches with a boosted model as a warm start for a single layer neural network, with provable convergence guarantees. We also see how gradient boosting on single node single hidden layer network essentially corresponds to sequential training of hidden layer nodes, and therefore can be used as a starting point for application of the backpropagation scheme for better results. Among these, we also look at the convergence analysis of functional gradient descent, which is used to train the weak learners, or nodes in our case, and empirical results received thereafter.

1. Introduction

Feed Forward Neural Networks are incredible function approximators. However, there are a few caveats when it comes to training a neural network. Firstly, there is less on the proper initialization of the neural network nodes to ensure convergence. Also, the most common method of optimization is Stochastic Gradient Descent, which although provably converges to the global optima in finite steps (for finite optimal points), offers no such guarantees in case of non-convex functions, and since a neural network is a multi-level mapping of non-linear functions, the objective is highly non-convex. Therefore, SGD methods only work as heuristics in the case of neural networks.

We attempt to remedy this, up to a certain sense, by incrementally training a neural network. Although we look at only single hidden layer neural networks, it is possible to extend the theory to multiple layers as well, but the complexity of training would increase in that case.

The essential motivation behind this is that a single hidden layer neural network can be thought of as an ensemble of multiple nodes, which in our analogy become the weak learners, and the whole network as an ensemble or mixture of these weak learners. This analogy allows us to compare neural networks with other ensemble techniques, such as bagging and boosting. Since the weak learners in our case are essentially single

node in a single hidden layer network, the learners have very high bias, and since bagging relies on the bias of each learner, we cannot find much use of it.

However, the method of boosting fits very naturally in our problem setting, where we can learn one node (one weak learner) on the input data, and learn the next node on the residual of this node, and the next on the residual of the second node, and so on. This allows us to incrementally train each node, where one node communicates with a previous node only through the residuals. Throughout the text, we refer to the boosting method as incremental training of the neural network, and the standard backpropogation scheme as the collective training of the neural network.

We first give a brief background of the tasks we have at hand, and then a brief history of boosting and a sketch of how we use boosting for our training. Later, we move on to an empirical comparison between incremental training and collective training of a neural network.

2. Relevant Background

2.1. Function Estimation

In the function estimation problem, we usually have a set of tuples of data points, and labels or response variables, commonly denoted by $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively. We assume these tuples to be generated from a data distribution $\mathcal{D}$, which is unknown to us, and the goal is to estimate a function $f : \mathcal{X} \to \hat{\mathcal{Y}}$ such that the error or risk (defined below) is minimized.

Definition 1.1 (Risk). The risk for a function estimate is given with respect to a loss function $l : \mathcal{Y} \times \hat{\mathcal{Y}} \to \mathbb{R}$ and a data distribution $\mathcal{D}$, and is defined as

$$\mathcal{R}^l_{\mathcal{D}}[f] \triangleq \mathop{\mathbb{E}}_{\mathbf{X},\mathrm{Y}\sim\mathcal{D}}[l(\mathrm{Y}, f(\mathbf{X}))]$$

The choice of loss function depends on the problem we wish to solve, and the range of the response variable $\mathcal{Y}$ and the function range $\hat{\mathcal{Y}}$. For example, if the response variable and the function range, both are Bernoulli variables, then an apt choice of the loss function is $0-1$ loss, which is given as $l^{0-1}(Y, \widehat{Y}) \triangleq \mathbb{I}\left[Y = \widehat{Y}\right]$.

We can now formally define the goal of a function estimation problem. Given a data distribution $\mathcal{D}$, where random variables $\mathbf{X} \in \mathcal{X}$ and $\mathrm{Y} \in \mathcal{Y}$ are draw from this distribution, we wish to estimate a function $f^* : \mathcal{X} \to \hat{\mathcal{Y}}$ such that the following holds

$$f^* \triangleq \arg\min_f \mathop{\mathbb{E}}_{\mathbf{X},\mathrm{Y}\sim\mathcal{D}}[l(\mathrm{Y}, f(\mathbf{X}))]$$

2.1.1. Working with Finite Data

Since we do not actually know the data distribution, it is impossible to find a function to minimize with respect to the data distribution. We therefore find a surrogate objective to minimize, and ensuring, under certain conditions [1], that the proxy function we found using the surrogate objective is representative of the function f^*, where f^* is as defined earlier.

[1] Some sufficient conditions are stability of the Loss function or Uniform Convergence of the hypothesis space. Details can be found in Shalev Shwartz and Ben David (2014)

The surrogate objective is defined as follows

$$\hat{f} \triangleq \arg\min_{f} \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x},y\sim\mathcal{S}} l(y, f(\mathbf{x}))$$

where the set $\mathcal{S} \triangleq \{(\mathbf{x}^n, y^n)\}_{n=1}^N$ is sampled from the data distribution $\mathcal{D}^N$ identically and independently.

The objective to minimize in the above expression is known as the empirical risk with respect to a sample set, and is formally given as

$$\mathcal{R}^l_{\mathcal{S}}[f] \triangleq \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x},y\sim\mathcal{S}} l(y, f(\mathbf{x}))$$

Remark. The term on the RHS of the above optimization objective is defined as the empirical risk, denoted by $\mathcal{R}^l_{\mathcal{S}}$.

Note. For the remaining text, we assume that we have a sample set $\mathcal{S}$ sampled i.i.d. from the data distribution which is used for the purposes of training. This is also known as the training set.

2.1.2. Working with Restricted Hypothesis Classes

Since the objectives described above are over unconstrained spaces, we can achieve minimum error by minimizing the error point-wise, however this is usually not desired. Therefore, we limit the minimization over a restricted Hypothesis Space, denoted by $\mathcal{F}$. This allows us to control the function class over which we optimize, and better analyze the algorithm. Hence, our new objective becomes

$$\hat{f} = \arg\min_{f\in\mathcal{F}} \frac{1}{N} \sum_{(\mathbf{x},y)\in\mathcal{S}\sim\mathcal{D}^N} l(y, f(\mathbf{x}))$$

2.2. Gradient Descent

Gradient Descent (Cauchy, 1847) or *Steepest Descent* is a method to optimize (minimize) convex objectives, usually where a closed form solution is either not available or slow to compute analytically. For example, consider the objective of logistic regression, where no closed form solution is available. Therefore, we need numerical methods to reach the optimal value, $\hat{f}$.

Assume we have a convex objective $\mathbf{\Phi}$ to be minimized over a variable $\mathbf{x} \in \mathcal{X}$. A vanilla gradient descent step for the given setting looks like the following

1. First, the gradient $\mathbf{g}_t$ at the current time step t is computed as

$$\begin{aligned} \mathbf{g}_t &\triangleq [\nabla_{\mathbf{x}} \mathbf{\Phi}(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_t} \\ &= \left[\frac{\delta(\mathbf{\Phi}(\mathbf{x}))}{\delta(\mathbf{x})}\right]_{\mathbf{x}=\mathbf{x}_t} \end{aligned}$$

2. The current estimation of the optimal point $\mathbf{x}_t$ is updated using the negative gradient weighted with a step value η_t as follows

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \cdot \mathbf{g}_t$$

Gradient Descent offers good convergence guarantees for convex objectives, and better for strongly-convex and/or strongly-smooth objectives. We have discussed this is detail in our survey of *Gradient Descent and its Variants*.

We now move on to Boosting Methods, especially Gradient Boosting in the next section.

3. Generic Boosting Algorithms for Convex Optimization

Definition 1.2 (Boosting). Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias and variance (Breiman, 1996) in supervised learning, and a family of machine learning algorithms which combine multiple weak learners to form a superior strong learner[2].

AdaBoost (Freund and Schapire, 1995) was the first successful boosting algorithm designed for classification tasks. AdaBoost uses decision stumps as weak learners and each weak learner is trained on the same data samples with the loss weighted by the rate of mis-classification for each data point. Freund and Schapire (1995) showed that AdaBoost in fact converges and provided an upper bound on the risk for the combined classifier. In fact, only algorithms that provably converge in the PAC framework can be classified as boosting algorithms.

Many other boosting algorithms exist for problem specific tasks exist, such as LPBoost, TotalBoost, BrownBoost, LogitBoost, etc. exist. Friedman (2001) provided a generic technique for boosting algorithms which provides general methods for application in any supervised learning setting with a convex objective.

Friedman in his paper proposed a novel boosting algorithm, known as Gradient Boost which is very closely related to the method of Gradient Descent, and therefore enjoys the convergence properties of Gradient Descent, although not directly.

Although there are other general boosting algorithms such as AnyBoost and DOOM proposed by Mason et al. (1999b), we exclude them from our report, since Gradient Boost fits more naturally into our paradigm. We discuss the Gradient Boost Algorithm in detail in the following sections.

3.1. Gradient Boost

For the standard Boosting Methods, we restrict the form of the function $\hat{f}$ to be an additive function, *i.e.* we restrict $\hat{f}$ such that there exist functions $\{f_m \in \mathcal{H}\}_{m=1}^{M}$ where $\mathcal{H}$ is a subset of the hypothesis space of $\hat{f}$ ($\mathcal{H} \in \mathcal{F}$) and values $\{\beta_m \in \mathbb{R}\}_{m=1}^{M}$, we have

$$\hat{f} \quad = \quad \sum_{m=1}^{M} \beta_m f(\mathbf{x}; \mathbf{a}_m) \tag{1}$$

where for each $m \in [\, M \,]$, $\mathbf{a}_m$ represent the set of parameters that define the m^{th} component of the function $\hat{f}$.

Remark. It is not necessary for the $\hat{f}$ to be in the hypothesis space $\mathcal{H}$ since the space might not be closed under addition or scalar multiplication, or both.

This is termed as an additive model, and boosting essentially builds an additive model. Friedman et al. (2000) showed that even AdaBoost builds an additive model, even though there is no explicit notion of additive modeling within AdaBoost.

[2] Source: https://en.wikipedia.org/wiki/Boosting_(machine_learning)

Since we want to minimize the empirical risk defined in Section 2.1.1, we can write the objective $\mathbf{\Phi} : \mathcal{F} \to \mathbb{R}$ as

$$\mathbf{\Phi}(f) \triangleq \frac{1}{N} \sum_{(\mathbf{x},y)\in\mathcal{S}\sim\mathcal{D}^N} l(y, f(\mathbf{x}))$$

and the optimal point is given as $\hat{f} = \arg\min_{f\in\mathcal{F}} \mathbf{\Phi}(f)$.

Since we know that the opimal point is an additive model, we can replace the objective as

$$\{\beta_m, \mathbf{a}_m\}_{m=1}^M = \underset{\{\beta_m,\mathbf{a}_m\}_{m=1}^M}{\arg\min} \mathbf{\Phi}\left(\sum_{m=1}^M \beta_m f(.\,;\mathbf{a}_m)\right)$$

However, for most objectives, this does not give a closed form solution. Therefore, we need to apply alternate methods to optimize $\mathbf{\Phi}$. A proposed method is to compute the individual components greedily, *i.e.* we keep adding a component $\beta_m f(.\,;\mathbf{a}_m)$ to $\hat{f}$ until convergence.

This is greedy stage wise procedure, where the current estimate, *i.e.* at time step $m-1$, of $\hat{f}$ is given by $\hat{f}_{m-1}$, and the update step for the stage wise procedure can be written as two steps, given below

$$\begin{aligned}\beta_m, \mathbf{a}_m &= \underset{\beta,\mathbf{a}}{\arg\min}\ \mathbf{\Phi}(\hat{f}_{m-1} + \beta f(.\,;\mathbf{a})) \\ \hat{f}_m &= \hat{f}_{m-1} + \beta_m f(.\,;\mathbf{a}_m)\end{aligned}$$

At this point, a resemblance can be seen between the above updates and the method of Gradient Descent (Mason et al., 1999a; Friedman, 2001). In fact, Gradient Boosting is essentially a combination of Gradient Descent and Boosting. Comparing the above update step to the update step of gradient descent, one can be convinced that replacing the m^{th} component of $\hat{f}$ or the update of the function $\hat{f}_{m-1}$ can be replaced by the gradient of the objective with respect to the update component, and replace β_m with the step value at time step m.

More formally, we replace the update steps as follows

1. Compute the gradient of the minimizing objective $\mathbf{\Phi}$ as

$$g_{m-1} = \left[\frac{\delta\left(\mathbf{\Phi}(f)\right)}{\delta\left(f\right)}\right]_{f=\hat{f}_{m-1}}$$

2. The function estimate is then updated as

$$\hat{f}_m = \hat{f}_{m-1} - \eta_m g_{m-1}$$

From the ideas of gradient descent, we can expect this to converge to the global optima in case the objective is convex with respect to the function space $\mathcal{F}$ or even the space covered by functions represented as an additive model.

We still have one problem to solve. The gradient in this case is a function. And in order to minimize the empirical risk, we minimize the loss function point-wise. Therefore, at a data point $(\mathbf{x}, y)$, we want the value of the gradient $g_m(\mathbf{x}, y)$ to be such that we take a maximum descent step from the current function estimate, at this point. More formally, we want the value of the gradient at point $(\mathbf{x}, y)$ to be such that

$$g_m(\mathbf{x}, y) = \left[\frac{\delta\left(l(y, f(\mathbf{x}))\right)}{\delta\left(f\right)}\right]_{f=\hat{f}_m}$$

as this would allow us to take steepest descent step for each data point.

Now, since the values of the gradient at any time step depend on the data points in the set $\mathcal{S}$, we only know the actual value of the gradient at these points. Therefore, instead of having a complete function, we have the value of the gradient function at a finite set of points $\mathcal{S}$.

This motivates us to approximate this gradient using another function, which is to be learned. This is the point where the notion of weak learners come in. We approximate or estimate the gradient using a function $h_m \in \mathcal{H}$ which belongs to a restricted hypothesis class. This function h_m acts as a proxy for the gradient g_m and therefore is used to update the function estimate $\hat{f}_{m-1}$ at the previous time step. We define the function h_m to be as follows

$$h_m \quad \triangleq \quad \arg\max_{h \in \mathcal{H}} \frac{\langle g_m,\ h_m \rangle}{\| h_m \|} \tag{2}$$

where the dot product and norm is defined over the L^2 function space (defined later). If the hypothesis space $\mathcal{H}$ is closed under scalar multiplication, the objective becomes (Grubb and Bagnell, 2011)

$$h_m \quad \triangleq \quad \arg\max_{h \in \mathcal{H}} \| h_m - g_m \|^2 \tag{3}$$

Since we do not have the data distribution, we cannot actually compute the dot product or the norm in the L_2 space, we approximate the objective using the sample set $\mathcal{S}$, which allows us to know the values of g_m at those points as well as find a good proxy function from the restricted hypothesis space.

This method is known as *Restricted Gradient Descent* where we use a surrogate to the gradient to make the actual gradient step.

The step is chosen either using Heuristics, or using line search, where the optimal value of β_m is computed such that the update minimizes the loss. The complete algorithm for Gradient Boost is given in Algorithm 1. We formalize the notion of Restricted Gradient Descent and Gradient Boosting in the next section, while giving a convergence bound for the same.

3.2. Convergence Analysis of Gradient Boost

We refer to Grubb and Bagnell (2011) for the convergence analysis of the Restricted Gradient descent method used for the objective here. The cited work contains some relevant definitions which we shall restate here for completeness. We shall then proceed to state theorems on convergence guarantees in regards to the notation thus introduced.

3.2.1. L^2 Function Space

Ratliff et al. (2009) proposed that the L^2 function space is a natural match for analyzing the convergence of gradient descent. For an input domain $\mathcal{X}$, an output vector space $\mathcal{V}$, and a measure μ, the $L^2(\mathcal{X}, \mathcal{V}, \mu)$ space is defined as the set of functions $f : \mathcal{X} \to \mathcal{V}$ such that the following Lebesgue integral is finite.

$$\int_{\mathcal{X}} \| f(\mathbf{x}) \|^2 \ \mathrm{d}\mu \quad < \quad \infty$$

If μ is a probability measure, then the above can be written as an expectation $\mathop{\mathbb{E}}_{P}[\, \| f(\mathbf{x}) \|\,]$.

Algorithm 1: Gradient Boost

Input: a sequence of data points $\mathcal{S} \sim \mathcal{D}^N$ and a loss function $l : \mathcal{Y} \times \hat{\mathcal{Y}} \to \mathbb{R}$

Output: $\hat{f}$, an estimate of the optimal function

Steps:

1. Initialize $\hat{f}_0 \in \mathcal{H}$
2. For $m = 0 \dots M-1$ or until convergence, do

$$\begin{aligned} \forall\, n \in [\, N \,], \quad g_{m,n} &= \left[\frac{\delta\left(l(y_n, f(\mathbf{x}_n))\right)}{\delta\left(f\right)} \right]_{f=\hat{f}_m} && \text{(Gradient Step)} \\ h_m &= \arg\max_{h \in \mathcal{H}} \frac{\langle g_m,\ h_m \rangle}{\| h_m \|} && \text{(Weak Learning Step)} \\ \hat{f}_{m+1} &= \hat{f}_m - \beta_m h_m && \text{(Update Step)} \end{aligned}$$

   where β_m is equal to $\eta_m \cdot \frac{\langle g_m,\, h_m \rangle}{\| h_m \|}$ and η_m is obtained using line-search or using a heuristic step value.

3. Return $\hat{f} = \hat{f}_M$

Initialize a function $\hat{f}_0$

The L^2 space is a Hilbert space, with the inner product given as

$$\begin{aligned} \langle f,\, g \rangle_P &= \int_{\mathcal{X}} f(\dot{\mathbf{x}}) g(\mathbf{x})_{\mathcal{V}} P(\mathbf{x})\, \mathrm{d}\mathbf{x} \\ &= \mathop{\mathbb{E}}_{P} \left[\, \langle f(\mathbf{x}),\, g(\mathbf{x}) \rangle_{\mathcal{V}} \,\right] \end{aligned}$$

and

$$\| f \|_P^2 = \langle f,\, f \rangle_P$$

If only a set of samples (say $\mathcal{S}_P$) are available from the measure P, we can replace the expectation using these samples as

$$\langle f,\, g \rangle_P^2 = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{S}_P} \langle f(\mathbf{x}),\, g(\mathbf{x}) \rangle_{\mathcal{V}}$$

3.2.2. Aligning Power of a Hypothesis Space

We will consider the Hilbert space of functions in the empirical space $\hat{P}$, where the inner product is defined as

$$\langle f, g \rangle = \frac{1}{N} \sum_{n=1}^{N} \langle f(\mathbf{x}_n), g(\mathbf{x}_n) \rangle$$

From the above algorithm, we see that the ideal direction for the gradient to move in the restricted space, is the learned weak learner. We can interpret as performing GD in the

function space, to minimize the value of the functional (in this case the loss function). Since the empirical space is restricted, there are only a restricted number of directions we can move. The best direction is given by the weak learner that is learned at every step.

We define the "aligning power" of the hypothesis space $\mathcal{H}$ (the restricted space of the weak learners), the ability of it to align to the real gradient of the functional. This is also known as the edge of the hypothesis space. We present the formal definition of edge as given by Grubb and Bagnell (2011).

Definition 1.3 (Edge). A restricted set $\mathcal{H}$ has edge γ if $\forall \mathbf{g}$, $\exists h \in \mathcal{H}$ such that $\langle g,\, h \rangle \geq \gamma \,\|\, g \,\|\, \|\, h \,\|$

3.2.3. Convergence Bound

We assume that the hypothesis space for the weak learners is a proper subset of the $L^2(\mathcal{X}, \mathcal{Y}, \mathcal{D})$ where $\mathcal{X}$ is the vector space of the input featues, $\mathcal{Y}$ is the vector space of the output vectors and $\mathcal{D}$ (a probability measure) defines the underlying data distribution we assume. Below we analyze the convergence in case the risk functional is strongly convex and strongly smooth for the Gradient Boosting Algorithm given in Algorithm 1.

Also, since we do not have the data distribution, we find the convergence of the empirical risk with respect to the sample set $\mathcal{S}$. To ensure convergence, between the empirical risk and the real risk, statistical analysis must be down, but we do not cover that in this report.

Theorem 1.1. Let $\mathcal{R}^l_{\mathcal{S}}[\,.\,]$ be a λ-strongly-convex and β-strongly-smooth functional over the L^2 space given by the tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{D})$. Also, let the edge of the restricted hypothesis space $\mathcal{H}$ be given by γ. Then, given a starting point f_0 and step size $\eta = \frac{1}{\beta}$, we have

$$\mathcal{R}^l_{\mathcal{S}}[\,f_T\,] - \mathcal{R}^l_{\mathcal{S}}[\,f^*\,] \leq \left(1 - \gamma^2 \frac{\lambda}{\beta}\right)^T \left(\mathcal{R}^l_{\mathcal{S}}[\,f_0\,] - \mathcal{R}^l_{\mathcal{S}}[\,f^*\,]\right) \tag{4}$$

Proof. From the definition of strong smoothness, we can write the following inequality,

$$\mathcal{R}^l_{\mathcal{S}}[\,f_{t+1}\,] - \mathcal{R}^l_{\mathcal{S}}[\,f_t\,] \quad \leq \quad \left\langle \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,],\; f_{t+1} - f_t \right\rangle + \frac{\beta}{2} \,\|\, f_{t+1} - f_t \,\|^2_{\mathcal{D}}$$

From the update step in Algorithm 1, we have $f_{t+1} - f_t = \frac{1}{\beta} \frac{\langle \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,],\, h_t \rangle}{\|\, h_t \,\|^2} h_t$. Putting this in the above expression, we get

$$\begin{aligned} \mathcal{R}^l_{\mathcal{S}}[\,f_{t+1}\,] - \mathcal{R}^l_{\mathcal{S}}[\,f_t\,] \quad &\leq \quad -\frac{1}{2\beta} \frac{\left\langle \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,],\; h_t \right\rangle^2}{\|\, h_t \,\|^2} \\ \implies \mathcal{R}^l_{\mathcal{S}}[\,f_{t+1}\,] - \mathcal{R}^l_{\mathcal{S}}[\,f^*\,] \quad &\leq \quad \mathcal{R}^l_{\mathcal{S}}[\,f_t\,] - \mathcal{R}^l_{\mathcal{S}}[\,f^*\,] - \frac{1}{2\beta} \frac{\left\langle \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,],\; h_t \right\rangle^2}{\|\, h_t \,\|^2} \end{aligned}$$

Since we know that the edge of the hypothesis space $\mathcal{H}$ is γ, we have $\left\langle \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,],\; h_t \right\rangle \geq \gamma \left\|\, \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,] \right\| \|\, h_t \,\|$. Therefore, we can transform the above inequality to write

$$\mathcal{R}^l_{\mathcal{S}}[\,f_{t+1}\,] - \mathcal{R}^l_{\mathcal{S}}[\,f^*\,] \quad \leq \quad \mathcal{R}^l_{\mathcal{S}}[\,f_t\,] - \mathcal{R}^l_{\mathcal{S}}[\,f^*\,] - \frac{\gamma^2}{2\beta} \left\|\, \nabla \mathcal{R}^l_{\mathcal{S}}[\,f_t\,] \right\|^2$$

Here, using strong convexity we have $\| \nabla R[f_t] \|^2 \geq 2\lambda(R[f_t] - R[f^*])$. Thus we get the result,

$$\begin{aligned} \mathcal{R}^l_{\mathcal{S}}[f_{t+1}] - \mathcal{R}^l_{\mathcal{S}}[f^*] &\leq \left(1 - \gamma^2 \frac{\lambda}{\beta}\right)\left(\mathcal{R}^l_{\mathcal{S}}[f_t] - \mathcal{R}^l_{\mathcal{S}}[f^*]\right) \\ \implies \mathcal{R}^l_{\mathcal{S}}[f_{t+1}] - \mathcal{R}^l_{\mathcal{S}}[f^*] &\leq \left(1 - \gamma^2 \frac{\lambda}{\beta}\right)^T \left(\mathcal{R}^l_{\mathcal{S}}[f_0] - \mathcal{R}^l_{\mathcal{S}}[f^*]\right) \end{aligned}$$

□

While the convergence in the above case is linear, other optimization techniques also perform similarly well on such constraints, giving linear rates of convergence. We also see that the advantage of this algorithm quickly breaks down in the case of non smooth objectives unless we constrain the domain, and as a result Grubb and Bagnell (2011) also propose a variant of the above algorithm, in which instead of taking one projection, we take multiple projections steps on the residual from the previous projection step.

The modified algorithm (Algorithm 2, Grubb and Bagnell, 2011) provides similar guarantees for strongly convex and strongly smooth objectives, and in case of strongly convex functionals provides sublinear rate of convergence, *i.e.* $\mathcal{O}\left(\frac{\log T}{T}\right)$. The same algorithm in the case of simply convex functionals provides a bound on the regret given below, which we shall interepret shortly.

$$\frac{1}{T}\sum_{t=1}^{T}(\mathcal{R}^l_{\mathcal{S}}[f_t] - \mathcal{R}^l_{\mathcal{S}}[f^*]) \leq \frac{F^2}{2\sqrt{T}} + \frac{G^2}{\sqrt{T}} + 2FG\frac{1-\gamma^2}{2\gamma^2} \tag{5}$$

where F is the upper bound on the empirical norm[3] of the gradients of the risk, and G is the upper bound on the empirical norm of the function f.

One can see from Equations 4 and 5, that the convergence benefits from a high value of the edge of the restricted space, however there is a key difference between the two equations. In case of strongly-convex and strongly-smooth, decreasing the value of γ would only slow down the convergence, however the algorithm would still converge to the global optima, with regret tending to 0.

However, in the above expression, one can see that with increasing number of steps, there is still a term that does not decay, proportional to $\frac{1}{\gamma^2} - 1$. This means that the "aligning power" of the weak learner plays a role in this case on the regret. A sufficiently good weak learner ($\gamma \simeq 1$) will cause the regret term to go down and the loss function converges to the optimum, however this would increase the time per iteration in the model. On the other hand, a terrible weak learner (having classification rate close to $\frac{1}{K}$ for K-class classification for example) will cause γ to be very small, correspondingly increasing regret, as well as slowing convergence.

We shall see this effect in our empirical results, discussed in the next section, seeing that the node activations we have used have poor classification rates by themselves, increasing the regret and producing suboptimal results.

4. Experiments - Softmax Regression

We perform a simple experiment on the MNIST Dataset, where the task in hand is to classify the images based on the digit in the image. We use Softmax Regression to approach this problem, using a single hidden layer neural network.

[3] the norm in the L^2 space with respect to the data distribution $\mathcal{D}$ approximated using the sample set $\mathcal{S}$

As discussed, we try two approaches to training, an incremental approach to training, and another, the standard backpropogation scheme. The rest of the parameters in training, wherever applicable, are set to be the same.

For both the cases, the number of nodes are chosen to be 50. This means that in the case of incremental training, we use 50 stops or 50 weak learners, and take an ensemble of the weak learners so as to form a single layer neural network.

4.1. Training of the weak learners

As mentioned in Section 3.1, we train each weak learner to maximize the mutual direction between the learned function and the gradients from the previous time step. In order to compute the gradient, we need to first formulate a loss function.

Since we are using softmax regression, we can use the standard loss function, that is the cross-entropy loss. Suppose we have K classes, then the loss function is given as

$$l(y, \hat{\mathbf{y}}) \quad = \quad -\sum_{k=1}^{K} \mathbb{I}\,[\,y = k\,] \log \hat{\mathbf{y}}_k$$

where $\hat{\mathbf{y}} = \sigma(f(\mathbf{x}))$ $\left(\text{where } f(\mathbf{x}) \in \mathbb{R}^K\right)$ is a vector with K dimensions, each dimension being equal to the predictive probability of the point $\mathbf{x}$ being in the corresponding class.

We can now write the gradient at the $(\mathrm{m} + 1)^{\mathrm{th}}$ time step for this loss as

$$\begin{aligned} g_m(\mathbf{x}, y) \quad &= \quad \left[\frac{\delta\left(l(y, \sigma(f(\mathbf{x})))\right)}{\delta\left(f\right)} \right]_{f=\hat{f}_m} \\ &= \quad [\,\mathbb{I}\,[\,y = k\,] - f(\mathbf{x})|_k\,]_{k=1}^{K} \end{aligned}$$

where $f(\mathbf{x})|_k$ denotes the k^{th} dimension of $f(\mathbf{x})$.

Since this is dependent on y, we can only compute the value of the gradient at points from the set $\mathcal{S}$ (sampled set). Therefore, we need to approximate this gradient using a weak learner from the hypothesis set $\mathcal{H}$. We choose this learner to be a single node single hidden layer neural networks (essential for the idea of incremental training), where the single node in the m^{th} time step corresponds to the m^{th} node in the hidden layer of the target neural network (to be learned incrementally).

Since in this case, the hypothesis space is closed under scalar multiplication (as the scalar will get multiplied only to the weights incoming on the output layer, which does not disturb the hypothesis class), we can set the objective as given in equation 3. Since the objective is given for the data distribution which is not known, we replace the objective by an empirical objective, where the optimization is only over the data points belonging to the sample set $\mathcal{S}$. Since our objective is a linear regressor, we choose the norm to be a l_2 norm. Therefore, our final objective becomes a simple least squares objective, given as

$$h_m \quad \triangleq \quad \underset{h \in \mathcal{H}}{\arg\min} \sum_{(\mathbf{x}, y) \in \mathcal{S}} \| \, g(\mathbf{x}, y) - h(\mathbf{x}, y) \, \|_2^2$$

In Figure 1, we have shown the performance of the said boosted system, or equivalently, an incrementally trained neural network. It is clear that each individual weak learner we have chosen is embarrassingly weak, yet the performance of the trained network is decent on the dataset. However the performance does not compare well with that of a neural network trained using the backpropogation scheme, which gives over 95% test accuracy. Therefore, even for a simple example, we can conclude that incremental training gives poorer results when compared to the collective training of a network.

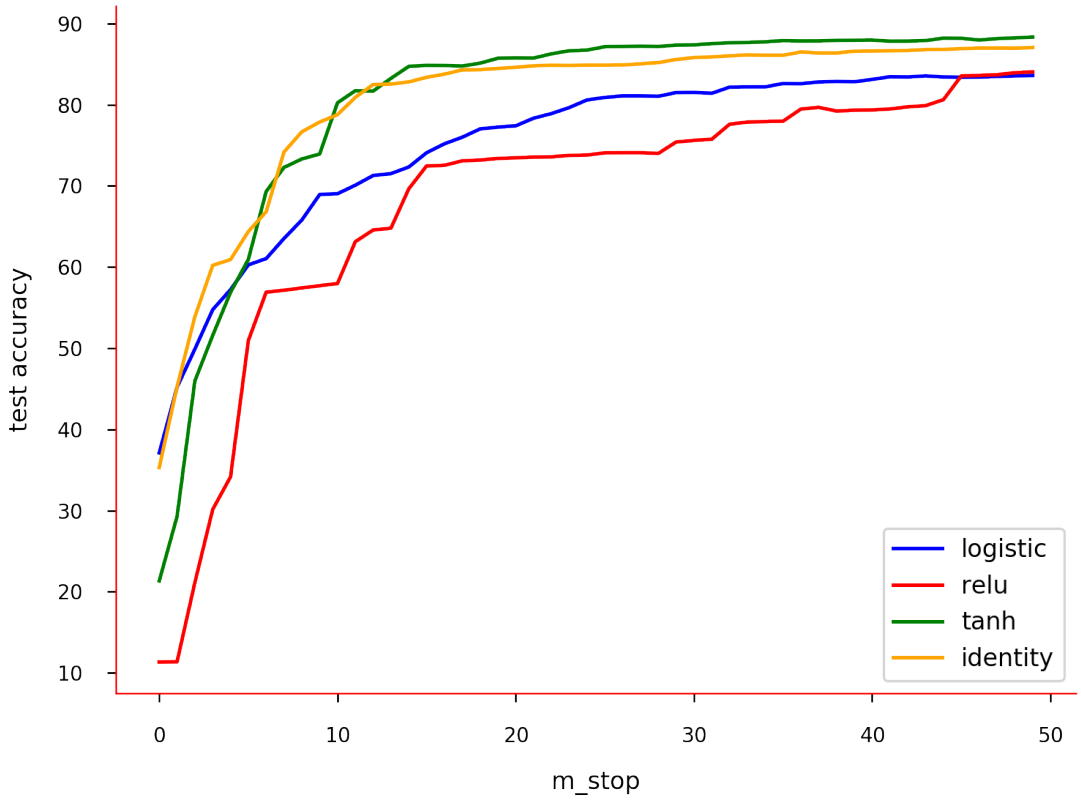

Figure 1: Test accuracy versus the number of trained nodes in an incrementally trained network

4.2. Incremental Training as a Pre-Training Step

Training of a neural network, or any model can be broken into two parts, pre-training and fine-tuning. For most tasks, the pre-training is skipped, and the weights are initialized using random values, therefore only the notion of fine-tuning remains.

Working on the hypothesis that the boosted model, or the incrementally trained neural network can be a good initialization (or pre-training), we performed experiments on the same classification task, with the incrementally trained neural network as a pre-training step for the network, and then applying the standard backpropogation over this pre-trained network for the purpose of fine-tuning.

The final test accuracies for each case has been reported in Table 1. It can be seen that there are marginal improvements on the test accuracies in the case of networks which have been incrementally pre-trained. This suggests that the pre-training, although does not change the performance by a lot, can improve performance marginally, which might be helpful in cases where the predictions are extremely sensitive, for example Cancer detection.

We explicitly perform experiments on the class imbalance case, with the Iris dataset where the imbalance is manually induced. The results for the same are shown in Table 2. It can be clearly seen that the accuracies obtained in the case of collectively trained neural network are embarrassingly low. We also ran the training of the network multiple times with an extremely low tolerance to ensure the convergence, however the network always seemed to converge with a similar final accuracy. This could be explained by the convergence of the network to a local optima, since the objective is highly non-convex. Whereas, the pre-trained neural network seems to avoid this local optima, and converge to a better optima.

Activations	**Incremental**	**Collective**	**Pre-trained + Fine-tuned**
sigmoid	85.000000	97.110000	97.460000
relu	84.900000	96.980000	97.310000
tanh	87.840000	96.450000	96.950000
identity	85.640000	91.710000	91.820000

Table 1: Test Accuracies for Differently Trained Neural Networks

Activations	**Incremental**	**Collective**	**Pre-trained + Fine-tuned**
sigmoid	78.181818	76.363636	94.545455
relu	97.272727	45.454545	97.272727
tanh	81.818182	48.181818	97.272727
identity	98.181818	92.727273	97.272727

Table 2: Test Accuracies for Differently Trained Neural Networks

We confirmed this hypothesis by reducing the number of nodes in the collectively trained neural network, and observed a boost in accuracy. This strongly suggests that the added non-convexity due to larger number of nodes was preventing the network to converge to the global optima. Whereas, the pre-trained network did not have this problem even with a large number of nodes, since the network is already close to a decent optima, local or global.

4.3. Mini-Batch Training — Stronger Weak Learners

One problem with the pre-training that is easily observable is that the training takes a lot of time, and in case of marginal performance boosts, one might simply prefer the collective training model for the sake of saving time. We further extend the training to a mini-batch setting in order to decrease the total time of training, while maintaining the lower non-convexity for each weak learner.

The idea of a mini-batch is to allow multiple nodes in one weak learner, and reduce the total number of stops (or boosted weak learners) so as to keep the total number of nodes. Since the strength of each weak learner is greatly improved, it will converge to a better accuracy. Also, since the collective training of a neural network takes lesser time, the total time is expected to be lesser than the time spend boosting single node single hidden layer networks.

However, increasing the number of nodes in a single weak learner increases the non-convexity of each learner, however, if we limit the number of nodes in a weak learner to a small value (say < 5), then we can almost always ensure that each weak learner will converge to its global optima. This allows us to decrease the total pre-training time, while ensuring similar convergence guarantees.

This remains a hypothesis, and we could not this due to shortage of time, but we leave this approach for future exploration.

5. Conclusion

In this project report, we have discussed the Gradient Boosting algorithm for convex objectives, applied it to sequentially train the nodes of a single hidden layer neural network and derived empirical results on different weak learners as node activations. The results are summarized in Table **??**.

There are a few observations worth noting. The improvement that was anticipated as a result of the warm start provided by incremental training of nodes is appreciable, and is attributed to provable convergence guarantees of a boosted model which allows the neural network to converge to a better optima.

Further, the pretraing method may induce structure in the individual activations of the nodes thereafter trained. This mode of training is vaguely similar to the greedy layer-wise approach in Deep Boltzmann Machines, though in this case the greedy approach is applied to single layer among the nodes. We also propose the mini-batch training, which could help in reducing the pre-training time while maintaining the good convergence properties of the sytem.

We also notice from Figure 1 that as the number of weak learners increase, the accuracy obtained slowly saturates to a suboptimal level compared to vanilla backpropagation. We may apply early stopping in this case to infer the number of nodes ideal to a layer in the network, which can be then fine tuned to yield better results.

Further work in this regard, in the direction of multi-layer pretraining using gradient boosting, may lead to efficient inference of model complexity and better warm starts. The problem that may arise in such a case is that objectives are no longer convex for the boosting procedure, giving us no such convergence guarantees for this methods. This remains a promising avenue for exploration, one that may benefit from advances in the developing theory of nonconvex optimization techniques.

References

Leo Breiman. Bias, Variance and Arcing Classifiers. 1996.

M. Augustine Cauchy. Méthode générale pour la résolution des syst'emes d'équations simultanées. 1847.

Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. *Computational learning theory, pp. 23–37, Springer*, 1995.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Ann. Statist.*, 28(2):337–407, 04 2000. doi: 10.1214/aos/1016218223. URL `https://doi.org/10.1214/aos/1016218223`.

Alexander Grubb and J Andrew Bagnell. Generalized boosting algorithms for convex optimization. *arXiv preprint arXiv:1105.2054*, 2011.

L. Mason, Baxter J, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. *MIT Press*, 1999a.

Llew Mason, Peter Bartlett, Jonathan Baxter, and Marcus Frean. *Boosting Algorithms as Gradient Descent*. NIPS'99. MIT Press, Cambridge, MA, USA, 1999b. URL `http://dl.acm.org/citation.cfm?id=3009657.3009730`.

Nathan D. Ratliff, David Silver, and J. Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, Jul 2009. ISSN 1573-7527. doi: 10.1007/s10514-009-9121-3. URL `https://doi.org/10.1007/s10514-009-9121-3`.

Shai Shalev Shwartz and Shai Ben David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014. ISBN 1107057132, 9781107057135.