Graphon-aided Graphical Lasso with Structured Sparsity

Anton Banta [*1] **Madeline Navarro** [*1] **Nicolas Zilberstein** [*1]

Abstract

In this paper, we consider the problem of inferring a graphical representation of complex data with specific interdependencies. The classic approach involves the graphical lasso method which aims to reconstruct a sparse graph using the ℓ_1-norm penalty. We build upon this framework by incorporating prior knowledge of the graph structure to guide the pruning process during graph inference. Specifically, we assume the underlying graph is drawn from a specific random graph model via the graphon. We provide a generalized method with user-defined structural characteristics for graph inference. Furthermore, we provide convergence guarantees of the algorithm and experimental results on synthetic data which demonstrate an improvement in graph inference when utilizing our proposed spectral penalty. This improvement was observed when the true graphon distribution was both known or approximated by the user.

1. Introduction

Estimating interdependencies among multiple measured variables allows us to structurally describe complex systems of many variables. For example, brain networks or gene-to-gene networks provide identifiers for individuals, applicable for improved diagnoses or personalized treatments (Yang et al., 2015). We represent dyadic statistical relationships via undirected graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ corresponding to observed variables and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ corresponding to conditional dependence between pairs of variables. Edges exist between variables i and j if and only if two variables are conditionally dependent given all other variables. These networks are called Markov random fields, and in the case of Gaussian random variables, we have Gaussian graphical models (GGMs).

We consider the common method of graphical lasso (Friedman et al., 2008), which estimates GGMs via maximum likelihood estimation with a penalty encouraging sparsity. The classic graphical lasso method estimates a sparse graph structure via a ℓ_1-norm penalty. Encouragement of a parsimonious structure mitigates computational burdens of potential downstream tasks and provides a more interpretable representation. However, some kind of prior knowledge about the underlying graph structure may be available. For example, in webpage, brain, or gene networks, certain nodes may be hubs, that is, a small number of nodes are densely connected to many other nodes (Barabási & Albert, 1999; Hao et al., 2012; Luo, 2014). Indeed, application of an ℓ_1 sparsity regularizer as with graphical lasso assumes that all edges must be equally penalized. Consider implementation of prior structural knowledge in a generalized setting of probabilistic structural information rather than expected families of graphs. In particular, we wish to encourage graph structure based on a known random graph model. With the probabilistic graph information, we can implement structured sparsity in place of the ℓ_1 sparsity regularizer.

Several ways of incorporating structured sparsity have been proposed. In general, it is done by combining different norms. Slawski et al. (2010) propose a structured elastic net, where the ℓ_1-norm is combined with $\|\cdot\|_{\mathbf{\Theta}}$, where $\mathbf{\Theta}$ is the graph Laplacian matrix. The latter penalty is equivalent to the forward operator defined for fused lasso (Tibshirani et al., 2005) for time-varying signals, and it captures the *a priori* association structure. Bach et al. (2012) demonstrate selecting groups of variables in a fine-tuned way if the variables form a union-closed set. They also mention that this form of structured sparsity can be applied to graphical lasso as long as the union closed set property is satisfied. In addition to enforcing structure by selecting specific edges for group lasso terms, we can also consider encouraging structural behavior over the entire graph. Vizuete et al. (2021) assume prior knowledge of a family of graphs whose spectra are expressive of the structural characteristics. They introduce a set of types of graphs that can be estimated by applying particular spectral constraints, which are detailed in the paper.

We introduce a modified graphical lasso method with *induced structured sparsity* based on prior statistical knowledge of the network structure. In particular, we assume that the underlying graph is sampled from a known random

[*]Equal contribution [1]Department of Electrical and Computer Engineering, Rice University, Houston, Texas, United States. Correspondence to: Madeline Navarro <nav@rice.edu>.

Project report for COMP414/514, Rice CS.

graph model (Diaconis & Janson, 2007; Erdős & Rényi, 1959; Holland et al., 1983; Lovász, 2012). The above methods of structured sparsity are designed for specific structures to be encouraged. We propose a generalized method with user-defined structural characteristics that encompasses the previous methods and permits arbitrary families of networks. By careful choice of random graph model, a desired structure can be encouraged. Additionally, our proposed method allows use of prior knowledge of the family to which the estimated graph belongs to aid estimation performance.

2. Problem Formulation

The graphical lasso formulation (Friedman et al., 2008) for estimating the precision matrix $\mathbf{\Theta} \in \mathbb{R}^{p\times p}$ given a sample covariance matrix $\mathbf{S}$ is

$$\begin{aligned}&\min_{\mathbf{\Theta}} -\log\det\mathbf{\Theta} + \text{tr}(\mathbf{S\Theta}) + \alpha\phi(\mathbf{\Theta})\\ &\text{s.to } \mathbf{\Theta} \in \mathbb{R}^{p\times p}_{++}\end{aligned}$$

where $\phi(\cdot)$ is a function inducing sparsity. We observe the effects of choosing the ℓ_0 pseudo-norm and its convex relaxation, the ℓ_1 norm to encourage sparsity. The precision matrix is positive definite, hence the constraint that $\mathbf{\Theta}$ is in the set of $p \times p$ positive definite matrices $\mathbb{R}^{p\times p}_{++}$. However, we can instead implement the constraint $\mathbf{\Theta} \in \mathbb{R}^{p\times p}_{+}$, where $\mathbb{R}^{p\times p}_{+}$ denotes the set of $p \times p$ positive semidefinite matrices, and we replace the generalized determinant $\text{gdet}(\mathbf{\Theta})$, i.e., the product of the nonzero eigenvalues of $\mathbf{\Theta}$ (Kumar et al., 2020). Then, the generalized graphical lasso formulation can be presented as

$$\begin{aligned}&\min_{\mathbf{\Theta}} -\log\text{gdet}\,\mathbf{\Theta} + \text{tr}(\mathbf{S\Theta}) + \alpha\phi(\mathbf{\Theta})\\ &\text{s.to } \mathbf{\Theta} \in \mathbb{R}^{p\times p}_{+}\end{aligned}$$

We may also reduce the feasible set by requiring that $\mathbf{\Theta}$ be a graph Laplacian matrix by the constraint $\mathbf{\Theta} \in \mathcal{S}_{\mathbf{L}}$, where

$$\mathcal{S}_{\mathbf{L}} = \Big\{\mathbf{\Theta} \in \mathbb{R}^{p\times p} : \Theta_{ij} = \Theta_{ji}\ \forall\, i \neq j,\ \Theta_{ij} \leq 0\ \forall\, i \neq j,\ \Theta_{ii} = -\sum_{j\neq i}\Theta_{ij}\Big\},$$

thus $\mathbf{\Theta}$ is positive semidefinite with k zero eigenvalues, where k is the number of connected components in the graph corresponding to the support of $\mathbf{\Theta}$ (Kumar et al., 2020). Then, we rewrite the graphical lasso problem as

$$\begin{aligned}&\min_{\mathbf{\Theta}} -\log\text{gdet}\mathbf{\Theta} + \text{tr}(\mathbf{S\Theta}) + \alpha\phi(\mathbf{\Theta})\\ &\text{s.to } \mathbf{\Theta} \in \mathcal{S}_{\mathbf{L}}.\end{aligned}$$

The graph Laplacian is symmetric with diagonal entries as linear combinations of the off-diagonal entries, thus there are $p(p-1)/2$ degrees of freedom for estimating $\mathbf{\Theta}$. We introduce the graph Laplacian operator $\mathcal{L} : \mathbb{R}^{p(p-1)/2} \to \mathbb{R}^{p\times p}$ (Kumar et al., 2020) to convert a vector to a matrix with graph Laplacian structure, where we have that

$$[\mathcal{L}\mathbf{x}]_{ij} = \begin{cases} -x_{i+d_j} & i > j,\\ [\mathcal{L}\mathbf{x}]_{ji} & i < j,\\ -\sum_{j\neq i}[\mathcal{L}\mathbf{x}]_{ij} & i = j\end{cases}$$

where $d_j = -j + \frac{j-1}{2}(2p-j)$. We let the adjoint $\mathcal{L}^* : \mathbb{R}^{p\times p} \to \mathbb{R}^{p(p-1)/2}$ be defined by

$$[\mathcal{L}^*\mathbf{Y}]_k = Y_{ii} - Y_{ij} - Y_{ji} + Y_{jj}$$

where $k = i - j + \frac{j-1}{2}(2p-j)$, and $i > j$. Note that $\mathcal{L}^*$ is defined to satisfy $\langle\mathcal{L}\mathbf{x}, \mathbf{Y}\rangle = \langle\mathbf{x}, \mathcal{L}^*\mathbf{Y}\rangle\ \forall\mathbf{x}, \mathbf{Y}$. Finally, we also have that the operator norm $\|\mathcal{L}\|_2 = \sqrt{2p}$.

We consider the case where we have prior knowledge about eigenvalues of $\mathbf{\Theta}$. If we let $\mathbf{\Theta} = \mathbf{U}\text{diag}(\boldsymbol{\lambda})\mathbf{U}^\top$ be the eigendecomposition of $\mathbf{\Theta}$, consider the case where we have an approximation of $\boldsymbol{\lambda}$ that we will denote as $\boldsymbol{\mu}$. Then, we can include the eigendecomposition in the optimization problem and include the prior knowledge of the eigenvalues while retaining graph Laplacian characteristics, i.e., at least one zero-valued eigenvalue. Then, the problem formulation is

$$\begin{aligned}&\min_{\mathbf{w},\mathbf{U},\boldsymbol{\lambda}} -\sum_{i:\lambda_i\neq 0}\log\lambda_i + \text{tr}(\mathbf{S}\mathcal{L}\mathbf{w}) + \alpha\phi(\mathbf{w})\\ &\text{s.to}\quad \mathbf{w} \geq 0,\ \mathcal{L}\mathbf{w} = \mathbf{U}\text{diag}(\boldsymbol{\lambda})\mathbf{U}^\top,\ \mathbf{U}^\top\mathbf{U} = \mathbf{I},\\ &\qquad \lambda_1 = 0,\ \lambda_i = \mu_i, i > 1.\end{aligned}$$

However, if we know that $\boldsymbol{\mu}$ is an approximation of $\boldsymbol{\lambda}$, we can relax the equality constraints. What follows is the formulation with relaxed equality constraints.

$$\begin{aligned}&\min_{\mathbf{w},\mathbf{U},\boldsymbol{\lambda}} -\sum_{i:\lambda_i\neq 0}\log\lambda_i + \text{tr}(\mathbf{S}\mathcal{L}\mathbf{w}) + \alpha\phi(\mathbf{w})\\ &\qquad + \frac{\beta}{2}\|\mathcal{L}\mathbf{w} - \mathbf{U}\text{diag}(\boldsymbol{\lambda})\mathbf{U}^\top\|_F^2 + \frac{\gamma}{2}\|\boldsymbol{\lambda} - \boldsymbol{\mu}\|_2^2\\ &\text{s.to}\quad \mathbf{w} \geq 0,\ \mathbf{U}^\top\mathbf{U} = \mathbf{I},\ \lambda_1 = 0 \qquad (1)\end{aligned}$$

3. Statistical assumptions

Consider the case where prior statistical knowledge about the graph structure may be known. Graphs associated with random graph models exhibit characteristics based on the model. For example, stochastic blockmodels can be designed for structures containing communities (Holland et al., 1983). In our case, we consider the more general random graph model, the graphon (Diaconis & Janson, 2007; Lovász, 2012), which provides a general form for many common exchangeable distributions on networks. For example,

Erdos-Renyi graph models and stochastic blockmodels can be represented by graphons that are constant or piecewise-constant, respectively (Erdős & Rényi, 1959; Holland et al., 1983).

A graphon is a bounded symmetric measurable function $W : [0, 1]^2 \to [0, 1]$ whose domain behaves as edges in an infinitely large adjacency matrix and whose range represents edge probabilities. Undirected graphs can be generated from a graphon by (i) assigning each node a random label between 0 and 1, and (ii) assigning an edge between nodes with probability equal to the value of the graphon at their randomly sampled labels. The steps can be presented formally as

$$\zeta_i \sim \text{Unif}([0, 1]) \qquad \forall\, i \in \mathcal{V}, \tag{2a}$$
$$S_{ij} = S_{ji} \sim \text{Bern}\left(W(\zeta_i, \zeta_j)\right) \qquad \forall\, (i, j) \in \mathcal{V} \times \mathcal{V}, \tag{2b}$$

where the latent variables $\zeta_i \in [0, 1]$ are i.i.d. samples drawn for each node i. The function W can be designed to be equivalent to existing popular exchangeable graph models but can also encompass a wider range of graph structures. Thus, graphons present a flexible model for families of graphs with similar structural characteristics. Figure 1 presents examples of families of graphs that can be described by proper choice of graphon.

Returning to the problem (1), we assume some prior information about the eigenvalues of the graph Laplacian. Consider the assumption that the graph to be estimated is *sampled from a graphon* W. Then, we have the following results associating the graph Laplacian spectrum with the graphon from which the graph is sampled.

Theorem 1 (From (Vizuete et al., 2021)) *Let W be a non-decreasing piecewise-Lipschitz graphon that is bounded away from zero. Specifically, we have that*

$$|W(x_1, y_1) - W(x_2, y_2)| \leq L(|x_1 - x_2| + |y_1 - y_2|)$$

for constant L and sequence of K non-overlapping intervals $I_k = [\alpha_{k-1}, \alpha_k)$, where $0 = \alpha_0 < \cdots < \alpha_{K+1} = 1$, and $(x_1, y_1), (x_2, y_2) \in I_k \times I_l$ for any k, l. Furthermore, we have $W(x_1, y) \leq W(x_2, y)$ when $x_1 \leq x_2$, and the infimum of W is bounded away from zero, $W(x, y) \geq \eta > 0$ for x, y. Then, for a graph $\mathcal{G}$ of size p sampled from W, where the eigenvalues of the Laplacian matrix of $\mathcal{G}$ are $0 = \lambda_1 \leq \cdots \leq \lambda_n$, with probability at least $1 - 3\nu$,

$$\|\lambda(x) - d(x)\|_2 \leq C_0 + \sqrt[4]{\frac{2}{p}}\sqrt{\|W\| + C_1} + C_1$$

where $\lambda(x)$ is a piecewise-constant function on $x \in [0, 1)$, i.e., $\lambda(x) = \lambda_i/p$ on $x \in [\frac{i-1}{p}, \frac{i}{p})$ and $\lambda(1) = \lambda_p/p$, $d(x) = \int_0^1 W(x, y)dy$, and C_0, C_1 are constants dependent on ν, L, and η.

Theorem 1 (Vizuete et al., 2021) demonstrates the relationship between the piecewise constant function containing the sorted eigenvalues of the graph Laplacian and the degree distribution of the nondecreasing graphon. Specifically, what this result describes is that the Laplacian eigenvalues become closer to the graphon degree function as the graph size grows. With this result, we can let the degree distribution of the graphon be $\mu(x)$ and apply it in the problem (1). Application of this result was performed in a similar manner in (Roddenberry et al., 2021). However, the authors assume a different graph signal model, and they do not provide theoretical results. In this work, we investigate the solution to the optimization problem along with its convergence guarantees for the case of Gaussian graphical model estimation.

We consider the case where the underlying graphon and the latent points ζ from (2a) are known and apply the prior statistical knowledge for graph Laplacian estimation in (1). In particular, we use the vector $\boldsymbol{\mu}$ whose entries correspond to the value of $\mu(x)$ when x takes values of the latent points ζ_i in (2b), that is, $\boldsymbol{\mu}_i = \mu(\zeta_i)$ for all i.

Remark. Note that in real-world applications, it is unrealistic to assume that the underlying graph family is known. The graph family may be estimated from empirical observations, e.g., senate networks that strongly exhibit clustering behavior of nodes based on political party (Navarro et al., 2020; Zhu et al., 2020). However, in the case where the underlying graph model cannot be known or estimated, the proposed method is still feasible via an approximation of the graphon characteristics. In particular, if a similar graph is available, i.e., a graph sampled from the same graphon, its degree vector can be applied in problem (1) as an approximation of the degree distribution of the graphon. Indeed, as the size of a sampled graph grows larger, the graph converges to the underlying graphon in the limit (Lovász, 2012), thus for a large enough similar graph, the graphon degree distribution can be approximated adequately based on the application. For graphical lasso applied to estimate the brain functional network of a patient who possesses a disease known to alter structural characteristics, the proposed penalty may be applied via the known brain network of a patient with the same disease.

4. Algorithm

Problem (1) can be solved by a block majorization-minimization (MM) algorithm (Kumar et al., 2020), where each variable $\mathbf{w}$, $\mathbf{U}$, and $\boldsymbol{\lambda}$ is updated sequentially. We also show that the proposed update steps converge to the optimal solution of (1).

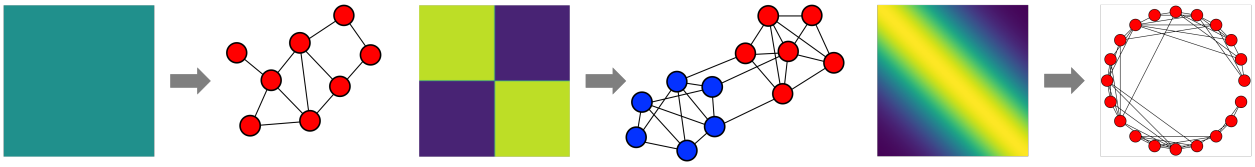


Figure 1. Examples of graph families generated (and thus represented) by graphons. (Left) Constant graphon to represent Erdős-Rényi random graphs, where edges between any pair of nodes are equally probable. (Middle) Piecewise-constant graphons represent graphs associated with stochastic blockmodels, where clusters of nodes are expected. (Right) Graphs with regular-like behavior can be represented by the graphon $W(x, y) = e^{-\phi(x-y)^2}$ for a constant ϕ to tune the approximate degree of each node.

4.1. Update for $\mathbf{w}$

We let $\mathbf{U}$ and $\boldsymbol{\lambda}$ be constants and solve the optimization problem for $\mathbf{w}$. We first consider the case where $\phi(\cdot) = \|\cdot\|_1$. The problem can be written as

$$\begin{aligned} &\min_{\mathbf{w}} \operatorname{tr}(\mathbf{S}\mathcal{L}\mathbf{w}) + \frac{\beta}{2}\|\mathcal{L}\mathbf{w} - \mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top\|_F^2 + \alpha\mathbf{1}^\top\mathbf{w} \\ &\text{s.to} \quad \mathbf{w} \geq 0 \end{aligned}$$

which is equivalent to

$$\begin{aligned} &\min_{\mathbf{w}} \quad f(\mathbf{w}) = \frac{1}{2}\|\mathcal{L}\mathbf{w}\|_F^2 - \mathbf{c}^\top\mathbf{w} \\ &\text{s.to} \quad \mathbf{w} \geq 0 \end{aligned} \tag{3}$$

where $\mathbf{c} = \mathcal{L}^*(\mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top - \frac{1}{\beta}\mathbf{S}) - \frac{\alpha}{\beta}\mathbf{1}$. We optimize a majorization function of the objective function in (3) to account for the inequality constraint (Kumar et al., 2020). Because the objective in (3) is strongly convex, we can write the majorizer of the objective in (3) as

$$\begin{aligned} g(\mathbf{w}|\mathbf{w}^{(t)}) = & f(\mathbf{w}^{(t)}) + (\mathbf{w} - \mathbf{w}^{(t)})^\top\nabla f(\mathbf{w}^{(t)}) \\ & + \frac{L_1}{2}\|\mathbf{w} - \mathbf{w}^{(t)}\|_2^2 \end{aligned}$$

where $f(\mathbf{w})$ corresponds to the objective in (3), and $L_1 = \|\mathcal{L}\|_2^2 = 2p$. We can then write the majorized version of the $\mathbf{w}$ update problem as

$$\begin{aligned} &\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \mathbf{w}^\top\mathbf{a} \\ &\text{s.to} \quad \mathbf{w} \geq 0 \end{aligned}$$

where $\mathbf{a} = \mathbf{w}^{(t)} - \frac{1}{L_1}\nabla f(\mathbf{w}^{(t)})$. Then, we can obtain the optimal solution to the $\mathbf{w}$ update via Karush–Kuhn–Tucker (KKT) conditions as

$$\mathbf{w}^{(t+1)} = \left(\mathbf{w}^{(t)} - \frac{1}{L_1}\nabla f(\mathbf{w}^{(t)})\right)_+ \tag{4}$$

where $(\mathbf{x})_+ := \max\{0, \mathbf{x}\}$ and $\nabla f(\mathbf{w})$ corresponds to the gradient of the objective in (3).

If we let $\phi(\cdot) = \|\cdot\|_0$, we can consider an iterative hard thresholding (IHT) algorithm. In particular, we let $\alpha = 0$ in (3), obtain the solution (4), and project the result onto the k-sparse vector via the sparse projection $\mathcal{H}_k(\mathbf{w})$ where $[\mathcal{H}_k(\mathbf{w})]_i = \mathbf{w}_i$ when i corresponds to k elements in $\mathbf{w}$ with the largest magnitude, and $[\mathcal{H}_k(\mathbf{w})]_i = 0$ otherwise. We leave implementation of this sparsity penalty and investigation of its convergence for future work.

4.2. Update for $\mathbf{U}$

For the $\mathbf{U}$-update problem formulation

$$\begin{aligned} &\min_{\mathbf{U}} \frac{\beta}{2}\|\mathcal{L}\mathbf{w} - \mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top\|_F^2 \\ &\text{s.to} \ \mathbf{U}^\top\mathbf{U} = \mathbf{I} \end{aligned}$$

on the orthogonal Stiefel manifold (Kumar et al., 2020), we have the solution as the eigenvectors of $\mathcal{L}\mathbf{w}$, that is, $\mathbf{U}^{(t+1)}$ satisfies

$$\mathbf{U}^{(t+1)}\operatorname{diag}(\bar{\boldsymbol{\lambda}})\mathbf{U}^{(t+1)} = \mathcal{L}\mathbf{w}^{(t+1)}$$

where $\bar{\boldsymbol{\lambda}}$ is a vector consisting of the eigenvalues of the current value of $\mathcal{L}\mathbf{w}^{(t+1)}$.

4.3. Update for $\boldsymbol{\lambda}$

Given the update problem for $\boldsymbol{\lambda}$

$$\begin{aligned} \min_{\boldsymbol{\lambda}} - \sum_{i:\lambda_i \neq 0} \log\lambda_i + \frac{\beta}{2}\|\mathcal{L}\mathbf{w} - \mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top\|_F^2 \\ + \frac{\gamma}{2}\|\boldsymbol{\lambda} - \boldsymbol{\mu}\|_2^2, \end{aligned}$$

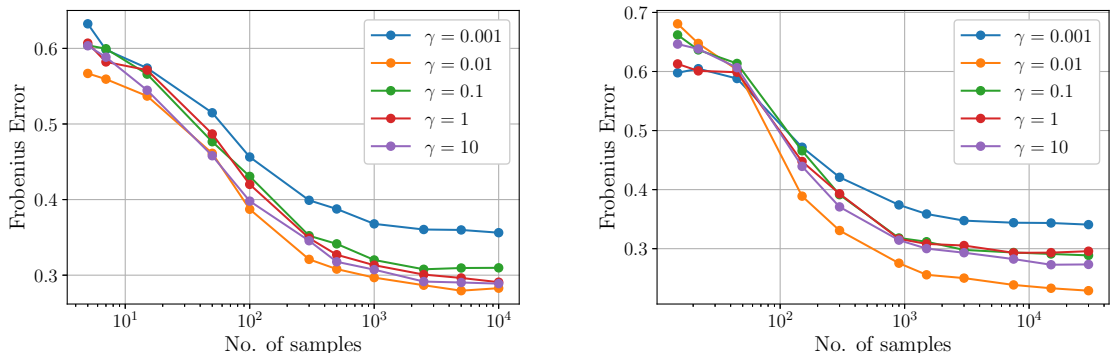
we can rewrite the problem as

$$\min_{\boldsymbol{\lambda}} - \sum_{i:\lambda_i \neq 0} \log\lambda_i + \frac{\beta+\gamma}{2}\|\boldsymbol{\lambda} - \mathbf{d}\|_2^2 \tag{5}$$

where $\mathbf{d} = \frac{\beta}{\beta+\gamma}\mathbf{U}^\top\mathcal{L}\mathbf{w}\mathbf{U} + \frac{\gamma}{\beta+\gamma}\boldsymbol{\mu}$. We update $\boldsymbol{\lambda}$ as

$$\lambda_i^{(t+1)} = \frac{1}{2}\left(d_i + \sqrt{d_i^2 + \frac{4}{\beta+\gamma}}\right) \ \forall i = 2, 3, \ldots, p$$

and we have that $\lambda_1^{(t+1)} = 0$ as required by the graph Laplacian structure.



(a) (b)

Figure 2. (a) Recovery error for 30-node graph sampled from the graphon $W(x, y) = \frac{1}{2}(x^2 + y^2)$ and recovered with the proposed structured graphical lasso (1) using the true graphon degree distribution. (b) Recovery error for 30-node graph sampled from the graphon $W(x, y) = \frac{1}{2}(x^2 + y^2)$ and recovered with the proposed structured graphical lasso (1) using an *approximation* of the graphon degree distribution obtained from a graph sampled from the same graphon.

5. Convergence results

In this section, we prove that the proposed algorithm updating $\mathbf{w}^{(t)}$, $\mathbf{U}^{(t)}$, and $\boldsymbol{\lambda}^{(t)}$ converges to the optimal set of solutions to (1).

Proof. The proof in this section follows the proof in (Kumar et al., 2020).

We demonstrate that the points $\mathbf{w}^{(\infty)}$, $\mathbf{U}^{(\infty)}$, and $\boldsymbol{\lambda}^{(\infty)}$ satisfy the KKT conditions of problem (1). First, to apply the KKT conditions we show that the linear independence constraint qualification (LICQ) is satisfied for the nonconvex unitary constraint $(\mathbf{U}^{(t)})^\top \mathbf{U}^{(t)} = \mathbf{I}$. Consider the set $\mathcal{S}_U = \{\mathbf{U} \in \mathbb{R}^{m\times n} | \mathbf{U}^\top \mathbf{U} = \mathbf{I}\}$. The set can be rewritten as

$$\mathcal{S}_U = \Big\{ \mathbf{U} \in \mathbb{R}^{m\times n} | g_{ij}(\mathbf{U}) = \sum_{k=1}^{p} U_{ki}U_{kj} - \delta_{ij} \quad \forall 1 \le i \le j \le n \Big\}$$

where $\delta_{ij} = 1$ when $i = j$ and 0 otherwise. Then we observe that $\nabla g_{ii}(\mathbf{U})$ has all zero columns except the ith column, which is $\mathbf{U}_i$, the ith column of $\mathbf{U}$. However, $\nabla g_{ij}(\mathbf{U})$ for $i \neq j$ has all zero columns except the ith and jth, which are $\mathbf{U}_j$ and $\mathbf{U}_i$, respectively. This, each $\nabla g_{ij}(\mathbf{U})$ is not able to be expressed as a linear combination of any of the others, so each $\nabla g_{ij}(\mathbf{U})$ is linearly independent.

We can then say that a solution to (1) exists and is bounded because the level set $\{(\mathbf{w}, \mathbf{U}, \boldsymbol{\lambda}) | f(\mathbf{w}, \mathbf{U}, \boldsymbol{\lambda}) \le f(\mathbf{w}^{(0)}, \mathbf{U}^{(0)}, \boldsymbol{\lambda}^{(0)})\}$ is compact, i.e., closed and bounded.

Each limit of the sequence $(\mathbf{w}^{(t)}, \mathbf{U}^{(t)}, \boldsymbol{\lambda}^{(t)})$ satisfies the KKT conditions of problem (1). The Lagrangian function of (1) is

$$\begin{aligned} &- \sum_{i:\lambda_i \neq 0} \log \lambda_i + \operatorname{tr}(\mathbf{S}\mathcal{L}\mathbf{w}) + \alpha\phi(\mathbf{w}) + \frac{\gamma}{2}\|\boldsymbol{\lambda} - \boldsymbol{\mu}\|_2^2 \\ &\quad + \frac{\beta}{2}\|\mathcal{L}\mathbf{w} - \mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top\|_F^2 \\ &\quad - \mathbf{m}^\top \mathbf{w} + \operatorname{tr}(\mathbf{M}^\top(\mathbf{U}^\top\mathbf{U} - \mathbf{I})) \end{aligned}$$

where $\mathbf{m}$ and $\mathbf{M}$ are dual variables.

The KKT conditions of (1) with respect to $\mathbf{w}$ are

$$\begin{aligned} \mathcal{L}^*\mathcal{L}\mathbf{w} - \mathcal{L}^*(\mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top - \beta^{-1}\mathbf{S}) - \beta^{-1}\mathbf{m} &= 0 \\ \mathbf{m}^\top \mathbf{w} &= 0 \\ \mathbf{w} &\ge 0 \\ \mathbf{m} &\ge 0. \end{aligned}$$

Furthermore, the update for $\mathbf{w}$ that solves (3) is derived from the KKT conditions of (3)., which is equivalent to the KKT conditions above for $\mathbf{w}$.

For $\boldsymbol{\lambda}$, the update is derived by the KKT conditions of (5), which are equivalent to the KKT conditions of (1).

The KKT conditions for $\mathbf{U}$ are

$$\begin{aligned} \mathbf{U}\operatorname{diag}(\boldsymbol{\lambda})^2 - \mathcal{L}\mathbf{w}\mathbf{U}\operatorname{diag}(\boldsymbol{\lambda}) + \frac{1}{2\beta}(\mathbf{M} + \mathbf{M}^\top) &= 0 \\ \mathbf{U}^\top\mathbf{U} &= \mathbf{I}. \end{aligned}$$

Furthermore, we have from (Kumar et al., 2020) that

$$\begin{aligned} \mathcal{L}\mathbf{w}\mathbf{U}\operatorname{diag}(\boldsymbol{\lambda}) - \mathbf{U}\Big(&\mathbf{U}^\top\mathcal{L}\mathbf{w}\mathbf{U}\operatorname{diag}(\boldsymbol{\lambda}) \\ &- \frac{1}{2}\left(\mathbf{U}^\top\mathcal{L}\mathbf{w}\mathbf{U}\operatorname{diag}(\boldsymbol{\lambda}) - \operatorname{diag}(\boldsymbol{\lambda})\mathbf{U}^\top\mathcal{L}\mathbf{w}\mathbf{U}\right)\Big) \end{aligned}$$

due to $\mathbf{U}$ admitting the first order optimality condition on the orthogonal Stiefel manifold. Since $\mathbf{U}^\top \mathcal{L}_{\mathbf{w}} \mathbf{U}$ is a diagonal matrix from the update of $\mathbf{U}$, there must be a $\mathbf{M}$ such that $\mathbf{U}$ satisfies the KKT conditions. Therefore, all three variables, and thus the tuple of iterates $(\mathbf{w}^{(t)}, \mathbf{U}^{(t)}, \boldsymbol{\lambda}^{(t)})$ satisfy the KKT conditions for (1).

6. Numerical Experiments

We demonstrate the efficacy of our proposed method for inferring a synthetic Gaussian graphical model with prior statistical information. We generate networks from the graphon $W(x, y) = \frac{1}{2}(x^2 + y^2)$ of $p = 30$ nodes. First, we consider the effects of applying the spectral penalty given the underlying graphon. Figure 2 (a) demonstrates the results of applying (1) with the known graphon degree distribution. We observe that the applying the spectral penalty exhibits improved performance in estimation.

Note that the underlying graphon may not be available in practice. We also demonstrate the application of our method with an approximation of the graphon degree distribution, i.e., the degree vector of a similar graph. In Figure 2 (b), the application of the spectral penalty with an approximation of the graphon degree distribution also demonstrates improved performance, validating our proposed method in more realistic circumstances.

7. Conclusion

In this paper we demonstrate a modification to the classic graphical lasso algorithm to incorporate prior structural information for improved graph estimation error. We provided an algorithm for solving the objective with spectral constraints via a block MM method and presented a proof of convergence. Numerical experiments were conducted to evaluate the efficacy of the method in practice and observed an improved inference error when the true graphon is known a priori and in a more realistic scenario with an approximated graphon distribution.

We plan to conduct more numerical experiments to better understand the efficacy of the algorithm with different numbers of nodes. Also, we plan to use the algorithm on real datasets to test the practical benefits of our method. Lastly, we plan to extend the algorithm to encompass a joint inference scheme, meaning we infer a set of graphs sampled from the same graphon rather than a single graph.

References

Bach, F., Jenatton, R., Mairal, J., and Obozinski, G. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 2012.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Diaconis, P. and Janson, S. Graph limits and exchangeable random graphs. *arXiv preprint arXiv:0712.2749*, 2007.

Erdős, P. and Rényi, A. On random graphs I. *Publicationes Mathematicae*, 1959.

Friedman, J., Hastie, T., and Tibshirani, R. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

Hao, D., Ren, C., and Li, C. Revisiting the variation of clustering coefficient of biological networks suggests new modular structure. *BMC Systems Biology*, 6(1):1–10, 2012.

Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

Kumar, S., Ying, J., de Miranda Cardoso, J. V., and Palomar, D. P. A unified framework for structured graph learning via spectral constraints. *J. Mach. Learn. Res.*, 21(22): 1–60, 2020.

Lovász, L. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.

Luo, X. A hierarchical graphical model for big inverse covariance estimation with an application to fMRI. *arXiv preprint arXiv:1403.4698*, 2014.

Navarro, M., Wang, Y., Marques, A. G., Uhler, C., and Segarra, S. Joint inference of multiple graphs from matrix polynomials. *arXiv preprint arXiv:2010.08120*, 2020.

Roddenberry, T. M., Navarro, M., and Segarra, S. Network topology inference with graphon spectral penalties. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5390–5394. IEEE, 2021.

Slawski, M., zu Castell, W., and Tutz, G. Feature selection guided by structural information. *The Annals of Applied Statistics*, 4(2):1056–1080, 2010.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

Vizuete, R., Garin, F., and Frasca, P. The Laplacian spectrum of large graphs sampled from graphons. *IEEE Transactions on Network Science and Engineering*, 8(2):1711–1721, 2021. doi: 10.1109/TNSE.2021.3069675.

Yang, S., Sun, Q., Ji, S., Wonka, P., Davidson, I., and Ye, J. Structural graphical lasso for learning mouse brain connectivity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1385–1394, 2015.

Zhu, Y., Schaub, M. T., Jadbabaie, A., and Segarra, S. Network inference from consensus dynamics with unknown parameters. *IEEE Transactions on Signal and Information Processing over Networks*, 6:300–315, 2020.

Accelerated Methods for Non-Convex Optimization in Machine Learning

Xiaochen Long [*1] **Yangfan Ren** [*1] **Peng Yang** [*1]

Abstract

Non-convex problems have received considerable attention in optimization and machine learning in recent years. However, solving non-convex problems could be NP-hard and computationally challenging. Therefore, acceleration for non-convex optimization has become an interesting topic. This report reviews two papers about accelerated methods (momentum) for non-convex optimization. The first paper explores a general non-convex optimization setting and proposes a first-order method that converges to a second-order stationary point with acceleration compared to gradient descent. The second paper focuses on two specific deep neural networks and theoretically proves an accelerated linear convergence rate using Polyak's momentum gradient descent.

1. Introduction

A non-convex optimization problem is any problem where either the objective or any of the constraints are non-convex and it involves a wide range of study (e.g. machine learning, signal processing, etc.). However, the most common optimizers (e.g. gradient descent) for minimizing a non-convex optimization problem is challenging, due to the fact the objective function may have multiple local minimas, saddle points, very flat regions and widely varying curvatures. Moreover, it is computationally intractable to find a global solution for this kind of problems, hindering the application to large-scale datasets. In general, there is no method that can solve this problem effectively in all cases, but several methods published in the literature are trying to address it in different scenarios and improve the learning rate (Carmon et al., 2018; Wang et al., 2021b; Kim et al., 2021).

There are substantial work on the convergence properties of optimization methods for non-convex problems in machine learning. Some recent work focuses on achieving stronger local optima than the simple first-order stationary points (Anandkumar & Ge, 2016; Ge et al., 2015; Lee et al., 2016). Other research investigates the acceleration on convergence rate (Allen-Zhu & Hazan, 2016; Reddi et al., 2016) and one of the most popular techniques is to incorporate momentum (Ghadimi & Lan, 2016; Li & Lin, 2015). Nesterov and Polyak (Nesterov & Polyak, 2006) introduce the Hessian information to achieve a better convergence rate by the cubic-regularized Newton method.

The main motivation of our review is to introduce two state-of-the-art theoretical analysis of accelerated methods for non-convex optimization. In particular, we will cover the two papers:

- Carmon, Yair, et al. "Accelerated methods for nonconvex optimization." SIAM Journal on Optimization 28.2 (2018): 1751-1772.
- Wang, Jun-Kun, Chi-Heng Lin, and Jacob D. Abernethy. "A Modular Analysis of Provable Acceleration via Polyak's Momentum: Training a Wide ReLU Network and a Deep Linear Network." International Conference on Machine Learning. PMLR, 2021.

2. Accelerated gradient methods for non-convex optimization

This paper considers a general optimization problem

$$\min_{x \in \mathbb{R}^d} f(x) \tag{1}$$

where f has both Lipschitz continuous gradient and Hessian, but is not limited to convex functions. Since finding a global optimizer is NP hard, the authors can instead look for stationary points with small enough gradient.

The accelerated method from this paper leverages two competing techniques. The first one is based on the knowledge that moving along the direction of negative curvature for a locally non-convex function will reduce the objective. This technique is widely used in cubic regularization(Nesterov & Polyak, 2006) and escaping from saddle points(Anandkumar & Ge, 2016; Ge et al., 2015). The second one is to apply proximal point techniques(Parikh & Boyd, 2014) and accelerated gradient descent to an almost-convex function

[*]Equal contribution [1]Department of Statistics, Rice University, Houston, USA. Correspondence to: Xiaochen Long <xl81@rice.edu>, Yangfan Ren <yr14@rice.edu>, Peng Yang <py11@rice.edu>.

Project report for COMP414/514, Rice CS.

to obtain acceleration. This is the more important part to achieve faster convergence rates for this method. In addition, the authors use PCA to approximate eigenvector faster.

2.1. Two structured non-convex problems

Roughly, the overall algorithm will be to alternate between finding directions of negative curvature of f and solving structured sub-problems that are nearly convex, meaning that the smallest eigenvalue of the Hessian has a lower bound γ, $\gamma > 0$, where $\gamma \ll L_1$. This paper turn to each of these pieces in turn.

2.1.1. Exploiting negative curvature

The idea of exploiting the negative curvature is to find a neighborhood where f is almost convex.

Definition 1 (Almost convexity) *A function f is σ_1-*strongly convex *if $\frac{\sigma_1}{2}\|y-x\|^2 \le f(y)-f(x)-\nabla f(x)^T(y-x)$ for some $\sigma_1 \in \mathbb{R}$. For $\gamma = \max\{-\sigma_1, 0\}$, we call such functions γ-*almost convex.

Lemma 2.1 *Suppose f is $(-\sigma_1)$-strongly convex, where $\sigma_1 \ge 0$. Then for any $x_0 \in \mathbb{R}^d$ the function $g(x) = f(x) + \sigma_1\|x - x_0\|^2$ is (σ_1)-strongly convex.*

By these definitions, one can easily know that $\lambda_{min}(\nabla^2 f(x)) \le -\alpha \implies \alpha$-almost convexity. Using this property, this paper developed an algorithm to reach almost convex neighborhood.

Algorithm 1 Negative-Curvature-Descent

Input: z_1, f, L_2, α, Δ_f, δ
Set $\delta' = \delta/(1 + 12L_2^2\Delta_f/\alpha^3)$
for $i = 1, 2, \ldots$ **do**
  Find a vector v_j such that $\|v_j\| = 1$ and, with probability at least $1 - \delta'$,

$$\lambda_{\min}(\nabla^2 f(z_j)) \ge v_j^T \nabla^2 f(z_j) v_j - \alpha/2$$

  using a leading eigenvector computation
  if $v_j^T \nabla^2 f(z_j) v_j \le -\alpha/2$ **then**
    Let

$$z_{j+1} \leftarrow z_j - \frac{2|v_j^T \nabla^2 f(z_j) v_j|}{L_2} sign(v_j^T \nabla f(z_j)) v_j$$

  else
    Return z_j
  end if
end for

For the leading eigenvector computation, the Lanczos method(Kuczynski & Wozniakowski, 1992) achieves the following guarantee:

Lemma 2.2 (Accelerated top eigenvector computation) *Let $H \in \mathbb{R}^{d\times d}$ be symmetric and PSD. There exists an algorithm that on input $\varepsilon, \delta \in (0, 1)$ runs in $O(L_2 \log(d/\delta)\varepsilon^{-1/2})$ time and, with probability at least $1 - \delta$, returns a relative ε-approximate leading eigenvector $\hat{v}$.*

Furthermore, the time complexity of the Negative-Curvature-Descent is given below in this lemma:

Lemma 2.3 *Let the function f be L_1-smooth and have L_2-Lipschitz continuous Hessian, $\alpha > 0$, $0 < \delta < 1$ and $z_1 \in \mathbb{R}^d$. If we call Negative-Curvature-Descent(z_1, f, L_2, α, Δ_f, δ) then the algorithm terminates at iteration j for some*

$$j \le 1 + \frac{12L_2^2(f(z_1) - f(z_j))}{\alpha^3} \le 1 + \frac{12L_2^2\Delta_f}{\alpha^3}, \tag{2}$$

and with probability at least $1 - \delta$

$$\lambda_{min}(\nabla^2 f(z_j)) \ge -\alpha. \tag{3}$$

Furthermore, each iteration requires time at most

$$O\left(T_{grad}\left[1 + \sqrt{\frac{L_1}{\alpha}} \log\left(\frac{d}{\delta}\left(1 + 12\frac{L_2^2\Delta_f}{\alpha^3}\right)\right)\right]\right). \tag{4}$$

where T_{grad} is defined below:

Assumption 1 *The following operations take $O(T_{grad})$ time:*

1. *The evaluation $\nabla f(x)$ for a point $x \in \mathbb{R}^d$.*
2. *The evaluation of $\nabla^2 f(x)v$ for some vector $v \in \mathbb{R}^d$ and point $x \in \mathbb{R}^d$.*
3. *Any arithmetic operation (addition, subtraction or multiplication) of two vectors of dimension at most d.*

Therefore, by applying the Negative-Curvature-Descent method, the algorithm can find a neighborhood around z_j where f is α-almost convex with complexity guaranteed by the bound above. The next step is to find a stationary point in the α-almost convex neighborhood.

2.1.2. Accelerated gradient descent for almost convex functions

This paper uses the Nesterov's accelerated momentum method:

Algorithm 2 Accelerated-Gradient-Descent

Input: f, y_1, ϵ, L_1, σ_1
Set $\kappa = L_1/\sigma_1$ and $z_1 = y_1$
for $j = 1, 2, \ldots$ **do**
  if $\|\nabla f(y_j)\| \leq \epsilon$ **then**
    Return y_j
  end if
  Let

$$y_{j+1} = z_j - \frac{1}{L_1}\nabla f(z_j)$$
$$z_{j+1} = \left(1 + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) y_{j+1} - \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} y_j$$

end for

Note that this is a nonconvex case (although almost convex), so the authors don't directly implement the method for optimization. Instead of miniming f, given the α-almost convexity, the paper considered the convex optimization problem:

$$\min_z g_j(z) := f(z) + \gamma\|z - z_j\|^2$$

where z_j is the jth iteration. When $g_j(z)$ converges, the authors show that $\|\nabla f(z)\|$ also converges. The algorithm for convergence in almost convex neighborhood is named Almost Convex AGD:

Algorithm 3 Almost-Convex-AGD

Input: f, z_1, ϵ, γ, L_1
for $j = 1, 2, \ldots$ **do**
  if $\|\nabla f(z_j)\| \leq \epsilon$ **then**
    Return z_j
  end if
  Let $g_j(z) = f(z) + \gamma\|z - z_j\|^2$
  $\epsilon' = \epsilon\sqrt{\gamma/(50(L_1 + 2\gamma))}$
  $z_{j+1} \leftarrow$ Accelerated-Gradient-Descent(g_j, z_j, ϵ', L_1, γ)

end for

Lemma 2.4 *Let f be be $\min\{\sigma_1, 0\}$-almost convex and L_1-smooth. Let $\gamma \geq \sigma_1$ and let $0 < \gamma \leq L_1$. Then ALMOST-CONVEX-AGD(f,z_1,ϵ,γ,L_1) returns a vector z such that $\|\nabla f(z)\| \leq \epsilon$ and*

$$f(z_1) - f(z) \geq \min\left\{\gamma\|z - z_1\|^2, \frac{\epsilon}{\sqrt{10}}\|z - z_1\|\right\} \quad (5)$$

in time

$$O\left(T_{grad}\left(\sqrt{\frac{L_1}{\gamma}} + \frac{\sqrt{\gamma L_1}}{\epsilon^2}(f(z_1) - f(z))\right)\log\left(2 + \frac{L_1^3\Delta_f}{\gamma^2\epsilon^2}\right)\right). \quad (6)$$

With these tools above with the guarantee of convergence, the paper proposes a leveraging method for smooth non-linear optimization.

2.2. Acceleration of smooth non-linear optimization

Given the two subroutines, the idea of the accelerated method for non-convex optimization can be illustrated by the following steps :

- Use NEGATIVE-CURVATURE-DESCENT to find a point $\hat{x}_k$ where the function is locally α-almost convex.
- Add a convex penalty $\rho_\alpha(x) := L_1\left[\|x\| - \frac{\alpha}{L_2}\right]_+$ on the function $f(x)$ where $[t]_+ = \max\{t, 0\}$ and obtain $f_k(x) = f(x) + \rho_\alpha(x - \hat{x}_k)$ which is proved to be globally 3α-almost convex and $5L_1$-smooth.
- Apply ALMOST-CONVEX-AGD on the function $f_k(x)$ to reduce $f(x)$ efficiently.

The procedure is iterated until reaching a point with our desired accuracy. The detailed steps are shown in Algorithm 4.

The parameter α controls the extent of non-convexity which can be chosen specifically to obtain an accelerated convergence rate compared with gradient descent on non-convex function. Theorem 2.5 indicates that this algorithm can converge to a second-order stationary point in time polynomial in the desired accuracy with logarithmic dependence on the problem dimension.

Theorem 2.5 *Assume function $f : \mathbb{R}^d \to \mathbb{R}$ has L_1-Lipschitz continuous gradient and L_2-Lipschitz continuous Hessian, but may be non-convex. Then with probability at least $1 - \delta$, ACCELERATED-NON-CONVEX-METHOD $(x_1, f, \epsilon, L_1, L_2, \alpha, \Delta_f, \delta)$ returns a second-order stationary point in desired accuracy ϵ with $\alpha = min\{L_1, max\{\epsilon^2\Delta_f^{-1}, \epsilon^{1/2}L_2^{1/2}\}$ and $\delta \in (0, 1)$. Specifically, the stationary point x satisfies*

$$\|\nabla f(x)\| \leq \epsilon \quad and \quad \lambda_{min}(\nabla^2 f(x)) \geq -2\epsilon^{1/2}L_2^{1/2}$$

where λ_{min} is the smallest eigenvalue. The convergence rate is

$$O\left(T_{grad}\left(\Delta_f L_1^{1/2}L_2^{1/4}\epsilon^{-7/4} + \Delta_f^{1/2}L_1^{1/2}\epsilon^{-1} + 1\right)log\tau\right),$$

where $\tau = 1 + 1/\epsilon + 1/\delta + d + L_1 + L_2 + \Delta_f$ and T_{grad} is defined in Assumption 1.

If the accuracy ϵ is small enough, i.e., smaller or equal to $(\Delta_f^2/L_2)^{-1/3}$, then the result can be simplified with convergence rate to be at most $\tilde{O}(T_{\text{grad}}\Delta_f\frac{L_1^{1/2}L_2^{1/4}}{\epsilon^{7/4}})$.

Algorithm 4 Acceleration of smooth non-linear optimization (ACCELERATED-NON-CONVEX-METHOD)

Input: $x_1, f, \epsilon, L_1, L_2, \alpha, \Delta_f, \delta$
Set $K := \lceil 1 + \Delta_f(12L_2^2/\alpha^3 + \sqrt{10}L_2/(\alpha\epsilon))\rceil$
Set $\delta := \frac{\delta}{K}$
for $k = 1, 2, \ldots$ **do**
  if $\alpha < L_1$ **then**
    $\hat{x}_k \leftarrow \text{NCD}(x_k, f, L_2, \alpha, \Delta_f, \delta)$
  else
    $\hat{x}_k \leftarrow x_k$
  end if
  if $\|\Delta_f(\hat{x}_k) \le \epsilon$ **then**
    return $\hat{x}_k$ {guarantees w.h.p., $\lambda_{\min}(\nabla^2 f(\hat{x}_k)) \ge -2\alpha$}
  end if
  Set $f_k(x) = f(x) + L_1([\|x - \hat{x}_k\| - \alpha/L_2]_+)^2$
  $\hat{x}_{k+1} \leftarrow \text{ALMOST-CONVEX-AGD}(f_k, \hat{x}_k, \epsilon/2, 3\alpha, 5L_1)$
end for

where NCD denotes NEGATIVE-CURVATURE-DESCENT.

Table1 compares the running time with related algorithms. Among five algorithms, three of them could target non-convex functions. Algorithm4 has an obvious acceleration compared to gradient descent. Although the convergence rate of cubic-regularized Newton method is faster in terms of number of gradient calculations, it requires either explicitly or approximately calculation of Hessian matrix in each iteration which increases computational cost. Although a number of researchers work on approximate Hessian methods which provide satisfied empirical results(Bianconcini et al., 2015; Cartis et al., 2011), they do not improve on the complexity of gradient descent.

In summary, there are two main advantages of ACCELERATED-NON-CONVEX-METHOD. The first one is the acceleration on the complexity of gradient descent in terms of number of gradient calculations with only first-order information. Therefore, the method is Hessian free which avoids the computational cost from Hessian calculations that second-order methods normally require. The second one is the convergence to second-order stationary points which are better approximations of local minimizers. Another notable result is the modification on strict-saddle functions (Lee et al., 2016; Ge et al., 2015) which achieves linear convergence rates.

3. Acceleration via Polyak's Momentum for deep neural network

Incorporating a so-called "momentum" dynamic in gradient descent methods is widely used in training neural networks in various applications (He et al., 2016; Vaswani et al., 2017; Krizhevsky et al., 2012). The most popular one, among all the momentum methods, seems to be Polyak's momentum (Polyak, 1964) as shown in Algorithm 5, which is the default choice of momentum in PyTorch and Tensorflow. In the algorithm, if $\beta = 0$, it essentially performs a gradient descent. When $\beta \neq 0$, the momentum accelerate the optimization process by the direction from the pervious step to the current. The success of Polyak's momentum has been widely recognized and been adopted to many recently developed adaptive gradient methods like Adam(Kingma & Ba, 2014), AMSGrad (Reddi et al., 2019), and AdaBound(Luo et al., 2019).

Algorithm 5 Gradient descent with Polyak's momentum

Required: step size η and momentum parameter β.
Init: $w_0 = w_{-1} \in \mathbb{R}^d$
for $t = 0$ to T **do**
  Given current iterate w_t, obtain graident $\nabla l(w_t)$
  update iterate $w_{t+1} = w_t - \eta\nabla l(w_t) + \beta(w_t - w_{t-1})$
end for

However, despite its popularity, little is known in theory that explains why Polayk's momentum would result in acceleration of training deep neural network. Recent work (Wang et al., 2021a) shows that Polyak's momentum helps escape saddle points faster compared with the case without momentum, which seems the only provable advantage of Polyak's momentum in non-convex optimization. Therefore, we are focusing on the understand of theoretical performance of Polyak's momentum method on a specific non-convex problem, which is deep neural network.

3.1. One-layer ReLU network

To maintain consistency, we follow the same framework as previous results(Du et al., 2018; Arora et al., 2019; Song & Yang, 2019) of studying one-hidden layer ReLU neural network and it follows the form,

$$N_W^{ReLU}(x) := \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma(\langle w^{(r)}, x\rangle), \tag{7}$$

where $\sigma(z) := z \cdot 1\{z \ge 0\}$ is the ReLU activation function, $w^{(1)}, \ldots, w^{(m)} \in \mathbb{R}^d$ are the weights of m neurons on the fist layer, $a_1, \ldots, a_m \in \mathbb{R}$ are weights on the second layer, and $N_W^{ReLU}(x)$ is the output predicted on input x. Assume n number of samples $\{x_i \in \mathbb{R}\}_{i=1}^n$ is given. Following previous work(Du et al., 2018; Arora et al., 2019; Song & Yang, 2019), we define a Gram matrix $H \in \mathbb{R}^{n\times n}$ for the weights W and its expectation $\bar{H} \in \mathbb{R}^{n\times n}$ over the random draws

Table 1. Running time comparisons for finding a first-order stationary point.

	# OF ITERATIONS	HESSIAN FREE?	GRADIENT LIPSCHITZ?	HESSIAN LIPSCHITZ?	CONVEX f?
GRADIENT DESCENT (NON-CONVEX CASE)	$O(\Delta_f L_1 \epsilon^{-2})$	YES	YES	NO	NO
GRADIENT DESCENT (CONVEX CASE)	$O(RL_1\epsilon^{-1})$	YES	YES	NO	YES
PROXIMAL ACCELERATED GRADIENT DESCENT	$\tilde{O}((RL_1)^{\frac{1}{2}}\epsilon^{-\frac{1}{2}})$	YES	YES	NO	YES
CUBIC-REGULARIZED NEWTON METHOD	$\tilde{O}(\Delta_f L_2^{\frac{1}{2}}\epsilon^{-\frac{3}{2}})$	NO	YES	YES	NO
ALGORITHM 1	$\mathbf{\tilde{O}(\Delta_f L_1^{\frac{1}{2}} L_2^{\frac{1}{4}} \epsilon^{-\frac{7}{4}})}$	**YES**	**YES**	**YES**	**NO**

$w^{(r)} \sim N(0, I_d) \in \mathbb{R}^d$ whose i, j entries are defined as follow,

$$H(W)_{i,j} := \sum_{r=1}^{m} \frac{x_i^T x_j}{m} 1\{\langle w^{(r)}, x_i\rangle \geq 0 \ \& \ \langle w^{(r)}, x_j\rangle\}$$
$$\bar{H}_{i,j} := E_{w^{(r)}}[x_i^T x_j 1\{\langle w^{(r)}, x_i\rangle \geq 0 \ \& \ \langle w^{(r)}, x_j\rangle\}].$$

Previous work (Du et al., 2018) shows the dynamics of prediction space is governed by the spectral property of Gram matrix (which can vary in each iteration) and as long as the least eigenvalue is lower bounded, gradient descent enjoys a linear rate. Further, the Gram matrix at later iterations is close to that in the initialization phase. Along with the observation that Gram matrix is only related to the activation patterns ($1\{w_r^T w_i\}$), the Gram matrix is close to its initialization. Then this property give us the insights to prove the convergence results on ReLU activated neural networks.

Prior work (Du et al., 2018; Wu et al., 2019) showed that using vanilla gradient descent for one-layer wide ReLU network with a step size $\eta = \frac{1}{c_1 \lambda_{min}(\bar{H})}$ leads to a convergence rate $(1 - \frac{1}{c_2 \kappa'})$ for some quantities $c_1, c_2 > 0$, where κ' is the condition number of a Gram matrix. However, the quantities c_1 and c_2 are not universal constants and actually depend on the problem parameter (e.g. $\lambda_{min}(\bar{H}), n, \delta$). Therefore, we aim to improve this dependency in this work.

Theorem 3.1 *Assume that $\lambda := \frac{3\lambda_{min}(\bar{H})}{4} > 0$ and that $w_0^{(r)} \sim N(0, I_d)$ and a_r uniformly sampled from $\{-1, 1\}$. Denote $\lambda_{max} := \lambda_{max}(\bar{H}) + \frac{\lambda_{min}(\bar{H})}{4}$, $\kappa := \frac{\lambda_{max}(\bar{H})}{\lambda_{min}(\bar{H})}$, and denote $\hat{\kappa} := \lambda_{max}/\lambda = 4(\kappa + 1)/2$. Set a constant step size $\eta = \frac{1}{\lambda_{max}}$, fix momentum parameter $\beta = (1 - \frac{1}{2\sqrt{\hat{\kappa}}})^2$, and the final set the number of network nodes $m = \Omega(\lambda^{-4} n^4 \kappa^2 log^3(n/\delta))$. Then, with probability at least $1 - \delta$ over the random initialization gradient descent with Polyak's momentum staisfies for any t,*

$$\|\xi_t, \xi_{t-1}\| \leq (1 - \frac{1}{4\sqrt{\hat{\kappa}}})^t 8\sqrt{\hat{\kappa}} \|\xi_0, \xi_{-1}\|. \tag{8}$$

Therefore, Theorem 3.1 essentially shows an accelerated linear rate $(1 - \Theta(\frac{1}{\sqrt{\kappa}}))$ and this rate has an improved dependency on the condition number.

3.2. Deep linear Network

In this paper, following previous work(Du & Hu, 2019; Hu et al., 2020), we studied training a $L-$layer linear network of the form,

$$N_W^{L-linear}(x) := \frac{1}{\sqrt{m^{L-1} d_y}} W^{(L)} W^{(L-1)} \dots W^{(1)} x, \tag{9}$$

where $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ is the weight matrix of the layer $l \in [L]$, and $d_0 = d$, $d_L = d_y$ and $d_l = m$ for $l \neq 1, L$. Therefore, except the fist layer $W^{(1)} \in \mathbb{R}^{m \times d}$ and the last layer $W^{(L)} \in \mathbb{R}^{d_y \times m}$, all the intermediate layers are $m \times m$ square matrics.

Similarly, prior work(Hu et al., 2020) shows with proper choice of step size $\eta = \frac{d_y}{2L\sigma_{max}^2(X)}$, it leads to a linear convergence rate with error decays $(1 - \frac{1}{\kappa})^t$ after t iterations. In this work, we aim to investigate the theoretical performance of Polay's momentum method on deep linear network.

Theorem 3.2 *Denote $\lambda := \frac{L\sigma_{min}^2(X)}{d_y}$ and $\kappa := \frac{\sigma_{max}^2}{\sigma_{min}^2(X)}$. Set a constant step size $\eta = \frac{d_y}{2L\sigma_{max}^2(X)}$, fix momentum parameter $\beta = (1 - \frac{1}{2\sqrt{\kappa}})^2$, and finally set a parameter m that controls the width $m \geq C\frac{\kappa^5}{\sigma_{max}^2(X)}(d_y(1 + \|W^*\|_2^2 + log(\bar{r}/\delta)))$ and $m \geq max\{d_x, d_y\}$ for some constant $C > 0$. Then, with probability at least $1 - \delta$ over the random orthogonal initialization, gradient descent with Polyak's momentum satisfies for any t,*

$$\|\xi_t, \xi_{t-1}\| \leq (1 - \frac{1}{4\sqrt{\kappa}})^t 8\sqrt{\kappa} \|\xi_0, \xi_{-1}\|. \tag{10}$$

Therefore, Theorem 3.2 essentially shows an accelerated linear rate $(1 - \Theta(\frac{1}{\sqrt{\kappa}}))$ and this result also suggests that the depth does not hurt optimization.

4. Discussion

In this report, we review two papers about theoretical analysis of accelerated methods (momentum) for non-convex problems. Non-convex optimization is an interesting topic as it provides flexibility and modeling power. On the one hand, higher demands of high dimensional spaces in applications require some structural constraints on the learning models such as sparsity and low-rank which are often non-convex. On the other hand, the objective of some learning tasks is naturally a non-convex function including training deep neural networks and tensor decomposition problems. The simplest method for obtaining a stationary point of non-convex objective is gradient descent. In recent years, there are substantial literature working on accelerated techniques which are successful in practice. However, it is challenging to provide theoretical guarantees for their convergence properties.

The first paper we review carefully combines of two subroutines to achieve a better convergence rate. The crux of the fast algorithm is a variant of Nesterov's classical accelerated gradient descent (momentum) under the almost-convex assumption. The significance of the method lies in the acceleration convergence rates for finding a second-order stationary point with first-order information. Nevertheless, the updating rule in negative curvature descent might not have been sufficiently studied and optimized. Moreover, as the method is Hessian-free on the ALMOST-CONVEX-AGD step, it still requires information from the Hessian matrix while calculating the negative curvature. The space for improvements from better descent methods in this part remains to be explored in the future.

The second paper we reviewed mainly discusses why the momentum-based method would result in accelerated convergence results for training the deep neural networks. Previous work only shows linear convergence rate by using vanilla gradient for training one-layer ReLU network and deep linear network (without activation function). By properly choosing learning rate and momentum parameter, the paper theoretically proved an accelerated linear convergence rate using Polyak's momentum gradient descent. Even though these two networks are perhaps the most popular canonical models for studying optimization and deep learning in the literature, the theoretical performance remains unknown for deep networks (with activation functions). Besides, there are more widely used adaptive gradient descent methods like Adam (Kingma & Ba, 2014), AMSGrad (Reddi et al., 2019) and AdaBound (Luo et al., 2019) for training deep neural network. However, the theoretical guarantees of these methods are only worse if the momentum parameter is non-zero and it deteriorates as the momentum parameter increase for convex problem (Alacaoglu et al., 2020). Therefore, it is worth investigating the convergence performance of adaptive related gradient descent methods for non-convex cases.

In addition to the methods we present in this report, there is also literature discussing other accelerated methods on non-convex problems. For methods applied to general non-convex smooth functions, Agarwal et al. (Agarwal et al., 2016) propose *FastCubic* algorithm, a second-order method based on the cubic-regularized Newton method. The convergence rate is proved to be identical to that from the ACCELERATED-NON-CONVEX-METHOD(Carmon et al., 2018). However, Jin et al (Jin et al., 2018) mentions that these methods (Agarwal et al., 2016; Carmon et al., 2018) both rely on more complex mechanisms and propose a simple momentum-based algorithm (PAGD for "perturbed AGD") which achieves the same convergence rates.

Besides the effort in general non-convex situations, some non-convex accelerated methods are developed for specific problems. Kim et al (Kim et al., 2021) proposed Momentum-Inspired Factored Gradient Descent (MiFGD), which extends the applicability of quantum tomography for larger systems. The method converges "provably" to the true density matrix at a linear rate, in the absence of experimental and statistical noise, and under common assumptions. Allen-Zhu (Allen-Zhu, 2017) proposed *Katyusha* momentum, a novel "negative momentum" on top of Nesterov's momentum for non-convex optimization in stochastic settings (SGD), which can also be incorporated into a variance-reduction based algorithm and speed it up, both in terms of sequential and parallel performance. This also sheds light on the spaces for improvement by combining momentum and SGD.

References

Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding local minima for nonconvex optimization in linear time. *arXiv preprint arXiv:1611.01146*, 2016.

Alacaoglu, A., Malitsky, Y., Mertikopoulos, P., and Cevher, V. A new regret analysis for adam-type algorithms. In *International Conference on Machine Learning*, pp. 202–210. PMLR, 2020.

Allen-Zhu, Z. The first direct acceleration of stochastic gradient methods. *Journal of Machine Learning Research*, 18(1):8194–8244, 2017.

Allen-Zhu, Z. and Hazan, E. Variance reduction for faster non-convex optimization. In *International conference on machine learning*, pp. 699–707. PMLR, 2016.

Anandkumar, A. and Ge, R. Efficient approaches for escaping higher order saddle points in non-convex optimization. In *Conference on learning theory*, pp. 81–102. PMLR, 2016.

Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pp. 322–332. PMLR, 2019.

Bianconcini, T., Liuzzi, G., Morini, B., and Sciandrone, M. On the use of iterative methods in cubic regularization for unconstrained optimization. *Computational Optimization and Applications*, 60(1):35–57, 2015.

Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018.

Cartis, C., Gould, N. I., and Toint, P. L. Adaptive cubic regularisation methods for unconstrained optimization. part i: motivation, convergence and numerical results. *Mathematical Programming*, 127(2):245–295, 2011.

Du, S. and Hu, W. Width provably matters in optimization for deep linear neural networks. In *International Conference on Machine Learning*, pp. 1655–1664. PMLR, 2019.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pp. 797–842. PMLR, 2015.

Ghadimi, S. and Lan, G. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 156(1-2):59–99, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, W., Xiao, L., and Pennington, J. Provable benefit of orthogonal initialization in optimizing deep linear networks. *arXiv preprint arXiv:2001.05992*, 2020.

Jin, C., Netrapalli, P., and Jordan, M. I. Accelerated gradient descent escapes saddle points faster than gradient descent. In *Conference On Learning Theory*, pp. 1042–1085. PMLR, 2018.

Kim, J. L., Kollias, G., Kalev, A., Wei, K. X., and Kyrillidis, A. Fast quantum state reconstruction via accelerated nonconvex programming. *arXiv preprint arXiv:2104.07006*, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Kuczynski, J. and Wozniakowski, H. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM journal on matrix analysis and applications*, 13(4):1094–1122, 1992.

Lee, J. D., Simchowitz, M., Jordan, M. I., and Recht, B. Gradient descent only converges to minimizers. In *Conference on learning theory*, pp. 1246–1257. PMLR, 2016.

Li, H. and Lin, Z. Accelerated proximal gradient methods for nonconvex programming. *Advances in neural information processing systems*, 28:379–387, 2015.

Luo, L., Xiong, Y., Liu, Y., and Sun, X. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

Nesterov, Y. and Polyak, B. T. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

Parikh, N. and Boyd, S. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Reddi, S. J., Hefny, A., Sra, S., Poczos, B., and Smola, A. Stochastic variance reduction for nonconvex optimization. In *International conference on machine learning*, pp. 314–323. PMLR, 2016.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Song, Z. and Yang, X. Quadratic suffices for overparametrization via matrix chernoff bound. *arXiv preprint arXiv:1906.03593*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wang, J.-K., Lin, C.-H., and Abernethy, J. Escaping saddle points faster with stochastic momentum. *arXiv preprint arXiv:2106.02985*, 2021a.

Wang, J.-K., Lin, C.-H., and Abernethy, J. D. A modular analysis of provable acceleration via polyak's momentum: Training a wide relu network and a deep linear network. In *International Conference on Machine Learning*, pp. 10816–10827. PMLR, 2021b.

Wu, X., Du, S. S., and Ward, R. Global convergence of adaptive gradient methods for an over-parameterized neural network. *arXiv preprint arXiv:1902.07111*, 2019.

A Brief Survey of Random Features for Kernel Approximations

Boyuan Deng [*1] **Erin Liu** [*1]

Abstract

This literature review gives a survey of the various methods to approximate kernel functions using random features. Since Rahimi and Racht published their paper to introduce the method of Random Fourier Feature (RFF) in 2007 (Rahimi & Recht, 2007), it has attracted huge amount of attention in the field. Starting from there, there are so many developments coming out of RFF that it merit a detailed record of the motivation, background and contributions of these random approximation methods. We use the categorization suggested by (Liu et al., 2021a) to divide all RFF variants into two broad categories based on whether training data is taken into account in the training procedure, and introduce two representatives in each category. The highlights and comparisons of these methods will be discussed, and overall this paper seeks to serve as a gentle introduction to this intriguing sub-field of machine learning.

1. Introduction

Kernel method is a well-studied, powerful tool to learn non-linear decision boundaries in supervised machine learning algorithms like Support Vector Machine. The natural solution to the problem of non-linearity is to project the data to another, often much higher-dimensional, space in the hope that they will be linearly separable there. This procedure would easily incur large computation and storage cost in the algorithm: a huge amount of computation is required to project the data, higher-dimensional data become computationally infeasible in vanilla training procedure, and the transformed data require much larger storage space. This easily becomes a problem as the size of the dataset grows.

Another way to view the problem at higher dimensional space is that the information included in inner product of the points, instead of the exact location of the point itself, could be exploited. More specifically, let the projection function be Φ, define the corresponding kernel function $k(x, y) = \langle \Phi(x),\ \Phi(y) \rangle$, with $\langle \cdot,\ \cdot \rangle$ being certain valid inner product in the higher dimensional space related to ϕ. The result of an inner product is a real number, which is much easier to deal with in machine learning algorithms. Another great news is that the projection function Φ become implicit. Instead, inner product in higher-dimensional space is directly calculated by the explicit form of the kernel function $k(x, y)$. In prediction phase, input points also go through the kernel function, and its category is determined by its relative positions to each of the points in the training set, which is also indicated by its inner product with them. In other words, the prediction function looks like $f(x) = \sum_i \alpha_i k(x_i, x)$ with x being the incoming points for prediction and x_i's the points in training set. The theoretical support of this formulation will be explained in later sections.

Some drawbacks are still present, especially when the size of the dataset is extremely large. In the training phase of kernel ridge regression, a covariance matrix is used. This matrix stores the inner product for every two points in the training set, hence would take $O(n^2)$ in storage, and $O(n^3)$ in running time, with n being the number of points in the training set. One intuition to proceed is that, since there is a plethora of information present, an approximation of the kernel function could be carried out, so reasonable results could be produced in a more reasonable time.

Many attempts had been made to train large-scale kernel machines more efficiently. Pure mathematical tools like decomposition-based method (Platt, 1999) has been tried, but with poor scalability. Then people started to notice the power of randomized algorithms, attempting to throw away entries (Achlioptas et al., 2001) or rows (Drineas & Mahoney, 2005) in the covariance matrix. These attempts finally inspired the brilliant idea of Random Fourier Features (RFF), which won the NeurIPS Test-of-Time award in 2017 and the ICML Best Paper Finalist in 2019 (Rahimi & Recht, 2007). The authors of the abovementioned paper came up with the excellent idea of drawing random features from the fourier expansion of the kernel function for approximation, gaining huge success and inspiring a flurry of papers in the next decade. In this literature review, the contributions and

[*]Equal contribution [1]Department of Computer Science, Rice University, Houston, Texas, United States. Correspondence to: Anastasios Kyrillidis <anastasios@rice.edu>.

Project report for COMP414/514, Rice CS.

methods of this RFF paper is throughly analysed.

After the Random Fourier Features has been published, many researchers has attempted to improve or build upon that idea. With the categorization criteria suggested by (Liu et al., 2021b), the methods are divided into two big categories, data independent or data dependent, based on whether they take training data into consideration in their procedures.

In this reivew, we first defines the key mathematical concepts used in the kernel method and introduce some relevant math and statistical ideas for readers to fully appreciate the random feature method. We then discuss the theoretical basis and complexity of the RFF algorithm. Finally, we introduce four representative variants of RFF and compare their theoretical performance.

2. Problem Definition

2.1. Kernel Function

We first give a formal definition of kernel functions. Suppose we are given the empirical data points

$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$$

where $\mathcal{X}$ is some nonempty set from which the inputs x_i (random variables) are taken and $y_i \in \mathcal{Y}$ are the classification labels of its corresponding input x_i. Since we set no restriction on the set $\mathcal{X}$, the observed data points (x_i, y_i) might not be linearly separable. Thus, we project these points to a higher dimension space for easier separation. Then, to avoid the heavy vector computation in higher dimensional spaces, we resort to kernel functions that measure the similarity of the projected points. Notice the space into which we project the data points must be a dot product space, otherwise we won't be able to compute similarity of points by taking their inner products. The projection function is defined as:

$$\Phi : \mathcal{X} \to \mathcal{H}$$

where $\mathcal{H}$ is the projection space, or more commonly known as the feature space. Finally, the kernel function k is defined as:

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

and

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle \tag{1}$$

2.2. Properties of Kernel Functions

2.2.1. Positive definite kernel

Definition 2.1 *A kernel function k is positive definite if*

$$\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0$$

For arbitrary $n \in \mathbb{N}$, $x_i, x_j \in \mathcal{X}$ and $c_i, c_j \in \mathbb{R}$.

Another interpretation of positive definite kernels involves the concept of a Gram matrix:

Definition 2.2 *Given a kernel function k and inputs $x_1, x_2, ..., x_n \in \mathcal{X}$, the $n \times n$ matrix matrix K where $K_{i,j} = k(x_i, x_j)$ is called the Gram matrix of k with respect to $x_1, x_2, ..., x_n$.*

A kernel function k is positive definite if its Gram matrix K with respect to the data points $x_1, x_2, ..., x_n$ is positive definite.

It has been proved by (Hofmann et al., 2008) that a kernel function k satisfies (1) if and only if k is positive definite. In other words, a kernel function that is not positive definite is not qualified to measure similarities of points in the feature space.

2.2.2. Shift invariant kernel

A kernel that is shift invariant only measures the relative distance of 2 points in $\mathcal{X}$. Mathematically, this means:

$$\forall x, y \in \mathcal{X}, k(x, y) = k(x - y, 0) = k(x - y)$$

2.3. The Representer Theorem

We have defined kernel functions to assist the measurement of data point similarities in feature space. But how do we train a classifier using the kernel function? The representer theorem tells us exactly this.

Definition 2.3 *Let k be a kernel defined on $\mathcal{X}$and $\mathcal{F}$ be its reproducing kernel hilbert space. If we are given $x_1, x_2, ..., x_n \in \mathcal{X}$ and $y_1, y_2, ..., y_n \in \mathcal{Y}$ and consider the optimization problem:*

$$\min_{f \in \mathcal{F}} \lambda \|f\|_{\mathcal{F}}^2 + \sum_{i=1}^{n} l(f(x_i), y_i) \tag{2}$$

where l is some arbitrary loss function. If (2) has a minimizer, then it can be written in the form:

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \tag{3}$$

2.4. The Kernel Method

Since equation (2) is exactly the learning objective in a classification problem, we can use the training data points to learn the weights α_i in (3) with gradient descent. Then, to predict the label of an unseen data x, we may simply evaluate (3) on x.

2.5. Example Kernel Functions

We also introduce a few commonly used kernel functions in classification problems (Liu et al., 2021b).

1. Gaussian kernel: one of the most preferred shift invariant kernel function.

$$k(x_i, x_j) = exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

where σ is the kernel width.

2. Polynomial kernel

$$k(x_i, x_j) = (x_i^T x_j + c)^d, c \geq 0$$

where c is some arbitrary constant and d is the degree of the polynomial.

3. Mathematical Background

In this section, we keep on introducing a few more important mathematical concepts that are fundamental to understanding the random feature approximation methods.

3.1. Bochner's Theorem

Theorem 1 *If the function $f : \mathbb{R}^m \rightarrow$ is a positive definite and continuous function, then there is some nonnegative Borel measure p on $\mathbb{R}^m$ such that f is the Fourier transform of p.*

There are 2 terms in it that need a little more explanation. The Fourier Transform is a mathematical transform that decomposes a function into sine waves. Mathematically, the Fourier Transform of f, defined on $\mathbb{R}^m$ can be written as:

$$\hat{f}(\xi) = \int_{\mathbb{R}^m} f(x) e^{-2\pi i x \xi} \, \mathrm{d}\xi \tag{4}$$

The Borel measure is a measure defined on an open set, in this case, the set $\mathbb{R}^m$. We may think of the Borel measure p as a probability distribution defined on $\mathbb{R}^m$.

Now consider some positive definite and shift invariant kernel function $k : \mathbb{R}^m \rightarrow \mathbb{R}$. Bochner's Theorem tells us that there exists a probability distribution p defined on $\mathbb{R}^m$ such that:

$$k(x - y) = \hat{p}(\omega) = \int_{\mathbb{R}^m} p(\omega) e^{-2\pi i \omega (x-y)} \, \mathrm{d}\omega \tag{5}$$

3.2. Importance Sampling

Suppose there is a function $f(x)$ where the input x follows a probabilistic distribution p. Then the expectation of the function could be calculated as:

$$E[f(x)] = \int f(x) p(x) \, dx$$

Suppose we wish to produce an estimate of the expectation, $\hat{E}[f(x)]$, we could geneate n samples, $x_1, \ldots, x_n$, and the estimation could be calculated by

$$\hat{E}[f(x)] = \frac{\sum_{i=1}^{n} f(x_i)}{n}$$

It is an unbiased estimation for $E[f(x)]$.

However, there will be problems if the original distribution, p, have some undesirable properties, such as high variance and/or is hard to compute. To address such problems, we can construct another simpler probability distribution q and perform a trick to the original expectation equation:

$$E[f(x)] = \int f(x) p(x) \, dx = \int f(x) \frac{p(x)}{q(x)} q(x) \, dx$$

Now we sample from the new distribution q and calculate the following:

$$\hat{E}[f(x)] = \frac{\sum_{i=1}^{n} f(x_i) \frac{p(x_i)}{q(x_)}}{n}$$

This is still an unbiased estimator of $f(x)$ under p. This time, each one of the samples we draw from q are weighted by a factor $\frac{p(x_i)}{q(x_i)}$, which is called the likelihood ratio, and the whole process is called importance sampling.

4. Random Fourier Features (RFF)

The theoretical foundation of using random fourier features to approximate kernels that are both positive definite and shift invariant is a direct result of Bochner's Theorem. If we do a little more transformation on equation (5), we will get:

$$\begin{aligned} k(x - y) &= \int_{\mathbb{R}^m} p(\omega) e^{-2\pi i \omega x} e^{2\pi i \omega y} \, \mathrm{d}\omega \\ &= \int_{\mathbb{R}^m} p(\omega) e^{-2\pi i \omega x} (e^{-2\pi i \omega y})^* \, \mathrm{d}\omega \\ &= \int_{\mathbb{R}^m} p(\omega) F_\omega(x) F_\omega(y)^* \, \mathrm{d}\omega \\ &= \mathbb{E}\left[F_\omega(x) F_\omega(y)^*\right] \end{aligned} \tag{6}$$

where $F_\omega(x) = e^{-2\pi i \omega x}$.

Equation (6) tells us that $F_\omega(x) F_\omega(y)^*$ is an unbiased estimation of the kernel function if ω is randomly sampled from the probability distribution $p(\omega)$.

(Rahimi & Recht, 2007) shows in their paper that equation (6) converges if the complex exponential term is replaced

with cosine functions and that a real-valued $F_\omega(x)$ satisfying the equation can be defined as:

$$F_\omega(x) = \sqrt{2}\cos(\omega^T x + b)$$

where $\omega \sim p(\omega)$ and $b \sim [0, 2\pi]$.

We are now ready to describe the full algorithm for generating the Random Fourier Features proposed in (Rahimi & Recht, 2007):

We are given a positive definite and shift invariant kernel function $k(x, y) = k(x-y)$ as input. Our goal is to obtain a random feature map $z(x) : \mathbb{R}^m \to \mathbb{R}^D$ so that $z(x)^T z(y) \approx k(x - y)$.

1. Compute the Fourier inverse transformation of $k(\delta)$ as $p(\omega)$
$$p(\omega) = \frac{1}{2\pi}\int_{\mathbb{R}^m} k(\delta)e^{-2\pi i\delta\omega}\,\mathrm{d}\delta$$
2. Draw D iid samples from $p(\omega)$ as $\omega_1, \omega_2, ..., \omega_D \in \mathbb{R}^m$ and D iid samples from $[0, 2\pi]$ as $b_1, b_2, ..., b_D \in \mathbb{R}$.
3. Compute $z(x)$ as
$$z(x) = \sqrt{\frac{2}{D}}(\cos(\omega_1^T x + b_1), ..., \cos(\omega_D^T x + b_D)) \tag{7}$$

The $z(x)$ vector obtained in step 3 is called the Random Fourier Feature of a data point x. The space and time complexity of this algorithm is $O(nD)$ and $O(nmD)$ as the major step is computing $z(x_i) \in \mathbb{R}^D, \forall 1 \le i \le n, i \in \mathbb{Z}^+$. I.e. the space and time complexity to compute RFF is linear respect to the size of train data, while the traditional kernel methods is quadratic w.r.t to n in time and cubic in space.

In the actual implementation of RFF to approximate Gaussian kernels, we don't ever need to do step 1 because the Fourier inverse transformation of a Gaussian kernel is also a Gaussian kernel. Thus we might substitute step 1 with randomly initializing a Gaussian kernel as $p(\omega)$.

5. Variants of Random Fourier Features

5.1. Overview

We have shown that the key step in random feature based algorithm lies in selecting the weights ω_i in equation (7) and α_i in equation (3). And variants of random feature based algorithms differ exactly in how they select these weights. (Liu et al., 2021a) As mentioned earlier, they can be separated into two categories, data independent methods and data dependent methods, depending on whether the training data is used to guide the selection of weights ω_i and α_i.

Data independent methods can be further split into acceleration focused method, eg. Fastfood (Le et al., 2014), and variance reduction focused method eg. Normalized RFF (NRFF) (Li, 2017). While data dependent methods can be further classified according to how weights are sampled or learned. The two main categories are leverage score sampling based methods, eg. Leverage Score-RFF (LS-RFF) (Li et al., 2021) and those that re-weight the random features, eg. Weighted Random Kitchen Sinks (RKS) (Rahimi & Recht, 2009).

In the next few sections, we explain these four most representative algorithms and compare them with the RFF method.

5.2. Data Independent Methods

5.2.1. FASTFOOD

As mentioned earlier, in (Rahimi & Recht, 2007), when approximating a Gaussian RBF kernel, the weight samples $\omega_i \in \mathbb{R}^m$ are drawn from a Gaussian/Normal distribution. If we slightly modify the last line of the vanilla RFF algorithm (7) by stacking the ω_i's into a matrix, we get:

$$\begin{aligned} z(x) &= \sqrt{\frac{2}{D}}[\cos(\omega_1 x + b), \ldots, \cos(\omega_D x + b)] \\ &= \sqrt{\frac{2}{D}}\cos(Vx + b) \end{aligned}$$

where $V = [\omega_1, \ldots, \omega_D]^T$. In practice, the matrix V is usually a dense Gaussian matrix, thus the multiplication step Vx takes $O(mD)$ time. A reminder that D is the number of random features.

It is then noticed that the workings of this dense Gaussian operator could be simulated with Hardamard matrices and a diagonal Gaussian operator (Le et al., 2014). Le et al. proposed the following decomposition of the dense Gaussian matrix V:

$$V = \frac{1}{\sigma\sqrt{D}}SHG\Pi HB$$

In this equation, $\Pi \in \{0, 1\}^{D\times D}$ is a random permutation matrix, S and B are diagonal matrices. More specifically, S is a random scaling matrix and B has random entries in $\{-1, 1\}$ on its main diagonal. G is the diagonal Gaussian operator, with its main diagonal entries randomly drawn from a Gaussian distribution.

The key for this method is the matrix H, the Walsh-Hardamard matrix. It is recursively defined, with the base case: $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, and the recursive case: $H_{2k} = \begin{bmatrix} H_k & H_k \\ H_k & -H_k \end{bmatrix}$. It can be defined this way because all

Hardamard matrices are square and have dimensions 2^k for some $k \in \mathbb{N}$.

With the the Walsh-Hardamard matrix H, the Fast Hardamard Transform, a well-known variant of Fast Fourier Transform, could be utilized to expedite the calculation. With the Fast Hardamard Transform, we can calculate Hx in $O(m \log D)$ time, instead of $O(mD)$, as in the normal matrix-vector multiplication. The multiplications between all other matrices and the vector x can all be done in linear time. Hence, this method brings the computational cost of (7) from $O(mD)$ down to $O(m \log D)$ and the overall time complexity of the algorithm from $O(nmD)$ to $O(nm \log D)$.

Additionally, the storage cost could be brought down to $O(m)$ as well. The Walsh-Hardmard matrix requires no storage at all, since its workings are within the Fast Hardmard Transform. All other diagonal matrices and the permutation matrix requires $O(m)$ storage. Hence, the Fastfood method also enjoys storage benefit as compared to the RFF method.

5.2.2. Normalized Random Fourier Feature (NRFF)

The ultimate goal of Random Fourier Features is to produce random feature vectors $z(x)$, $z(y)$ such that

$$z(x)z(y) \approx k(x, y)$$

Since the features $z(x)$ are constructed based on randomly sampled weights, the accuracy of the approximation might fluctuate heavily. A group of researchers (Li, 2017) noticed the problem and proposed a technique to reduce the variance of the approximation. In the field of statistics, a procedure called normalized linear kernel or correlation is often used to reduce variance. It is calculated as below:

$$p(x, x') = \frac{\sum_{i=1}^{D} x_i x_i'}{\sqrt{\sum_{i=1}^{D} x_i^2} \sqrt{\sum_{j=1}^{D} {x_j'}^2}}$$

Now, if this is applied to any two random feature vectors instead of the inner product to estimate the kernel function, we will have:

$$z(x)z(y) \approx k(x, y) \rightarrow p(z(x), z(y)) \approx k(x, y) \tag{8}$$

The authors claimed that this additional step (8) would lead to great reduction in variance for the prediction results. This means the results of the Random Fourier Features would be more stable, and the authors gave their method the name: Normalized Random Fourier Features, due to the use of the correlation procedure.

The authors of NRFF did not compare the time and space complexity of this method to vanilla RFF. However, since it adds an additional step to the estimation of kernel function, it has no hope in reducing the time complexity. However, as the L-2 norm of each data point x_i can be pre-computed and stored, the additional step (8) only takes an extra $O(1)$ to complete. It is thus able to bring the benefit of low variance without affecting running time too much. Storage-wise, the L-2 norm could be stored alongside each random feature vectors after they are generated, hence NRFF incurs a small storage increase. Overall, it is worthy to preform such an additional procedure.

5.3. Data Dependent Methods

5.3.1. Leverage Score Random Fourier Features (LS-RFF)

Ridge leverage score (Alaoui & Mahoney, 2015) is essentially extending the idea of leverage score to kernel ridge regression. Over the years, researchers came back and forth about this topic, and one function people widely agreed on is (Avron et al., 2017):

$$l_\lambda(\omega) = p(\omega) z_\omega(x)^T (K + n\lambda I) z_\omega(x) \tag{9}$$

Here, λ is a regularization parameter, ω's are the random features being sampled, l is the leverage score of ω. $z_\omega(x)$ is the Random Fourier Feature generated, and $K \in \mathbb{R}^{n \times n}$ is the kernel matrix defined as:

$$K_{i,j} = k(x_i, x_j)$$

Now, if we integrate the leverage score over the entire space of ω, we will get:

$$\int_{\mathbb{R}^m} l_\lambda(\omega)\, d\omega = d_K^\lambda$$

where d_K^λ is the number of effective degrees of freedom. If we then divide the leverage score of ω, l_ω, by d_K^λ, we effectively have a probability distribution:

$$q^*(\omega) = \frac{l_\lambda(\omega)}{d_K^\lambda}$$

which is called the empirical ridge leverage score distribution (Li et al., 2021).

In section 3.2 we introduced the idea of importance sampling. Instead of sampling directly from the initial distribution $p(\omega)$ as in the vanilla RFF procedure, we can sample ω_i's from $q^*(\omega)$ as defined above. Then weight each sample by the likelihood ratio, $\frac{p(\omega)}{q^*(\omega)}$. In this way, less samples are be needed. In fact, the number of samples needed could be reduced from $\Omega(\sqrt{m})$, as in original RFF procedure, to $\Omega(1)$ in the leverage-score based importance sampling procedure, which we call LS-RFF, if the spectrum is of finite rank together with certain other conditions (Li et al., 2021).

Now, with fewer samples, it is expected that the time and storage complexity would decrease. However, this performance improvement relies on the computation of $q^*(w)$, essentially $l_\lambda(\omega)$ in (9), which is very costly. Thus the researches that proposed LS-RFF also proposed a fast approximation algorithm of sampling from $q^*(\omega)$, which we describe below (Li et al., 2021):

We are given empirical data points $(x_1, y_1), ..., (x_m, y_m)$, the kernel function to approximate $k(x, y)$, and a regularizer parameter λ. We want to output $k (k < D)$ new random features sampled from $q^*(\omega)$:

1. Sample $\{\omega_1, \ldots, \omega_D\}$ from $p(\omega)$.
2. Create feature matrix $Z_D \in \mathbb{R}^{m \times D}$ such that its i^{th} row is $[z_{\omega_1}(x_i), \ldots, z_{\omega_D}(x_i)]^T$.
3. Associate each ω_i with a real number p_i such that p_i is equal to the i^{th} diagonal element of the following matrix:
$$Z_D^T Z_D ((\frac{1}{D}) Z_D^T Z_D + m\lambda I)^{-1}$$
4. $K \leftarrow \{(\omega_i, \frac{p_i}{k}) : k = \sum_{i=1}^m p_i\}_{i=1}^D$
5. Finally, sample k features from K using the multinomial distribution given by the vector $(\frac{p_1}{k}, \ldots, \frac{p_D}{k})$.

This requires a time complexity of $O(mD^2 + D^3)$ for each data point and $O(nmD^2 + nD^3)$ for the entire dataset. Clearly, LS-RFF is slower than RFF. But since $D \ll m$, the time complexity increase isn't too much. In addition, LS-RFF algorithm has been proven to have much better empirical prediction accuracy by (Li et al., 2021). Thus, LS-RFF stands as a very strong variant of RFF, even though one might want to consider the trade-off between speed and accuracy before deciding if to use LS-RFF.

5.3.2. Weighted Random Kitchen Sinks (RKS)

Following from RFF that was published in 2008, Rahimi and Recht generalized their ideas of using randomness in machine learning. In traditional kernel method, we choose a kernel function $k(x, y)$ and construct the objective function $f(x)$ by learning α_i in equation (3). And random features facilitates the evaluation of the kernel function k. However, why do we have to use a precise kernel to project the training data at all? Why can't we randomize the feature functions and just learn the weights α? That is exactly the essence of the weighted sums of random kitchen sink method proposed in (Rahimi & Recht, 2009).

The algorithm is very simple. It takes as input a set of n points $x_1, x_2, ..., x_n \in \mathbb{R}^m$ and their labels $y_1, y_2, ..., y_n$, some arbitrary non-linear function $\phi(x, w)$, a scalar C, and some $k \in \mathbb{Z}^+$:

1. Randomly initialize k weight vector w where each w_i is sampled iid from some pre-specified distribution $p(\omega)$.
2. Compute the feature vectors of every point x_i
$$z(x_i) = (\phi(x_i, w_1), \phi(x_i, w_2), ..., \phi(x_i, w_k))$$
3. With ω fixed, solve the following minimization problem w.r.t. α
$$\min_{\alpha \in \mathbb{R}^k} \frac{1}{n} \sum_{i=1}^n l(\alpha^T z(x_i), y_i) \tag{10}$$
such that $|\alpha_i| \leq Cp(\omega_i)$. l is the loss function and (Rahimi & Recht, 2009) uses the quadratic loss.
4. Output the learned prediction function
$$f(x) = \sum_{i=1}^k \alpha_i \phi(x, \omega_i)$$

This method is linear in storage cost w.r.t to n, similar to RFF and other variants. However, without using a kernel function, nor to say approximating it, this method is much faster than RFF. The authors also compared this method to Adaboost, another population learning method that uses weighted sums of weak classifier. Due to the difficulty of comparing the two's time complexity theoretically, their empirical performances were discussed. RKS and Adaboost were used to learn 3 different datasets (size 3200×123, 5000000×127, 200000×223, in the form: the number of samples $\times$ the dimension size). In all three datasets, RKS is between one and three orders of magnitude faster than Adaboost, at a similar error rate.

Even though this method is much less guided than the traditional kernel method and RFF, its accuracy is still comparable with them. The constraint on α in (10) guarantees that a feature function $\phi(x_i, \omega_i)$ will not get a high weight (α_i) if it is (represented by ω_i) unlikely to be sampled from $p(\omega)$. And it has been proven mathematically in (Rahimi & Recht, 2009) that the test error of RKS is linear w.r.t. $\frac{1}{\sqrt{m}} + \frac{1}{\sqrt{k}}$.

An interesting observation is that this method is essentially a randomly initialized neural network, except that it has an extra constraint of the weights α, which gives it a nice theoretical error bound.

6. Discussion and Conclusion

Ever since the vanilla RFF method was published, numerous researchers were inspired and attempted to improve it by modifying the weight sampling scheme, compressing the weight matrix etc. As the training dataset for machine learning today is getting ever larger, approximation is a great way

to get reasonable results in reasonable time. The competition for the best method in the field is multi-dimensional, with criteria ranging over speed, storage, learning rate, variance, prediction accuracy and so on. It is impossible to say which one is the best, as different methods are better at different places. What we discussed above showed exactly this: Fastfood and RKS improve on speed and storage, NRFF on variance, LS-RFF on prediction accuracy. With its strong theoretical support in Mathematics and Statistics, the random features method for kernel approximation is an interesting digression from today's popular machine learning algorithms. The popular algorithms like convolutional neural networks and deep learning models feature a massive amount of parameters, but suffer badly from criticisms against their explanability. Not every research laboratory or company have comparable computing resources as technology giants like Google, hence lightweight models like those introduced in this paper can get on the table. Research in random features to approximate kernel functions has also sparked new discussions outside the field, one notable example being the Lottery Ticket Hypothesis which also achieved great success. In the future, it can be speculated that more methods would be developed in this field, and for the users, the most important thing would be to choose the one that is most suited for their task.

References

Achlioptas, D., McSherry, F., and Schölkopf, B. Sampling techniques for kernel methods. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pp. 335–342, Cambridge, MA, USA, 2001. MIT Press.

Alaoui, A. E. and Mahoney, M. W. Fast randomized kernel ridge regression with statistical guarantees. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 775–783, Cambridge, MA, USA, 2015. MIT Press.

Avron, H., Kapralov, M., Musco, C., Musco, C., Velingker, A., and Zandieh, A. Random fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 253–262. JMLR.org, 2017.

Drineas, P. and Mahoney, M. W. On the nyström method for approximating a gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6:2153–2175, dec 2005. ISSN 1532-4435.

Hofmann, T., Schölkopf, B., and Smola, A. J. Kernel methods in machine learning. *The Annals of Statistics*, 36(3): 1171 – 1220, 2008. doi: 10.1214/009053607000000677.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Le, Q. V., Sarlos, T., and Smola, A. J. Fastfood: Approximate kernel expansions in loglinear time, 2014.

Li, P. Linearized gmm kernels and normalized random fourier features, 2017.

Li, Z., Ton, J.-F., Oglic, D., and Sejdinovic, D. Towards a unified analysis of random fourier features, 2021.

Liu, F., Huang, X., Chen, Y., and Suykens, J. A. K. Random features for kernel approximation: A survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021a. doi: 10.1109/TPAMI.2021.3097011.

Liu, F., Huang, X., Chen, Y., and Suykens, J. A. K. Random features for kernel approximation: A survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021b. doi: 10.1109/TPAMI.2021.3097011.

Platt, J. Using analytic qp and sparseness to speed training of support vector machines. *Advances in Neural Information Processing Systems*, 11, 02 1999.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pp. 1177–1184, Red Hook, NY, USA, 2007. Curran Associates Inc. ISBN 9781605603520.

Rahimi, A. and Recht, B. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L. (eds.), *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2009.

Empirical Review of GAN training

Bhargav Ghanekar Camille Little Lucy Liu

Abstract

Recent work in games in deep learning has lead to a popular adversarial framework called generative adversarial networks (GANs). The GAN framework tries to solve a zero-sum, two-player game in an attempt to find a local Nash equilibrium. Compared to a single-objective optimization problem, the two-player game introduces more complexity and challenges to the training process. In this work, we investigate the challenges in successfully training GANs and explore existing solutions to address the challenges. We describe the algorithms of the proposed approaches, discuss why they should address some of the difficulties in GAN training, and empirically study and compare the success of the algorithms with respect to training GANs using the MNIST dataset.

1. Introduction

Generative adversarial networks (GANs) have recently become an extremely hot topic in deep learning research. Their ability to generate new data based on training data proves them useful in several areas, including computer vision, natural language processing, and medicine (Wu et al., 2017). GANs, proposed in 2014 by (Goodfellow et al., 2014), consist of two neural networks: a generator and a discriminator. The generator attempts to capture the distribution of the true data samples and generate new data samples. The discriminator, on the other hand, attempts to distinguish between the true data samples and the generated data samples. The optimization of GANs is a mini-max optimization problem where the goal is to reach the Nash equilibrium (Ratliff et al., 2013). This occurs when a saddle point is at a maximum with respect to the discriminator and a minimum with respect to the generator. GANs are traditionally trained using stochastic gradient descent with backpropagation (Goodfellow et al., 2014).

The growing popularity of GANs has lead to the exposure of common issues in GAN training. The mini-max optimization problem often makes GAN training very fickle and unstable. One issue that arises is mode collapse. This occurs when the generator network learns to generate samples from only a few modes of the data distribution, but misses others present in the true data (Srivastava et al., 2017). Vanishing gradients are another common reason why GAN training fails. This occurs when the generator updates become vanishingly small, making it impossible for the GAN to converge to a satisfying solution (Li et al., 2018). Additionally, it has been shown that zero-sum games often lead to limit oscillatory behavior, rather than reaching equilibrium (Mertikopoulos et al., 2017).

Several methods have been proposed to address the common challenges in GAN training. In this work, we aim to empirically study these methods to understand which methods work in practice in successfully addressing the instability of training GANs. Our contributions are as follows: We first formalize the definition of GANs and their traditional training methods. We then discuss three methods that alter the GAN optimization formulation in an attempt to improve instability during GAN training. Next, we empirically evaluate and compare each method on the MNIST dataset. Lastly, we summarize our findings and conclusions.

2. GAN Formulation

In 2014, (Goodfellow et al., 2014) proposed a framework for estimating generative models via an adversarial process. To achieve this, they simultaneously train two models: a generator and a discriminator. The generative model G captures the data distribution and the discriminative model D estimates the probability that the sample came from the true data or from the generator. The goal of the generator is to maximize the discriminator's error. The GAN framework is a zero-sum, two-player game. It's easiest when G and D are defined as multi-layer perceptions so the system can be trained with back-propagation. The GAN formulation is formalized as follows

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\ P_{data}(\mathbf{x})}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\ P_{\mathbf{z}}(\mathbf{z})}[log(1 - D(G(z)))],$$

where $P_{\mathbf{z}}(\mathbf{z})$ represents the prior on input noise variables and $G(\mathbf{z}; \theta_g)$ represents a mapping to data space. G is a multi-layer perceptron with parameters θ_g and $D(\mathbf{x}, \theta_d)$ is a multi-layer perception with parameters θ_d that outputs a single scalar. The traditional GAN framework is typically solved via stochastic gradient descent with backpropogation. The GAN training methodology is formalized in the

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 1. GAN training algorithm from (Goodfellow et al., 2014)

Algorithm 1.

2.1. Wasserstein GANs

First introduced in 2017 by (Arjovsky et al., 2017), Wasserstein GANs (WGANs) were proposed to improve stability in GAN training and deal with issues such as mode collapse. The WGAN formulation modifies the discriminator's loss function. Specifically, instead of treating the discriminator as a classifier, it tries to approximate the earth-mover metric between the true distribution and the distribution of the generator. As a result, the discriminator function for WGANs can be summarized as follows:

$$V(D, G) = \nabla_{\theta_d} \frac{1}{m} \sum_{1=1}^{m} [D(\mathbf{x^{(i)}}) + D(G(\mathbf{z}^{(i)}))].$$

2.2. Difficulties in training GANs

Training GANs properly is notoriously difficult in practice. In general, all the gradient descent-based techniques are based on the objective of minimizing a cost function. Thus, to simply use them directly for a mini-max GAN objective to find a Nash equilibrium point is bound to not work properly. This is due to a variety of reasons -

- **Vanishing gradients**: In gradient-based learning methods, neural network (NN) weight updates are computed through backpropagation. Thus in NNs with many hidden layers, the gradients get "vanishingly" small for the initial layers, which prevents the weight from changing after update.

- **Mode Collapse**: This occurs when many points/noise instances in the latent space map to the exact same output, the generator does generate realistic images, but only of one kind, and lacking diversity.

- **Failure to Converge**: Converging can be challenging as the generator and the discriminator compete against one another during training. The vector field associated with the update step could be non-conservative, leading to cyclic/rotational behaviors.

3. Training with Optimism

Training GANs with optimism was first introduced in 2017 by (Daskalakis et al., 2017). The goal of training with optimism is to address the limit cycling behavior present in GAN training. They propose training WGANs via a variant called Optimistic Mirror Decent (OMD). They do this because it has been shown that using OMD leads to faster convergence rates than gradient descent in convex-concave zero-sum games (Rakhlin & Sridharan, 2013). They employ this method by applying a small change in the update step within the general mirror decent framework. To begin, they first consider the loss function in the WGAN framework

$$L(\theta, \varphi) = \mathbb{E}_{x\ Q}[D_\varphi(x)] - \mathbb{E}_{z\ F}[D_\varphi(G_\theta(z))],$$

where the the generator G controls θ and the discriminator D controls φ. Additionally, Q represents the true data distribution. When training standard WGAN, the typical approach to solve the game is to run gradient descent for each player

$$\varphi_{t+1} = \varphi_t + \eta \cdot \nabla_{\varphi,t}$$
$$\theta_{t+1} = \theta_t + \eta \cdot \nabla_{\theta,t}$$

where $\nabla_{\varphi,t} = \nabla_\varphi L(\theta_t, \varphi_t)$ and $\nabla_{\theta,t} = \nabla_\theta L(\theta_t, \varphi_t)$. The Optimistic Mirror Decent algorithm proposed by (Rakhlin & Sridharan, 2013) uses the previous iteration's gradient as a predictor for the next iteration's gradient. In 2015, it was shown by (Syrgkanis et al., 2015) that this leads to faster convergence of each individual player in normal form games. To apply this to the WGAN formulation, a small tweak is made to the update rules for φ and θ:

$$\varphi_{t+1} = \varphi_t + 2\eta \cdot \nabla_{\varphi,t} - \eta \cdot \nabla_{\varphi,t-1}$$
$$\theta_{t+1} = \theta_t + 2\eta \cdot \nabla_{\theta,t} - \eta \cdot \nabla_{\theta,t-1}$$

The predictor of the next iteration's gradient is either by the previous gradient or an average of a window of previous gradients. OMD exhibits a last iteratation rate of $O(\frac{1}{T})$, where T is the number of iterations. This method presents a very small adaptation of normal gradient descent within the WGAN framework that largely improves convergence rates and addresses the limit oscillatory behavior often found in GANs.

Because of the success of the Adam algorithm on training images WGANs, (Daskalakis et al., 2017) propose an optimistic version of Adam. We examine how well this method works in practice on the MNIST dataset in the empirical studies section.

Algorithm 1 *Optimistic ADAM*, proposed algorithm for training WGANs on images.

Parameters: stepsize η, exponential decay rates for moment estimates $\beta_1, \beta_2 \in [0, 1)$, stochastic loss as a function of weights $\ell_t(\theta)$, initial parameters θ_0
for each iteration $t \in \{1, \ldots, T\}$ **do**
  Compute stochastic gradient: $\nabla_{\theta,t} = \nabla_\theta \ell_t(\theta)$
  Update biased estimate of first moment: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \cdot \nabla_{\theta,t}$
  Update biased estimate of second moment: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) \cdot \nabla^2_{\theta,t}$
  Compute bias corrected first moment: $\hat{m}_t = m_t / (1 - \beta_1^t)$
  Compute bias corrected second moment: $\hat{v}_t = v_t / (1 - \beta_2^t)$
  Perform *optimistic gradient step*: $\theta_t = \theta_{t-1} - 2\eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \eta \frac{\hat{m}_{t-1}}{\sqrt{\hat{v}_{t-1} + \epsilon}}$
Return θ_T

Figure 2. Optimistic Adam algorithm from (Goodfellow et al., 2014)

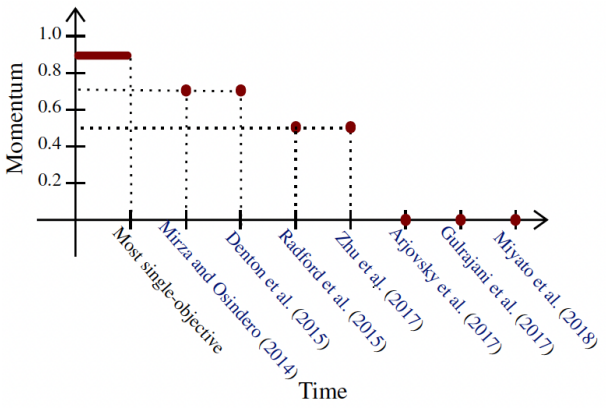


Figure 3. Decreasing trend in the value of momentum used in GAN literature. Use the figure from (Gidel et al., 2019)

4. Training with Negative Momentum

The paper by (Gidel et al., 2019) explored the effects of negative momentum in Min-Max zero-sum Game Formulation (i.e. GANs), inspired by a decreasing trend of momentum values over time for GAN training, as shown in Figure 3.

We can interpret GAN training as a differentiable two-player game – the discriminator D_φ with its loss function L_D and the generator G_θ with L_G. Both of them want to minimize their own loss function, so the GAN objective can be written as

$$\theta^* \in \arg\min_{\theta \in \Theta} \; L_G(\theta\,, \varphi^*)$$
$$\varphi^* \in \arg\min_{\varphi \in \Phi} \; L_D(\theta^*\,, \varphi)$$

From a game theory point of view, GAN training aims to find a local Nash Equilibrium (φ^*, θ^*), where neither discriminator nor generator can improve their loss by training more epochs. The dynamics of gradient-based methods near a Nash Equilibrium can be analyzed using the *gradient vector field*:

$$v(\varphi, \theta) = [\nabla_\varphi L_D(\varphi, \theta) \;\; \nabla_\theta L_G(\varphi, \theta)]^T$$

Then for the gradient update, it is defined as

$$F_\eta(\varphi, \theta) = [\varphi \;\; \theta]^T - \eta \; v(\varphi, \theta) \tag{1}$$

where $(\varphi, \theta) \in \mathbb{R}^m$, η is the step-size. For simplicity, we denote $w := (\varphi, \theta) \in \mathbb{R}^m$. Then for $t \in \mathbb{N}$, $w_t = (\varphi_t, \theta_t)$ is the t^{th} point from the gradient update, such that $w_t = \underbrace{F_\eta \circ \cdots \circ F_\eta(w_0)}_{t} = F_\eta^{(t)}(w_0)$. If the gradient method converges and $w^* = (\varphi^*, \theta^*)$ is a fixed-point of F_η such that $\nabla v(w^*)$ is positive-definite, then w^* is a local Nash Equilibrium and a stationary point of the gradient dynamics (i.e. $v(w^*) = 0$).

From Equation 1, we can get

$$\nabla F_\eta(w^*) = I_m - \eta \nabla v(w^*)$$
$$Sp(\nabla F_\eta(w^*)) = \{1 - \eta\lambda | \lambda \in Sp(\nabla v(w^*))\}$$

If the eigenvalues of $\nabla v(w^*)$ all have positive real parts and η is small enough, the eigenvalues of $\nabla F_\eta(w^*)$ are inside a convergence circle of radius $\rho_{max} < 1$, as shown in Figure 4. Hence, Prop. 4.4.1 in (Bertsekas, 1997) guarantees the existence of an optimal η^* which yields a non-trivial convergence rate $\rho_{max} < 1$. The spectral radium $\rho_{max}(\eta)$ of $\nabla F_\eta(w^*)$ is the solution of a convex quadratic problem and satisfies

$$\max_{1 \le k \le m} \sin(\psi_k)^2 \le \rho_{max}(\eta^*)^2 \le 1 - \Re(\frac{1}{\lambda_1})\delta \tag{2}$$
$$with \;\; \delta := \min_{1 \le k \le m} |\lambda_k|^2 (2\Re(\frac{1}{\lambda_k}) - \Re(\frac{1}{\lambda_1})) \tag{3}$$
$$and \;\; \Re(\frac{1}{\lambda_1}) \le \eta^* \le 2\Re(\frac{1}{\lambda_1}) \tag{4}$$

where $(\lambda_k = r_k e^{i\psi_k})_{1 \le k \le m} = Sp(\nabla v(w^*))$ are sorted such that $0 < \Re(\frac{1}{\lambda_1}) \le \cdots \le \Re(\frac{1}{\lambda_m})$. ($\Re$ means the real part). Hence we have $\delta > 0$ since $\Re(\frac{1}{\lambda_k}) \ge \Re(\frac{1}{\lambda_1})$ for $\forall\, 1 \le k \le m$.

From Figure 4, we know that η^* exists when ∇v is positive-definite because of one or more limiting eigenvalues. Then from Equation 2, if the Jacobian of v has a large imaginary value $r_j e^{\psi_j}$, then $\sin(\psi_j)$ becomes close to 1, which means the convergence rate of the gradient may be arbitrarily close to 1. (Zhang & Mitliagkas, 2017) analyzes the performance of momentum method for quadratic functions, and claims that adding momentum that makes the Jacobian eigenvalues turn from positive reals into complex conjugate pairs achieve the best model condition.

From above, we know eigenvalues with large imaginary parts only works with small step-size, thus leading to slow convergence rates, so the authors consider adding a negative momentum value. Intuitively, negative momentum

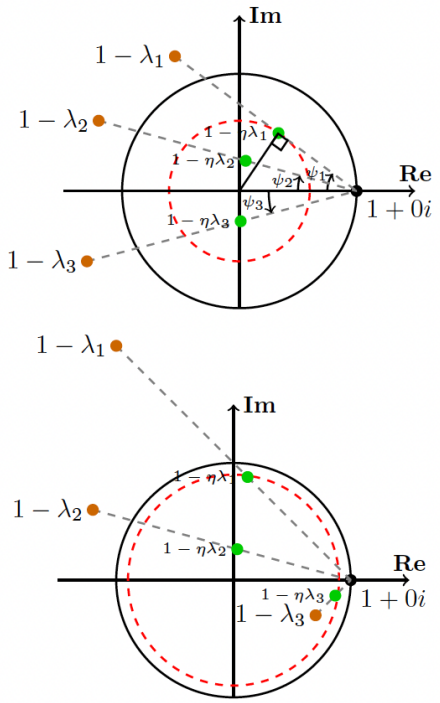


Figure 4. An illustration of Eigenvalues λ_i of the Jacobian $\nabla v(w^*)$ and their trajectories $1-\eta\lambda_i$ for growing η. The unit circle is in **black**, the red circle has radius of the largest eigenvalue μ_{max}. Smaller red circles mean better convergence rates. **Top:** The red circle is limited by the tangent trajectory line $1-\eta\lambda_1$. It means the best convergence rate is limited only by the eigenvalue which will pass furthest from the origin as η grows.(i.e., $\lambda_i = \arg\min \Re(\frac{1}{\lambda_i})$ **Bottom:** The red circle is cut (not tangent) by the trajectories at points $1-\eta\lambda_1$ and $1-\eta\lambda_3$. The η is optimal because any increase will push λ_1 out of the red circle, while any decrease will retract λ_3 out of the red circle. Use the figure from (Gidel et al., 2019)

can be seen as friction that can damp oscillations. Following (Zhang & Mitliagkas, 2017), let $(w_t, w_{t-1}) = (\varphi_t, \theta_t, \varphi_{t-1}, \theta_{t-1}) \in \mathbb{R}^{2m}$, then Equation 1 can be updated to

$$F_{\eta,\beta}(w_t, w_{t-1}) = (w_{t+1}, w_t)$$
$$where\ w_{t+1} := w_t - \eta v(w_t) + \beta(w_t - w_{t-1})$$

in which, $\beta \in \mathbb{R}$ is the momentum parameter. When $\beta = 0$, we obtain the original gradient method.

Then we can derive the eigenvalues of the Jacobian of $F_{\eta,\beta}(w^*)$:

$$\mu_{\pm}(\beta, \eta, \lambda) = (1 - \eta\lambda + \beta)\frac{1 \pm \Delta^{\frac{1}{2}}}{2}$$

where $\Delta := 1 - \frac{4\beta}{(1-\eta\lambda+\beta)^2}$, $\lambda \in Sp(\nabla v(w^*))$, and $\Delta^{\frac{1}{2}}$ is the complex square root of Δ with positive real part (if Δ has negative real part, $\Delta^{\frac{1}{2}} = i\sqrt{-\Delta}$).

Then we apply Taylor Expansion and get an approximation:

$$\mu_+(\beta, \eta, \lambda) = 1 - \eta\lambda - \beta\frac{\eta\lambda}{1-\eta\lambda} + O(\beta^2)$$
$$\mu_-(\beta, \eta, \lambda) = \frac{\beta}{1-\eta\lambda} + O(\beta^2)$$

In this case, when β is small, Δ gets close to 1. As a result, μ_+ is close to $1-\eta\lambda$, the original eigenvalue for gradient method, and μ_- is close to 0. Then the authors formalize an intuition that in some situations, if $\beta < 0$ is suited well, it can pushing the eigenvalues of Jacobian towards the origin, thus improving the convergence rate to a local stationary point.

In order to prove this intuition, the paper explores the effect of β on the eigenvalues of $F_{\eta,\beta}(w^*)$ with relatively large magnitude. Then they define the eigenvalues' magnitude as $\rho_{\lambda,\eta}(\beta)$ for optimization:

$$\rho_{\lambda,\eta}(\beta) := \max\{|\mu_+(\beta, \eta, \lambda)|^2, |\mu_-(\beta, \eta, \lambda)|^2\} \quad (5)$$

For $\forall\lambda \in Sp(\nabla v(w^*))$ such that $\Re(\lambda) > 0$, we can get

$$\rho'_{\lambda,\eta}(0) > 0 \Leftrightarrow \eta \in I(\lambda) := \left(\frac{|\lambda| - |\Im(\lambda)|}{|\lambda|\Re(\lambda)}, \frac{|\lambda| + |\Im(\lambda)|}{|\lambda|\Re(\lambda)}\right)$$

where $\Re$ and $\Im$ means the real and imaginary parts respectively. Particularly, $\rho'_{\lambda,\eta}(0) = 2\Re(\lambda)\Re(\frac{1}{\lambda}) > 0$.

This theorem shows that a properly adjusted small β decreases $\rho_{\lambda,\eta}$ which corresponds to faster convergence. In this case, a small negative momentum can improve the magnitude eigenvalues when it satisfies one of the two conditions:

1. When there is only one limiting eigenvalue λ_1, the optimal step-size $\eta^* = \Re(\frac{1}{\lambda_1}) \in I(\lambda_1)$.

2. When there are several limiting eigenvalues $\lambda_1, \cdots, \lambda_k$ and their intersection $\lambda_1 \cap \cdots \cap \lambda_k$ is not empty. If the absolute value of the argument of $\lambda_1 > \frac{\pi}{4}$ then by Equation 4, the optimal step-size $\eta^* \in I(\lambda_1)$

The above theorem only provides a local result of β, so it could not guarantees on large negative values of momentum. They, nevertheless, claimed that by numerically optimizing Equation 5 with respect to β and η, the optimal momentum is negative for any non-imaginary fixed eigenvalue λ, and the corresponding optimal $\eta^* > \widehat{\eta}(\lambda)$.

In Empirical Study Section, we analyzed the performance of negative momentum for GAN training with real dataset.

5. Consensus Optimization

Training GANs can be thought of as a two-player zero-sum game, where the generator and discriminator are the *two players* with utility functions $f(\theta, \varphi)$, $g(\theta, \varphi)$ respectively. One player (generator) controls the parameters θ (corresponding to the weights of the generator network), while the other (discriminator) controls the parameters φ (corresponding to the weights of the discriminator network). While training GANs, the objective is to achieve Nash equilibrium, i.e. to find weights $(\bar{\theta}, \bar{\varphi})$ such that

$$\bar{\theta} = \underset{\theta}{\operatorname{argmax}}\ f(\theta, \bar{\varphi})$$

$$\bar{\varphi} = \underset{\varphi}{\operatorname{argmax}}\ g(\bar{\theta}, \varphi)$$

One way to perform the above optimizations is to perform simultaneous gradient ascent on both parameters θ, φ i.e.

$$x_{t+1} = x_t + hv(x_t) \tag{6}$$

where $x_t = \begin{bmatrix} \theta_t \\ \varphi_t \end{bmatrix}$ and $v(x)$ corresponds to the gradients i.e.

$$v(x) = \begin{bmatrix} \nabla_\theta f(\theta, \varphi) \\ \nabla_\varphi g(\theta, \varphi) \end{bmatrix} \tag{7}$$

In a standard gradient ascent setting, such a vector field $v(x)$ is conservative, i.e. it is simply a gradient of some scalar function. However, in the case of simultaneous gradient ascent, that is not the case, and thus the vector field $v(x)$ could be non-conservative. If a vector field is non-conservative, updating the parameters $x = \begin{bmatrix} \theta \\ \varphi \end{bmatrix}$ as per Equation 6 might not work - the updates may diverge or just simply go about cyclically around a stationary point of the vector field.

In the work by (Mescheder et al., 2017), the authors explain the same in a theoretical sense. Let $\bar{x}$ be a point of local Nash equilibrium. They show that local convergence of the iterates expressed in Equation 6 for simultaneous gradient ascent is guaranteed provided that the Jacobian $v'(\bar{x})$ has eigenvalues with negative real parts, and furthermore, show that the step size h has to be smaller than

$$h < \frac{1}{\Re(\lambda)} \frac{2}{1 + (\frac{\Im(\lambda)}{\Re(\lambda)})^2}$$

where λ is the largest eigenvalue of the Jacobian $v'(\bar{x})$. This bound highlights the issue with simultaneous gradient ascent – for a Jacobian with a very small eigenvalue real part, and/or when the ratio of the imaginary to the real part of eigenvalues is very high – the step size becomes vanishingly small and leads to very slow convergence.

The way (Mescheder et al., 2017) solved this problem was by adding a regularizer term to the vector field $v(x)$. The quantity $L := \frac{1}{2}\|v(x)\|_2^2$ corresponds the norm of the gradients of f and g. At local Nash equilibrium points, $v(x) = 0$, implying $L = 0$ as well. Thus minimizing L is useful while finding local Nash equilibrium points. However, simply minimizing L is not correct - since it does not distinguish between unstable points, maxima, minima, saddle points. (Mescheder et al., 2017) shows that adding ∇L to the vector field $v(x)$ is useful, and leads to regularization of the vector field $v(x)$, making the problem less ill-posed. The consensus optimization update step, thus, as outlined in (Mescheder et al., 2017) is as follows:

$$x_{t+1} = x_t + hw(x) \tag{8}$$

$$\text{where} \qquad w(x) = v(x) - \gamma \nabla L \tag{9}$$

This amounts to the following procedure for consensus optimization:

Algorithm 1 Consensus Optimization

while not converged **do**
  $v_\theta \leftarrow \nabla_\theta(f(\theta, \varphi) - \gamma L(\theta, \varphi))$
  $v_\varphi \leftarrow \nabla_\varphi(g(\theta, \varphi) - \gamma L(\theta, \varphi))$
  $\theta \leftarrow \theta + hv_\theta$
  $\varphi \leftarrow \varphi + hv_\varphi$
end while

(Mescheder et al., 2017) show that the ratio $\Im(\lambda)/\Re(\lambda)$ can be controlled/reduced by increasing the value of the hyperparameter γ, thus making the problem better posed by allowing larger values of the step size h and achieving faster convergence.

We analysed how the consensus optimization procedure performs in practice, the details for the same have been outlined in Section 6.

6. Empirical Studies

We perform empirical studies of GAN training with optimization methods mentioned in the previous sections on MNIST dataset (Deng, 2012), which contains 60,000 labeled images of digits.

6.1. Experiments

6.1.1. Training GANs

We first train Vanilla GAN, Deep Convolutional GAN (DCGAN) (Radford et al., 2015) and WGAN as the baseline models for comparisons with our explored optimization methods. For Vanilla GAN and DCGAN, we use learning rate of 0.0002, while for WGAN, the learning rate is 0.00005. Given the success of Adam Optimizer (Kingma & Ba, 2014) in neural networks, we use it with parameter $\beta_1 = 0.5$ and $\beta_2 = 0.999$. In Figure 5, we can see the plot of loss function and generated digits after 200 epochs.

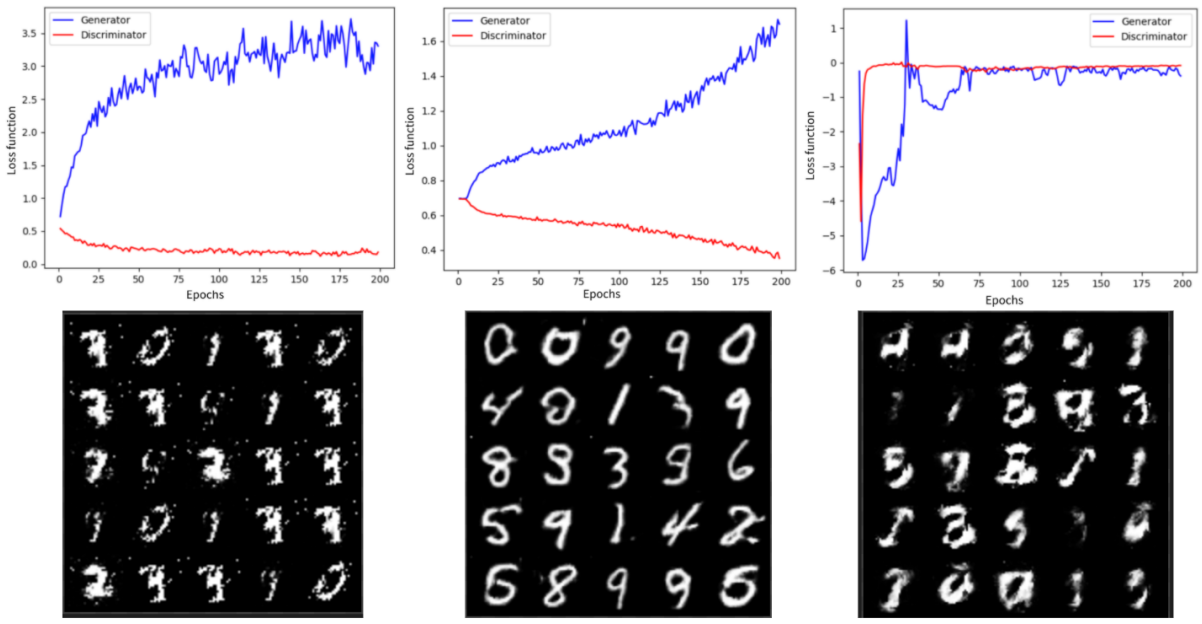


Figure 5. GANs loss function for GANs training (**Top**) as well as generated digit images after 200 epochs (**Bottom**). **Left:** Vanilla GAN, **Middle:** DCGAN, **Right**: WGAN.

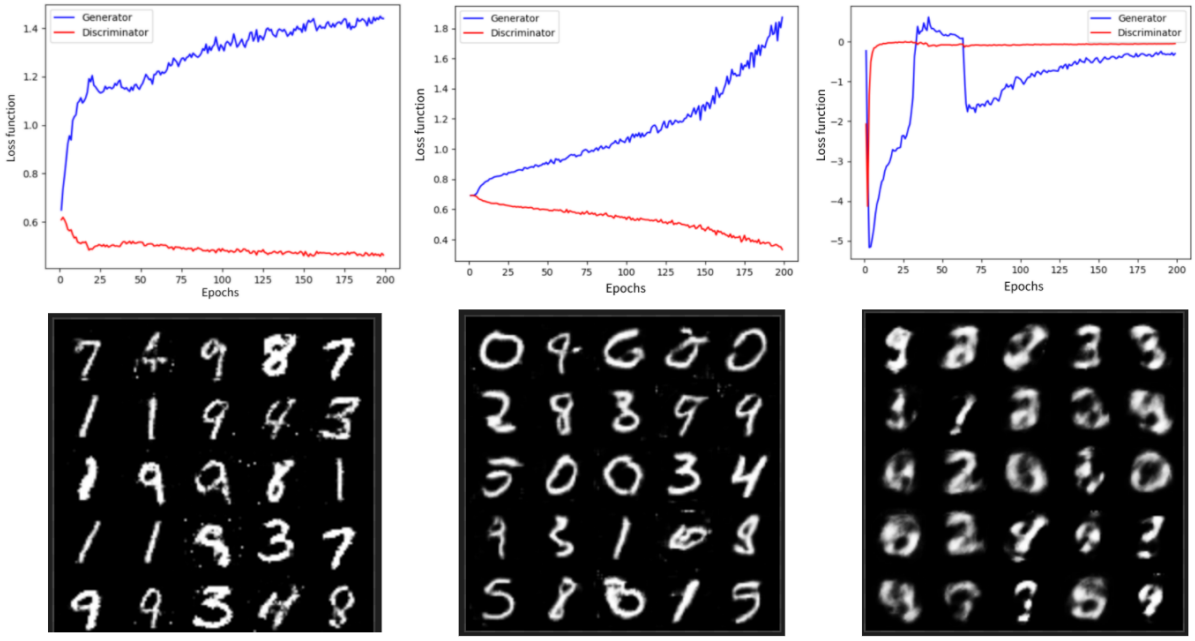


Figure 6. GANs loss function for training with Optimistic Adam (**Top**) as well as generated digit images after 200 epochs (**Bottom**). **Left:** Vanilla GAN, **Middle:** DCGAN, **Right**: WGAN.

6.1.2. Optimism

We implement an optimistic variant of Adam optimizer (OAdam). In (Daskalakis et al., 2017), the authors claim that Optimistic Adam could be of independent interest even beyond training WGANs, so we examine its performance on Vanilla GAN, DCGAN and WGAN. We adopt the same hyperparameters as the baseline models. Figure 6 shows the plots of loss function for three GANs as well as the generated digits after running 200 epochs.

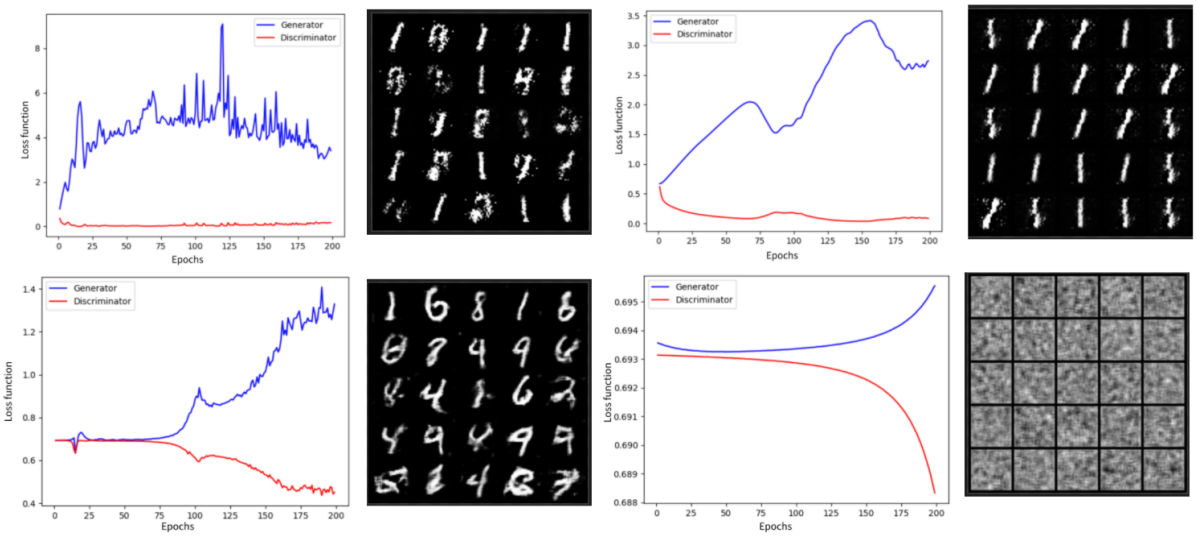


Figure 7. GANs loss function and generated digit images after 200 epochs for training with SGD with positive (**Left**) and negative momentum (**Right**). **Top:** Vanilla GAN, **Bottom:** DCGAN.

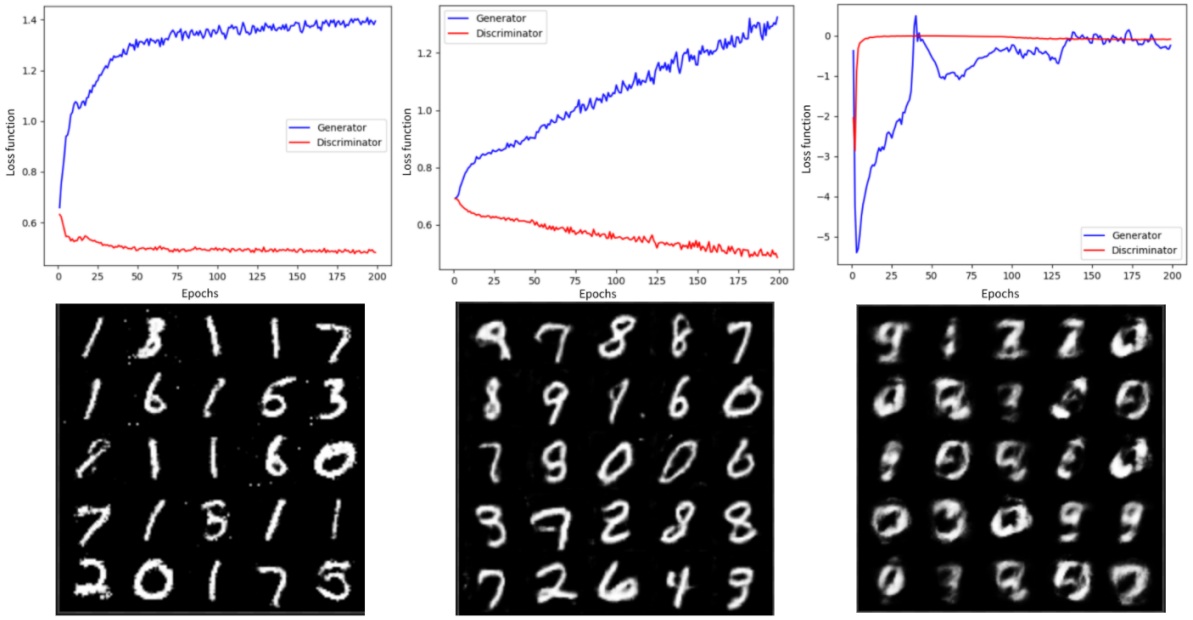


Figure 8. GANs loss function for training with Adam with negative momentum (**Top**) as well as generated digit images after 200 epochs (**Bottom**). **Left:** Vanilla GAN, **Middle:** DCGAN, **Right**: WGAN.

6.1.3. Negative Momentum

We apply negative momentum on both Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951) with momentum and Adam optimizer. For SGD with momentum, we use learning rate of 0.001, momentum value $\beta_+ = 0.9$ for regular momentum and $\beta_- = -0.5$ for negative momentum. For Adam optimizer, we follow the learning rate from the baseline models, with $\beta_{1-} = -0.5$ for negative momentum. We choose the specific negative momentum value based on the experiments conducted in (Gidel et al., 2019). In Figure 7, we present the loss function and generated images of SGD with momentum (both positive and negative). After experiments, we realize that WGAN could not converge with SGD, so we omit that from the results. Figure 8 shows the loss function and generated images of Negative Momentum with Adam on three types of GANs.

6.1.4. CONSENSUS OPTIMIZATION

We implement Consensus Optimization method for generating MNIST digit images using the DCGAN architecture. Training was performed with the Adam optimizer, with learning rate 0.001 and $\beta_1 = 0.5, \beta_2 = 0.999$, and the regularization parameter γ was set to 0.01. We report that in practice consensus optimization did not work well. While the images generated after early stopping during training looked good, the final result after full result was poor - all the generated images looked the same. Figure 9 shows the loss curves plot, while Figures 10, 11 show a sample of generated images for the early stopping and final training cases.

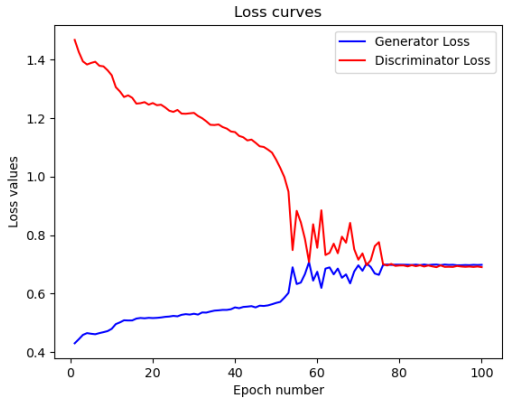


Figure 9. Generator and discriminator loss curves for consensus optimization method.

Figure 10. Generated MNIST images for consensus optimization method, after early stopping (50 epochs)

6.2. Inception Score

Inception Score (IS) (Salimans et al., 2016) is one of the most common ways to evaluate the quality of images generated by GANs. It uses an Inception Network (Szegedy et al., 2015) pre-trained on ImageNet to calculates the statistic of

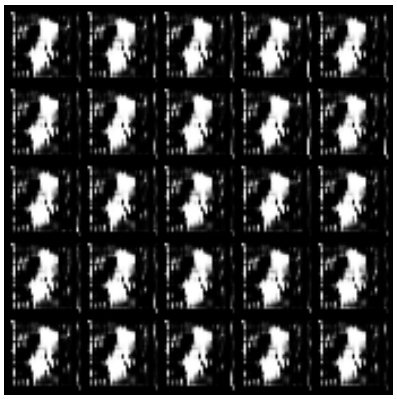


Figure 11. Generated MNIST images for consensus optimization method, after full training (100 epochs)

generated images. The score is evaluated with the equation:

$$IS(G) = \exp(\mathbb{E}_{x \sim p_g} D_{KL}(p(y|x) \parallel p(y)))$$

where $x \sim p_g$ means that x is a generated image sampled from the learned generator distribution p_g, $\mathbb{E}$ is the expected value over the set of generated images. D_{KL} is the KL-divergence between the conditional class distribution $p(y|x)$ (y is the class label) and the marginal class distribution $p(y)$ = $\int_x p(y|x)p_g(x)$. It exponentiates the results so the values are easier to compare, and it can be ignored without loss of generality when we use $\log(IS(G))$ for evaluation.

In the original paper which proposed the Inception Scores (Salimans et al., 2016), $p(y|x)$ measures whether the generated images are clear or blurry. That is, if the Inception Network is confident that there is a single object in the image, $p(y|x)$ would be high entropy. Also, $p(y)$ indicates if the generative models could output a diverse set of images for all the classes in ImageNet. Images with high diversity means a lower entropy of $p(y)$. Since $D_{KL}(p(y|x) \parallel p(y)) = p(y|x) \log(\frac{p(y|x)}{p(y)})$, if a generative network could generate diverse images with clear objects, we expect a large KL-divergence, resulting in a large Inception Score.

Tables 1, 2 present a comparison of the Inception Score on GAN training with explored optimizing strategies.

7. Discussion and Conclusions

In this work, we explored existing ideas surrounding training GANs. GANs are notoriously hard to train, and thus there have been many developments focusing on the GAN min-max optimization problem. We specifically focused on three methods, namely, training GANs with optimism, training with negative momentum, and consensus optimization. We empirically tested out each of these methods by training GANs to generate MNIST images, and we compared the

Model	IS	Model	IS	Model	IS
GAN	1.878 +/- 0.012	DCGAN	2.412 +/- 0.021	WGAN	2.355 +/- 0.017
GAN + OAdam	3.085 +/- 0.033	DCGAN + OAdam	**2.453 +/- 0.018**	WGAN + OAdam	2.026 +/- 0.009
GAN + AdamNM	**3.148 +/- 0.019**	DCGAN + AdamNM	2.427 +/- 0.010	WGAN + AdamNM	**2.459 +/- 0.015**
GAN + SGD	2.691 +/- 0.019	DCGAN + SGD	2.208 +/- 0.017	WGAN + SGD	1.007 +/- 0.000*
GAN + SGDNM	1.547 +/- 0.006	DCGAN + SGDNM	1.100 +/- 0.001*	WGAN + SGDNM	1.018 +/- 0.000*

Table 1. Inception Score of various GAN optimization experiments for training with optimism and negative momentum. IS: higher is better. Note: * means GAN not able to generate reasonable images.

Model	IS
DCGAN+ConsenusOpt (early stopping)	2.453 +/- 0.018
DCGAN+ConsensusOpt (final)	1.202 +/- 0.002

Table 2. Inception Score of Consensus Optimization method experiments. IS: higher is better.

results to standard GANs. Our conclusions are as follows -

- We can see from the Vanilla GAN baseline results that there is not a smooth convergence. Additionally, the images produced from the Vanilla GAN are a bit blurry. The DCGAN trains relatively well and produces clear images. The WGAN images produced are hardly legible.

- For training with Optimistic Adam, we theoretically expect a faster convergence rate than baseline methods, as optimistic mirror descent typically leads to faster convergence. We empirically validate theoretical claims using the MNIST dataset and can see that it converges a bit faster than the Vanilla GAN. Also, we see that the images produced from the Vanilla GAN and DCGAN are slightly clearer than the baseline results.

- For negative momentum, we theoretically and experimentally explore its effects in a GAN training setup. Theoretically, negative momentum could push the eigenvalues of the Jacobian appropriately into a smaller convergence region, thus improving the convergence rate of gradient-based methods. Experimentally, we validate the performance of negative momentum on Adam optimizer for GAN training with MNIST dataset. From Figure 8, we can see that the loss function becomes smoother and converges slightly faster than the original GAN setting. This supports the intuition that in simple yet intuitive examples, using negative momentum makes convergence to the Nash Equilibrium easier. However, we were unable to make negative momentum with SGD performing as well after a relatively thorough hyperparameter tuning, as shown in Figure 7, despite the fact that the original paper (Gidel et al., 2019) claimed it to be working.

- For consensus optimization, we were unable to reproduce the quality of results as shown in the original work (Mescheder et al., 2017), inspite of best efforts. Moreover, the code provided by (Mescheder et al., 2017) was not in a working state, and thus, we had to write our own code to perform the experiments related to this method. Initial training of GANs with this method seemed stable, the loss curves were showing the correct trend, and were also smooth and not wildly fluctuating as seen in standard GANs. However, beyond a point the results got worse and ultimately the method failed to converge properly. Initial results of images generated from the GANs also were of good quality, with a high Inception Score of 2.45, but then it seems like the training suffered from a mode collapse and eventually started generating a constant blob, and the loss curve seemed to have stabilized (See Figures 9-11, and Table 2). One guess as to why this happened may be that the hyperparameter γ which controls the regularization may be too sensitive, or that it might be stuck in a spurious minima due to introduction of the regularizer term (See `https://www.inference.vc/my-notes-on-the-numerics-of-gans/`). Overall, although the method was justified well through theory by the authors, we failed to reproduce this is in practice.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 214–223. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/arjovsky17a.html`.

Bertsekas, D. P. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H. Training gans with optimism. *arXiv 1711.00141*,

abs/1711.00141, 2017. URL http://arxiv.org/abs/1711.00141.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Denton, E., Chintala, S., Szlam, A., and Fergus, R. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

Gidel, G., Hemmat, R. A., Pezeshki, M., Le Priol, R., Huang, G., Lacoste-Julien, S., and Mitliagkas, I. Negative momentum for improved game dynamics. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1802–1811. PMLR, 2019.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of wasserstein gans. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Li, J., Madry, A., Peebles, J., and Schmidt, L. On the limitations of first-order approximation in GAN dynamics. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3005–3013. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/li18d.html.

Mertikopoulos, P., Papadimitriou, C. H., and Piliouras, G. Cycles in adversarial regularized learning. *CoRR*, abs/1709.02738, 2017. URL http://arxiv.org/abs/1709.02738.

Mescheder, L., Nowozin, S., and Geiger, A. The numerics of gans. *arXiv preprint arXiv:1705.10461*, 2017.

Mirza, M. and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Rakhlin, S. and Sridharan, K. Optimization, learning, and games with predictable sequences. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/f0dd4a99fba6075a9494772b58f95280-Paper.pdf.

Ratliff, L. J., Burden, S. A., and Sastry, S. S. Characterization and computation of local nash equilibria in continuous games. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 917–924, 2013. doi: 10.1109/Allerton.2013.6736623.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242, 2016.

Srivastava, A., Valkov, L., Russell, C., Gutmann, M. U., and Sutton, C. Veegan: Reducing mode collapse in gans using implicit variational learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/44a2e0804995faf8d2e3b084a1e2db1d-Paper.pdf.

Syrgkanis, V., Agarwal, A., Luo, H., and Schapire, R. E. Fast convergence of regularized learning in games. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.

neurips.cc/paper/2015/file/7fea637fd6d02b8f0adf6f7dc36aed93-Paper.pdf.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Wu, X., Xu, K., and Hall, P. A survey of image synthesis and editing with generative adversarial networks. *Tsinghua Science and Technology*, 22(6):660–674, 2017. doi: 10.23919/TST.2017.8195348.

Zhang, J. and Mitliagkas, I. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

CORESET SELECTION FOR DATA-EFFICIENT TRAINING

Xinyu Yao, Kejia Ren, Tianyi Zhang
Department of Computer Science
Rice University
6100 Main St, Houston, TX 77005, USA
{xy38,kr43,tz21}@rice.edu

ABSTRACT

As machine learning datasets become larger and models become deeper to reach higher accuracy, the high financial costs and substantial environmental impact associated with training machine learning models have become a more prominent problem. As a result, methods for data-efficient training of machine learning models have become a popular research area. One of the best solutions to the problem is coreset, which is a weighted subset of the data that can be used to train a model with negligible loss in accuracy. In our project, we review previous works that led to the development of coreset for deep learning, and analyze the three state-of-the-art coreset selection methods GLISTER, CRAIG, and GRAD-MATCH, and compare their performance empirically along with random selection and full gradient descent baselines. We improve GLISTER by associating weights with each element in the coreset, and show that the modified algorithm outperforms the original algorithm.

1 INTRODUCTION

Over the past two decades, the field of deep learning has made remarkable progress on various tasks, including objection recognition, language translation, product recommendation, etc (Schwartz et al., 2020). As a result, more and more efforts and resources are being expended on machine learning research every year. A problem emerged as more sophisticated machine learning models were developed: training machine learning models incur significant financial and environmental impact (Schwartz et al., 2020). Large-scale machine learning models are data-hungry, so training relies on specialized hardware such as powerful GPUs. Training consumes an enormous amount of energy and has a large carbon footprint. To make matters worse, training a single model often requires multiple runs due to hyper-parameter tuning.

One idea for tackling this problem is to find a relatively small subset of the training data that captures the "essence" of the entire training set. Training the model on that smaller subset of data can reduce the reliance on dedicated hardware and the carbon footprint of the training process while maintaining comparable model accuracy. A justification for the possibility of reducing the size of the training set while maintaining model accuracy comes from the fact that many modern image datasets have been shown to contain significant redundancies (Birodkar et al., 2019).

1.1 CORESETS

Over the past few decades, various coreset selection algorithms have been developed for areas such as machine learning, computer vision, graphics, databases, and theoretical computer science (Jubran et al., 2019). At the start of this century, coreset selection algorithms were developed and used for linear learning algorithms such as k-Means clustering (Har-Peled & Kushal, 2007) and kernel methods such as SVM (Tsang et al., 2005). More recently, as deep learning starts to gain popularity, coreset selection algorithms for training neural networks are developed to mitigate the voracity for data of deep learning models.

Coresets are weighted subsets of the data which approximate specific characteristics of the full data (Killamsetty et al., 2021). Coresets enable efficient learning at multiple levels. To begin with, learning on coresets is performed on low computational resources without requiring high numbers of GPU and CPU servers. Moreover, coresets significantly reduce the end-to-end turnaround time for multiple training runs on hyper-parameter tuning (Killamsetty et al., 2021). Coreset algorithms have broad applications for many problems, such as clustering problems (k-means and k-median clustering).

All existing coreset selection algorithms for deep learning are a form of adaptive data subset selection. While gradient descent leverages the gradients of all training instances, adaptive data subset selection produces a weighted subset of training data on which the model is trained. Furthermore, the subset is incrementally refined as learning proceeds. This iterative refinement to the training subset enables the data selection to be tailored to the learning process and the weaknesses of the model's current state.

There are four challenges in finding coresets: firstly, there is no clear guiding principle for subset selections. Selecting training points close to the decision boundary (fine tuning the decision boundary) or picking the most diverse set of data points are two ideal guiding principles (better comprehension of data distribution). Moreover, finding this coreset has to be fast since the purpose of using coreset is to speed up. In addition, one has to decide the gradient step size for each data point in the coreset. Last but not least, the method might not be well-performed on all datasets, which is saying the method has to have theoretical understanding and mathematical convergence guarantee (Mirzasoleiman et al., 2020).

1.2 Importance Sampling

Importance Sampling is a technique that focuses on informative data and disregards the data that do not contribute to the final results. This technique aims at increasing the convergence speed of SGD by focusing computation on samples that induce a change in the model parameters. The selection of coreset perform importance sampling for sensitivity score to provide high-probability solutions for different problems, like k-means and k-median clustering (Har-Peled & Mazumdar, 2004).

The current importance sampling methods can be categorized into methods applied to convex problems and methods designed for deep neural networks (Katharopoulos & Fleuret, 2018). For methods applied to convex problems, LASVM (Bordes et al., 2005) is an online algorithm that trains kernelized support vector machines using importance sampling. Another application on convex problems is to connect importance sampling and variance of the gradient estimates of stochastic gradient descent. For the problems on Deep learning, Bengio & Senécal (2008) designed a sampling scheme, and Schaul et al. (2015) used the loss to create the sampling distribution.

1.3 Incremental Gradient Methods

The incremental gradient method is an algorithm that minimizes a finite sum of a convex function. This incremental gradient method is applied to many fields, including large-scale data processing and distributed optimization over networks (Gurbuzbalaban et al., 2019). Furthermore, mathematical optimization lies at the center of large-scale machine learning training, like maximizing a submodular function, minimizing loss function, or optimizing empirical risk functions. Applying IG to the selected coreset is guaranteed to converge to the (near)optimal solution with the same convergence rates as for convex optimization (Gurbuzbalaban et al., 2019).

The optimization problem is stated as the following: Given a convex loss L, and a μ-strongly convex regularizer r, one aims to find model parameter vector θ_* over the parameter space Θ that minimizes the loss f over the training data V:

$$\theta_* \in \operatorname{argmin}_{\theta\in\Theta} \quad f(\theta),$$
$$f(\theta) := \sum_{i\in V} f_i(\theta) + r(\theta), \quad f_i(\theta) = L(\theta, (x_i, y_i)),$$

where V = {1,...,n} is an index set of the training data, and functions $f_i : \mathbb{R}^d \to \mathbb{R}$ are associated with training examples (x_i, y_i), where $x_i \in \mathbb{R}^d$ is the feature vector, and y_i is the point i's

label. Standard Gradient Descent could be used to solve this optimization problem which requires abundant computations of the full gradient $\nabla f(\theta)$.

1.4 Connection to Active Learning

Sener & Savarese (2017) formulated the active learning problem as a coreset selection problem. In an active learning problem, the model iteratively selects data points from a pool of unlabelled examples for labeling and training since the labeling process may be prohibitively expensive or time-consuming for the entire dataset. Previous works (Settles, 2009) have shown that there exist many heuristics that are computationally efficient while being capable of selecting training instances that lead to high model accuracy. Sener & Savarese (2017) observe that previous active learning methods are inefficient when applied to convolutional neural network models due to batch sampling and propose to adapt the active learning problem to batch sampling by proposing to formulate it as a coreset selection problem.

1.5 Selection via Proxy

Coleman et al. (2019) proposed selection via proxy (SVP), which works in conjunction with existing data selection methods such as active learning and coreset selection, and offers computational savings. The idea of SVP is inspired by heterogeneous uncertainty sampling (Lewis & Catlett, 1994), which leverages a less computationally expensive model to select training instances for a more computationally expensive model (e.g. using a Naive Bayes model to select training instances for a decision tree). SVP extends that idea to deep learning by scaling down large neural networks with methods such as removing layers, switching to a simpler architecture, and reducing training epochs. These simpler deep learning models, albeit much less accurate for the intended task, perform well empirically as proxy models to select training instances for their much more computationally expensive counterparts. Furthermore, Coleman et al. (2019) discover that there are high Spearman's and Pearson's correlations between the smaller model and its larger counterpart on multiple metrics. Empirical evidence suggests that when employed together with existing active learning and coreset selection methods, SVP greatly reduces the computational costs of training sophisticated deep learning models.

2 Literature Review

Inspired by previous works, multiple coreset selection algorithms that are suited for training deep learning models were developed in recent years, such as GLISTER (Killamsetty et al., 2020), CRAIG (Mirzasoleiman et al., 2020), and GRAD-MATCH (Killamsetty et al., 2021). Each of them either selects a subset or a weighted subset from the entire training dataset by optimizing over an objective function, which can be approximated by some sort of similarity between the gradients of the subset loss and the full training loss with respect to the model parameters. Some details of these three methods are briefly introduced in Sec. 2.1, Sec. 2.1, and Sec. 2.3, respectively, under the supervised-learning context. To unify the notations, we denote the entire train set as $X = \{X_i\}$, the subset as $S = \{S_j\}$ whose cardinality is a natural number k, the model parameters as a vector θ, and the loss as $L(\cdot, \cdot)$, given a differentiable model and some training instances with corresponding labels.

2.1 GLISTER

The GLISTER strategy was initially proposed with a bi-level discrete optimization formulation, where the algorithm selects a subset such that the loss of the entire training dataset is minimized by the model trained with this subset. However, due to the complexity of the inner-optimization, the authors of GLISTER adopted an online approximation to the inner-optimization solution by one-step gradient descent. Additionally, to further reduce computation, the outer-optimization is also approximated by the first-order Taylor expansion. Mathematically, such approximation process can

be described by the following derivations, where η is the step size of the gradient update.

$$\begin{aligned}
&\min_{S\subseteq X,|S|\leq k} L(\text{argmin}_\theta L(\theta, S), X) \\
\approx &\min_{S\subseteq X,|S|\leq k} L(\theta - \eta\nabla_\theta L(\theta, S), X) \\
\approx &\min_{S\subseteq X,|S|\leq k} L(\theta, X) - \eta\nabla_\theta L(\theta, S)^\top \nabla_\theta L(\theta, X) \\
= &\max_{S\subseteq X,|S|\leq k} \nabla_\theta L(\theta, S)^\top \nabla_\theta L(\theta, X)
\end{aligned}$$

After the above approximation, it is apparent that the original bi-level optimization is reduced to maximizing the similarity of the subset gradient and the full gradient represented by their inner-product, which can be solved easily by a naive greedy selection algorithm. Unlike CRAIG and GRAD-MATCH which use a weighted subset, GLISTER uses a unweighted subset as the coreset. In other words, each data sample in the coreset computed with GLISTER takes equal importance in updating the model.

2.2 CRAIG

Unlike GLISTER, CRAIG measures the gradient similarity between the subset and training set by Euclidean distances, and its objective takes the form of the following expression:

$$\min_{S\subseteq X,|S|\leq k} \sum_{X_i\in X} \min_{S_j\in S} \|\nabla_\theta L(\theta, S_j) - \nabla_\theta L(\theta, X_i)\|$$

The above optimization problem is categorized as a Facility Location problem which is a well-studied generic submodular optimization problem. In a general form, the Facility Location problem looks for a selector for optimizing a function $F(U, V) = \sum_{u\in U} \max_{v\in V} \phi(u, v)$, where V is the subset, U is the groundset, and $\phi(u, v)$ evaluates a score to reflect the similarity between u and v. Fortunately, the Facility Location problem can be efficiently solved in many ways, which mitigates the implementation difficulty of CRAIG.

One important characteristic of CRAIG is that it implicitly uses a weighted coreset for training. Concretely, CRAIG associates an element in the coreset with each instance in the training set based on the highest similarity. Then, for each instance in the coreset, CRAIG counts the number of training instances it is associated with, and uses that number as the weight of the gradient for updating the model, as expressed by the following equations:

$$\begin{aligned}
&\sum_{S_j\in S} w_j \nabla_\theta L(\theta, S_j), \\
&w_j = \frac{\sum_{X_i\in X} \mathbb{I}\Big[S_j = \text{argmin}_{s\in S} \|\nabla_\theta L(\theta, s) - \nabla_\theta L(\theta, X_i)\|\Big]}{|X|}
\end{aligned}$$

2.3 GRAD-MATCH

Instead of counting assignments to weight coreset data, GRAD-MATCH directly optimize the value of weights $\mathbf{w}$ in a continuous space. Its objective takes the following form, which has been proved to be upper-bounded by the objective of CRAIG (Killamsetty et al., 2021).

$$\min_{\mathbf{w}, S\subseteq X,|S|\leq k} \sum_{S_j\in S} \|w_j \nabla_\theta L(\theta, S_j) - \nabla_\theta L(\theta, X)\|$$

The above objective of GRAD-MATCH is sovled using the Orthogonal Matching Pursuit (OMP) algorithm (Elenberg et al., 2018), which is an iterative algorithm for matrix recovery. Breifly speaking, at each iteration, OMP greedily selects one data to be added into the coreset based on its correlation with the current residuals and optimizes the weight vector based on the current coreset selection until the coreset cardinality constraint becomes active.

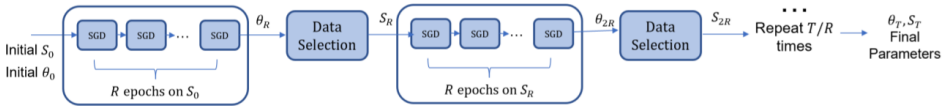
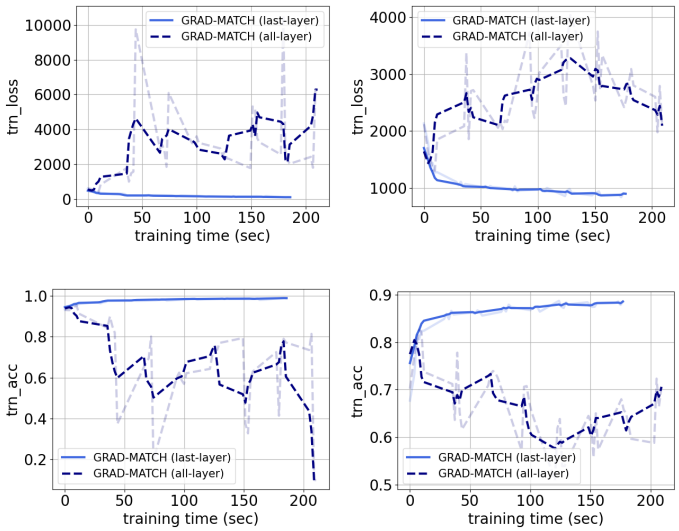


Figure 1: Data selection for every R epochs (Killamsetty et al., 2021).



(a) MNIST (b) Fashion-MNIST

Figure 2: Experimental results of GRAD-MATCH on the MNIST and Fashion-MNIST datasets, by using last-layer or all-layer gradients to select the coreset. The first row and the second row are respectively the plots for train loss and train accuracy with respect to the training time in seconds.

2.4 Techniques for Speeding Up Coreset Selection

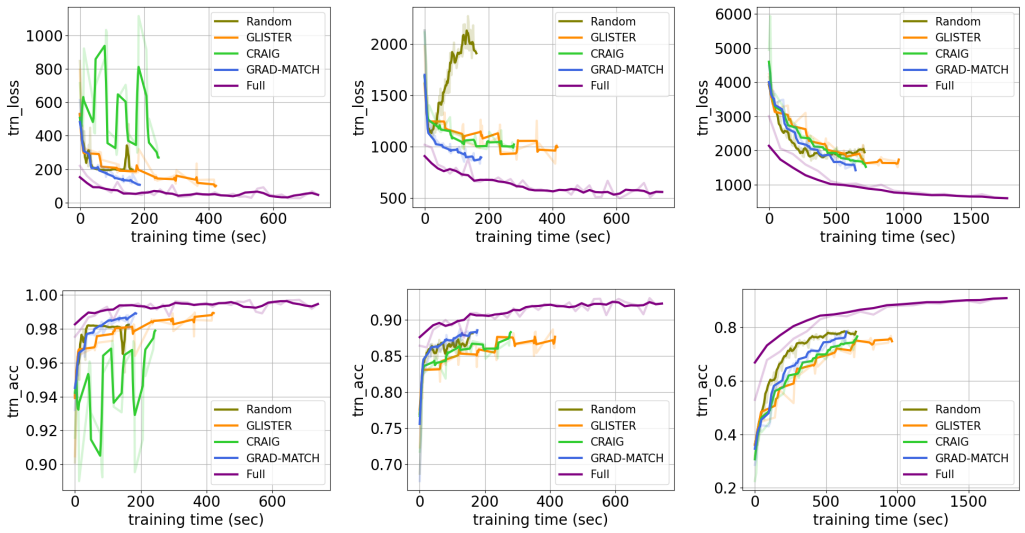
Last-layer gradients: Using gradients of all parameters of a model to compute coresets is computationally expensive. Since the variation of the gradient norm is mostly captured by the gradients of the last layer, we only use the last-layer gradients to compute coresets (Killamsetty et al., 2021).

Per-class and per-batch approximations: Storing gradients of all instances leads to high memory requirements. Instead, we solve the gradient matching problem for each class individually. Alternatively, we can consider only the gradients of entire batches (Killamsetty et al., 2021).

Warm-starting: Initially, we train the model with full gradients for T epochs without adaptive data selection, in order to stabilize gradients (Killamsetty et al., 2021).

Perform data selection every R epochs: For any adaptive data selection algorithm, the data selection is performed every R epochs and the (stochastic) gradient descent updates are performed on the subsets obtained by the data selection, as illustrated in Fig. 1 (Killamsetty et al., 2021).

To verify the effectiveness of the speed-up technique of using only the last-layer gradients, we run experiments of training a neural network classifier on the MNIST and Fashion-MNIST datasets, and comparing using gradients from the last layer with using all gradients for the GRAD-MATCH algorithm. The results are shown in Fig. 2. The results show that using only the last-layer gradients could potentially stabilize the training. We hypothesize that it is because the gradients from all layers have greater variance and thus make training more sensitive to factors such as noise. More importantly, when running the experiments, we discovered that using all-layer gradients is not practical since the computation of such high-dimensional gradients is not only slow but requires much memory especially for a deep model.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Figure 3: Experimental results of the performance evaluations on each algorithm. (a) and (b) show the results on the MNIST and Fashion-MNIST dataset using a LeNet model; (c) shows the result on the CIFAR-10 dataset using a ResNet18 model. The first row and the second row are respectively the plots for train loss and train accuracy with respect to the training time in seconds.

3 Contributions and Results

In this project, we are especially interested in understanding the connections and differences of GLISTER, CRAIG and GRAD-MATCH, analytically and empirically. We first evaluate to compare the performance of these three methods in Sec. 3.1. Based on the experimental results, we propose a modification to the original algorithms for tentative improvements in Sec. 3.2.

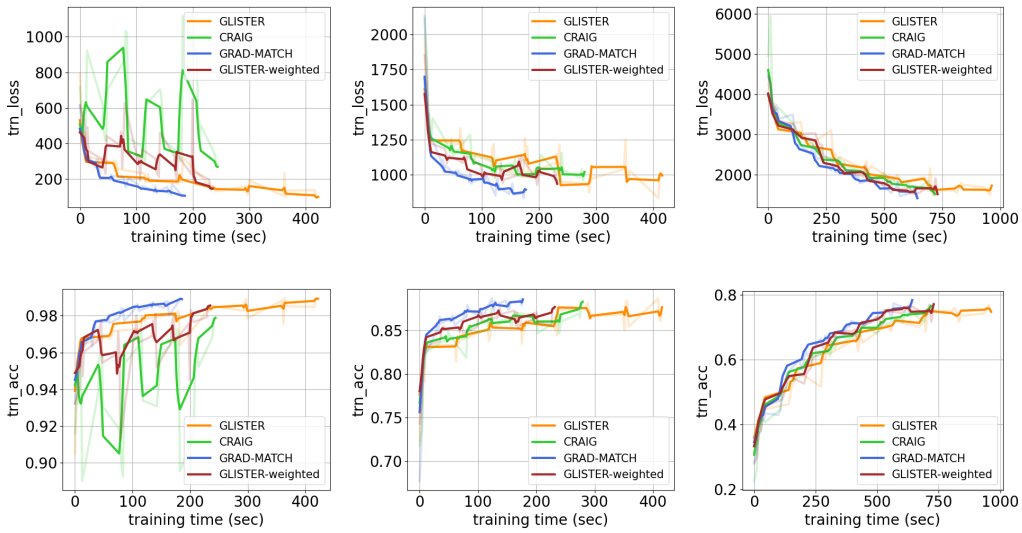
3.1 Performance Evaluation

The performance of the three algorithms are evaluated individually by our experiments, with comparison to a baseline "Random" strategy and the "Full" strategy. The "Random" strategy simply selects coresets by uniformly random sampling, and the "Full" strategy means train using the entire dataset rather than a subset. We test all the strategies on three well-known image classification datasets: MNIST, Fashion-MNIST, and CIFAR-10. For a fair comparison, we set the hyperparameters unchanged across all the trials discussed in this section. Specifically, we set the batch-size to 20, the selection fraction to 0.1 (ie. $k = 0.1 \cdot |X|$), and a coreset re-selection for every 5 epochs. The models are all trained with a stochastic gradient descent optimizer scheduled by cosine annealing. In almost all the experiment runs, the model is trained for 40 epochs; however, for the "Full" strategy on CIFAR-10, we truncate the training time to be only 20 epochs due to its low speed. The results are summarized in Fig. 3, where the slope of the curve indicates the efficiency of an algorithm.

From the results, the "Full" strategy can completely capture all the underlying information in the train dataset, and thus it achieves the best performance in terms of train metrics and serves as a standard in our experiments. In practice, the "Full" strategy runs more iterations in one epoch than the other coreset algorithms by $|X|/k$ times. For the coreset strategies, GRAD-MATCH is the optimal in all experiments. CRAIG and GLISTER performs similarly well, but GLISTER requires significantly more computation time for subset re-selection than CRAIG and GRAD-MATCH.

3.2 Tentative Improvement

As described in the previous section, one major difference of these three methods is that GLISTER uses an unweighted subset while GRAIG and GRAD-MATCH use weighted subsets. Additionally,



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Figure 4: Experimental results on evaluating the weighted GLISTER by comparison to the other three coreset algorithms. (a) and (b) show the results on the MNIST and Fashion-MNIST dataset using a LeNet model; (c) shows the result on the CIFAR-10 dataset using a ResNet18 model. The first row and the second row are respectively the plots for train loss and train accuracy with respect to the training time in seconds.

unlike CRAIG which counts the discrete data assignments as weights, GRAD-MATCH directly optimizes the weights in the continuous space. Theoretically, the solution found by GRAD-MATCH or CRAIG should be much more optimal than GLISTER since the use of the weights enlarges the possible solution space for search and the speed of GLISTER is slow due to its computational complexity.

However, from the experimental results shown above, we find GLISTER performs much better than we expected in many scenarios. This observation led us to a hypothesis, that the weighted subsets are more optimal while the inner-product approximation used by GLISTER still has some advantages for alleviating errors. Therefore, we tentatively modify the original GLISTER by adding weights into coreset data in the following way, where the modified objective function takes the form of a weighted sum of inner-products:

$$\max_{S\subseteq X,|S|\leq k} \nabla_\theta L(\theta,S)^\top \nabla_\theta L(\theta,X) = \max_{S\subseteq X,|S|\leq k} \sum_{S_j\in S} \nabla_\theta L(\theta,S_j)^\top \nabla_\theta L(\theta,X)$$

$$\longrightarrow \max_{\mathbf{w},S\subseteq X,|S|\leq k} \sum_{S_j\in S} w_j \nabla_\theta L(\theta,S_j)^\top \nabla_\theta L(\theta,X)$$

To simplify the implementation, we further approximate the above weighted objective by a Facility Location function in a similar way as in CRAIG:

$$\max_{\mathbf{w},S\subseteq X,|S|\leq k} \sum_{S_j\in S} w_j \nabla_\theta L(\theta,S_j)^\top \nabla_\theta L(\theta,X) \approx \max_{S\subseteq X,|S|\leq k} \sum_{X_i\in X} \max_{S_j\in S} \nabla_\theta L(\theta,S_j)^\top \nabla_\theta L(\theta,X_i)$$

In other words, the weights used by the modified GLISTER are approximately optimized by the following equation, where S_j and w_j denote the j-the data point in the coreset and its associated weight value.

$$w_j = \frac{\sum_{X_i\in X} \mathbb{I}\Big[S_j = \text{argmax}_{s\in S} \nabla_\theta L(\theta,s)^\top \nabla_\theta L(\theta,X_i)\|\Big]}{|X|},$$
$$\text{s.t.} \quad S\subseteq X, \quad |S|\leq k$$

We test the weighted GLISTER algorithm by the same experimental setting. The results are shown in Fig. 4. We can observe from our experiments that the performance is improved relative to the original GLISTER algorithm. Thanks to the speed-up by a Facility Location approximation, the efficiency is also improved a lot. However, our modified version of GLISTER still cannot outperform GRAD-MATCH in whichever case since the weights are optimized in a discrete space which is intuitively more sub-optimal.

4 CONCLUSIONS

In this project, we reviewed previous works that led to the development of coreset selection algorithms for deep learning, and the three current state-of-the-art coreset algorithms for deep learning: GLISTER, CRAIG, and GRAD-MATCH. We verified with experiments that the last-layer gradient updates speed-up is effective for GRAD-MATCH in practice. Not only does it reach much lower training loss than the baseline in the same amount of time, the gradients of the last-layer parameters are much more stable.

Moreover, by empirically comparing the performance of the three algorithms on training models for popular image classification benchmark datasets, we conclude that GRAD-MATCH is the best performing coreset selection algorithm, since it reaches the lowest loss and the highest accuracy among all coreset selection algorithms in the same span of time. Furthermore, we noticed that GLISTER uses an unweighted subset as the coreset and hypothesize that its performance can be improved by simply associating each element in the subset with a weight, similar to the best performing GRAD-MATCH algorithm. We show that empirically, the modified version of GLISTER, GLISTER-weighted, performs better than the original algorithm, confirming our hypothesis, albeit it does not reach the efficiency of GRAD-MATCH.

Another important observation is that, surprisingly, the "Full" strategy works better than all coreset selection methods considered in our experiments, demonstrated by lower training loss and higher accuracy than other methods for fixed training time. We hypothesize that this is because full gradient descent gives more stable gradients at the start of the training process, and the size of the models and datasets we considered are not sufficiently large to demonstrate the efficiency advantages of coresets.

REFERENCES

Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722, 2008.

Vighnesh Birodkar, Hossein Mobahi, and Samy Bengio. Semantic redundancies in image-classification datasets: The 10% you don't need. *arXiv preprint arXiv:1901.11409*, 2019.

Antoine Bordes, Seyda Ertekin, Jason Weston, Léon Botton, and Nello Cristianini. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6(9), 2005.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*, 2019.

Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *The Annals of Statistics*, 46(6B):3539–3568, 2018.

M Gurbuzbalaban, Asu Ozdaglar, and Pablo A Parrilo. Convergence rate of incremental gradient and incremental newton methods. *SIAM Journal on Optimization*, 29(4):2542–2565, 2019.

Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, 2007.

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. pp. 291–300, 2004.

Ibrahim Jubran, Alaa Maalouf, and Dan Feldman. Introduction to coresets: accurate coresets. *arXiv preprint arXiv:1910.08707*, 2019.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. *arXiv preprint arXiv:2012.10630*, 2020.

Krishnateja Killamsetty, Durga Sivasubramanian, Baharan Mirzasoleiman, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: A gradient matching based data subset selection for efficient learning. *arXiv preprint arXiv:2103.00123*, 2021.

David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine learning proceedings 1994*, pp. 148–156. Elsevier, 1994.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Burr Settles. Active learning literature survey. 2009.

Ivor W Tsang, James T Kwok, Pak-Ming Cheung, and Nello Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(4), 2005.

Literature Review for Neural Network Training and Convex Optimization

Debolina Halder Lina [*1] **Yufeng Yang** [*1]

Abstract

This literature review aims to discover existing techniques of convex optimization in Neural Networks. The first two papers are summarized by Debolina Halder Lina, and the last three are summarized by Yufeng Yang. The first two papers applies L1 regularization to make the problem finite. Convex neural network solves an exact or approximate linear classifier at each step. Global convergence bound can be achieved for L1 regularization with Frank Wolfe algorithm. The third paper introduced a modified convex objective function for neural network, which preserve the same optimal value but converges much faster. The forth and fifth paper uses the convexity in dual problem to compute the layer weight via standard convex optimization technique, which can achieve higher accuracy than stochastic gradient descent(SGD). Related Numerical experiments are also summarized to support the role convex optimization taking in neural network training.

1. Introduction:

With the increasing demand for automated systems, deep neural networks (NN) are taking over a variety of aspects like text mining, recommendation system, image processing, speech recognition, bioinformatics, etc. Neural networks are a collection of nodes commonly called neurons stacked into layers consisting of one input layer, one or multiple hidden layers, and one output layer. Figure 1 shows the structure of a neural network. The neurons receive a signal from another neuron, processes it, and transmit the signal to another layer. The output of each neuron is the linear sum of its input and a non-linear activation of the linear summation. The output of a neural network can thereby be written like the following-

$$f(w^T x + w_0) \tag{1}$$

Here x is the input vector, w is the input weights and w_0 is the bias term. f is the nonlinear activation function. ReLU, Sigmoid, tanh, etc. are some widely used activation functions for neural networks. Features at the input layer of the NN propagate through the network. They first go through an affine transformation and then a non-linear activation. Due to this non-linear activation function, the optimization of the neural network is non-convex. One major concern of optimizing neural networks is that it has infinitely many local optimal values. Another issue is exploding and vanishing gradient due to sharp and flat valleys of the loss function. Despite being a non-convex problem, if sufficiently large neural networks are used then the local optimal incurs a very low loss. Then they can be considered as a global minimum if the loss is sufficiently low. So, if infinitely large neural networks are considered, then we can say that they are asymptotically convex. The goal of this study is to summarize different techniques of convex optimization of neural networks. We have mainly focused on how the authors are modifying the neural networks to construct a convex optimization function and what approaches they are taking to optimize the loss function. Many mathematical details are excluded from the summary to ensure brevity.

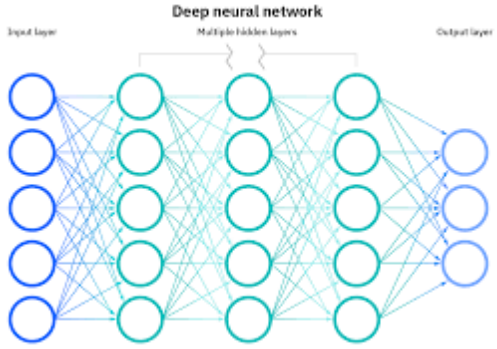


Figure 1. Deep Neural Networks

2. Convex Optimization of Neural Network:

2.1. Convex Neural Network

Training multi-layer neural network involves infinitely many variables. Bengio et. al (Bengio et al., 2006) showed that

[*]Equal contribution [1]George Brown School of Engineering, Rice University. Correspondence to: Debolina Halder Lina <dl73@rice.edu>, Yufeng Yang <yy94@rice.edu>.

Project report for COMP414/514, Rice CS.

this infinite variable problem can be converted to a convex optimization problem if the output loss function is convex in the neural network(NN) output and the output layer weights are regularized by a convex penalty. If precision limitations are imposed on all hidden units of a NN, a countable or even a finite set can be obtained. For such a NN, at the end what we get is an ordinary feedforward NN. This can be achieved by using a regularization penalty on the output weights that yield sparse solutions, such as the L1 penalty. Let $x \in \mathbb{R}^d$ be the data matrix. $\bar{x} \in \mathbb{R}^{d+1}$ be an extension of the data matrix with one element 1. Let $h_i(x) = s(v_i.\bar{x})$ where s is the nonlinear activation function. Then the prediction is $\hat{y} = \sum_{i=1}^{m} w_i h_i(x)$. The goal is to learn m, w_i's and v_i's. Let Q be a convex regularization function and $\Omega(w) = \lambda\|w\|_1$. Then the cost function becomes-

$$C = \lambda\|w\|_1 + Q(w.h(x_i), y_i) \tag{2}$$

So, Q and Δ is convex in w and C is a summation of both of them. So, C is convex in w. NN can be seen as a convex optimization problem for infinite-dimensional space as long as the number of hidden units can be selected by the learning algorithm(Bengio et al., 2006). have chosen a regularizer that generates a sparse solution to control the number of active hidden units and to transfer the problem into infinite dimensions. They have proved this in the case of hinge loss. In terms of hinge loss, $Q(y, \hat{y}) = max(0, 1 - y\hat{y})$. For this case the cost function is-

$$C(w) = K\|w\|_1 + \sum_{t=1}^{n} max(0, 1 - y_t w.h(x_t)) \tag{3}$$

As constrained optimization problem, the cost function can be rewritten as the following-

$$\begin{array}{ll} \min_{\xi,w} K\|w\|_1 + \sum_{t=1}^{n} \xi_t & \\ s.t. \quad y_t[w.h(x_t)] \geq 1 - \xi_t & (C_1) \\ \xi_t \geq 0, \ for\ t = 1, 2.....n & (C_2) \end{array} \tag{4}$$

By taking the Lagrange multipliers of the constraints and the dual of this problem ($\max \sum_t q_t h_i(x_t)$), according to (Hettich & Kortanek, 1993), the solution of the dual can be attained with constraints C_2 and only $n + 1$ con- straints C_1. Their incremental ConvexNN algorithm is summarized in Algorithm- 1 They have proved that at termination the algorithm reaches global optima but it needs solving a sub-linear problem involving a linear classifier with weighted errors. Another property of the algorithm is it behaves like boosting in the case of sign function when a linear classifier that minimizes the weighted cost is selected. In step 6, the algorithm requires to find the linear classifier. This is an NP-hard problem. They have mathematically proved that finding the linear classifier can be achieved in $O(n^3)$ steps when the input dimension is 2. It can also be extended to multidimensional setting (d) where the complexity becomes

Algorithm 1 ConvexNN

Input: training set $D = (x_1, y_1), ..., (x_n, y_n)$, convex loss function Q, and scalar regularization penalty λ. s is either the sign function or the tanh function.

1: Set $v1 = (0, 0, ..., 1)$ and select $w_1 = \min_{w_1} \sum_t Q(w_1 s(1), y_t) + \lambda\|w_1\|$
2: i = 2
3: **repeat**
4: Let $q_t = Q'(\sum j = 1^{i-1} w_j h_j(x_t), y_t))$
5: **if** s = sign **then**
6: train linear classifier $h_i(x) = sign(v_i.\bar{x})$ with examples $(x_t, sign(q_t))$ and errors weighted by mod q_t, for $t = 1...n$ (i.e maximize $\sum_t q_t h_i(x_t)$)
7: **else if** s = tanh **then**
8: train linear classifier $h_i(x) = tanh(v_i.\bar{x})$ to maximize $\sum_t q_t h_i(x_t)$
9: **end if**
10: **until** $\sum_t q_t h_i(x_t) \leq \lambda$
11: Select $w_1, ..., w_i$ (and optionally $v_2, ..., v_i$) minimizing $C = \lambda\|w\|_1 + Q(w.h(x_i), y_i)$
12: return the predictor $\hat{y_i} = \sum_j = 1^i w_j h_j(x)$

$O(log(n)n^d)$. However, for higher dimensions, finding the exact minimizer is not a practical idea. Step 8 trains an approximation instead of an exact minimizer. Popular approximation techniques to find a linear classifier with the weighted cost is linear SVM, logistic regression, and the Perceptron algorithm. Considering the hinge loss doesn't generate a solution close to the exact minimum which is proved by their experimental results. In step 13, both w and v is optimized simultaneously. At the end of each stage, with a few training iterations of the whole NN using an ordinary gradient descent mechanism, one optimizes w_j's and the v_j's. Then v_j's are fixed and w_j's are optimized for that v_j's. They have tested the algorithm with an exact and approximate linear classifier using a 2-d double moon toy dataset. The approximate algorithm has yielded an average test classification error of 3.68% and the exact algorithm has yielded an average test classification error of 5.3%.

2.2. Global Convergence of Frank Wolfe on One Hidden Layer Networks

Conditional gradients Frank-Wolfe algorithm is one of the well-known incremental algorithms for training one single hidden layer network. In a constrained minimization problem where projection on a set is hard, solving a Linear Minimization Oracle (LMO) is often easy using Frank Wolfe. Aspremont at. al (d'Aspremont & Pilanci, 2020) have shown that the LMO for one hidden layer NN can efficiently be solved under overparameterized and mild preconditioning assumptions. They have also shown that the

overparameterized setting has a convex epigraph and no duality gap. They have focused on training the infinitely wide NN which is asymptotically convex. Like (Bengio et al., 2006) they have also considered a L_1 penalty to decide the locations of active neurons via the LMO. Each LMO adds a fixed number of neurons to the solution (In the case of (Bengio et al., 2006), each iteration added a single neuron to the solution. 1. The objective function can be written as follows-

$$\begin{array}{l} minimize \sum_{i=1}^{n}(f(a_i)-y_i)^2 \\ subject\ to \quad \gamma_1(f) \leq \delta \end{array} \tag{5}$$

Here $\gamma_1$is the variation norm(an extension to L_1 norm in infinite dimension). $A \in R^{n\times d}$ is the input data matrix, y is the output, and

$$f(a_i) = \int \sigma_\theta(a_i) d\mu(\theta) \tag{6}$$

Though equation 5 is non-convex, for a large number of neurons and higher dimension, the objective becomes very close to convex. If started from one single unit, then the objective function becomes-

$$\begin{array}{c} minimize \quad \|z-y\|_2^2 \\ subject\ to \quad \sigma(\theta^T a_i) = z_i,\ for\ i = 1,..,n \end{array} \tag{7}$$

Though this is non-convex, it's epigraph is convex when the number of neurons are greater than the number of sample. After taking the convex hull of its epigraph, the objective becomes-

$$\begin{array}{c} minimize \quad \|z-y\|_2^2 \\ subject\ to \quad \sum_{j=1}^{n+2} \alpha_j \sigma(\theta_j^T a_i) = z_i,\ for\ i = 1,..,n \end{array} \tag{8}$$

So, in the case of training one single hidden layer network, the problem can be treated as convex when the number of neurons exceeds the sample size. The activation function σ_θ is parameterized by $\theta \in V$ where V is a compact topological vector space. $f(a_i) = \int \sigma_\theta(a_i) d\mu(\theta)$ is an action of the Radon measure μ on the activation function. This is an infinite dimension problem that can be solved by Frank Wolfe where the LMO is solved over a γ_1 ball. The Frank Wolfe invokes the LMO using the gradient at each iteration. Then the algorithm takes convex combinations of iterations. If the loss function if $L(f)$, then

$$L(f) = \sum_{i=1}^{n}(f(a_i)-y_i)^2 \tag{9}$$

$$L'(f) = \sum_{i=1}^{n} g_i \delta \tag{10}$$

where,

$$g_i = 2(\int \sigma_\theta(a_i) d\mu(\theta) - y_i) \tag{11}$$

So the LMO that the Frank Wolfe solves becomes-

$$minimize \sum_{i=1}^{n} g_i f(a_i) \tag{12}$$

By switching sums-

$$\begin{aligned} &\inf_{\gamma_1(\int \sigma_\theta() d\mu(\theta) \leq 1} \sum_{i=1}^{n} g_i (\int \sigma_\theta(a_i) d\mu(\theta)) \\ &= \inf_{\gamma_1(\int \sigma_\theta() d\mu(\theta) \leq 1} (\int (\sum_{i=1}^{n} g_i \sigma_\theta(a_i)) d\mu(\theta)) \\ &\geq -\max_{\theta \in v} \sum_{i=1}^{n} g_i \sigma_\theta(a_i), \end{aligned} \tag{13}$$

if and only if $\mu = \mu_- - \mu_+$. So, the key of solving the LMO is solving $\max_{\theta\in v} \sum_{i=1}^{n} g_i \sigma_\theta(a_i)$ The authors have solved this maximization problem for ReLU activation function. The overall algorithm is described in Algorithm 2. After T iterations, $L(\int \sigma_\theta() d\mu_T(\theta)) - L^\star \leq$

Algorithm 2 Frank-Wolfe Algorithm

Input: A target precision $\epsilon > 0$

1: Set $t := 1$, $\mu_1(\theta) = 0$.
2: **repeat**
3: Get $\mu_d(\theta)$ solving (LMO) for g_i
4: $g_i = 2(\int \sigma_\theta(a_i) d\mu_t(\theta) - y_i)$ for $i = 1, ...n$
5: Set $\mu_{t+1(}\theta) := (1-\lambda_t)\mu_t(\theta) + \lambda_t \mu_d(\theta)$ for $\lambda := 2/(t+1)$
6: Set t:= t+1
7: **until** gap$_t \leq \epsilon$

Output: $\mu(\theta) t_{max}$

$\frac{4R^2\delta^2}{T+1}$ where $R^2 = \sup_{\theta\in V}\{\sum_{i=1}^{n} \sigma_\theta(a_i)^2\}$. Frank Wolfe also gives an upper bound of the duality gap $gap_t = \sum_{i=1}^{n} g_i(\int \sigma_\theta(a_i) d\mu_t(\theta) - \int \sigma_\theta(a_i) d\mu_d(\theta))$ where $\mu_t(\theta)$ is the current state and $\mu_d(\theta)$ is the solution of the linear minimization oracle. When the number of sample n is greater than the dimension d, the minimization objective becomes a finite sum problem. For this scenario, the authors have proposed a stochastic Frank Wolfe algorithm which solves a linear minimization oracle at each iteration on a subset of the samples. The sample size m_t is equal to $(\frac{G(t+1)}{LD})^2$. Here L is the Lipschitz constant and G is the upper bound of the Lipschitz constant of the gradient of the objective function. For small m_t, the stochastic Frank Wolfe is similar to Algorithm 2. The authors have tested the convergence of Frank Wolfe and Stochastic Frank Wolfe algorithms using toy examples. In the first case, the ground truth is generated using ten neurons in dimension 25 using Gaussian weights, observing 20 data points. In the case of Stochastic Frank Wolfe, the ground truth is generated by ten

neurons, in dimension 20 using 25 samples. The empirical results show that overparameterized networks are inherently easier to train.

2.3. Convex Relaxation for shallow neural networks

In this paper, the neural network model the author considers is shallow network, which only contains one hidden layer.it is much easier to analyze. Also, with the number of neurons increases, shallow neural network can approximate any functions,which has important practical significance. In the following part, the derivation of convex relaxation will be shown from the simplest case (i.e. one neuron in hidden layers) to multiple neuron case. The author shows such generalization is always held.

For single neuron case, assume there exists planted parameter x^*, the model considered here is:

$$y = g(Ax^*) \tag{14}$$

the objective function is:

$$\min_x \frac{1}{2} f(x) = \min_x \frac{1}{2} \|g(Ax) - y\|_2^2 \tag{15}$$

Where $g(.)$ is the ReLU activation function: $g(x) = max(0, x)$, $A \in R^{n\times d}$ is the input data matrix, $x \in R^d$ is the parameter vector for first layer and y is considered from the planted model such that $y = g(Ax^*)$

Taking gradient of $f(x)$:

$$\nabla f(x) = A^T D(g(Ax_t) - y) \tag{16}$$

Where D_t is the diagonal matrix and its i^{th} diagonal entry is computed as 1 if $a_i^T x \geq 0$, 0 otherwise.

The objective above function may not be a convex function because of the existence of ReLU.However, if we expand Objective function $f(x)$ and relax it as:

$$f_r(x) = \|g(Ax)\|_2^2 - 2y^T Ax + \|y\|_2^2 \tag{17}$$

The author shows that $f_r(x)$ is convex (Proposition 1.) and also holds the same optimal value as $f(x)$. (Theorem 1.)(Ergen & Pilanci, 2019). Thus, by convex function property, $f_r(x)$ will only have one global minimum. Gradient descent with proper step size will converge to the global minimum. When it comes to the hidden layer with multiple neurons, instead of considering the model $y = g(Ax^*)$, the model is:

$$y_i = \sum_1^m g(a_i^T x_j^*) \forall i \in 1....n \tag{18}$$

x_j^* is the planted parameter vector of each neuron. Similarly as single neuron case, the objective function is defined as:

$$\min_X f_m(X) = \min_X \frac{1}{2} \left\| \sum_{j=1}^m g(Ax_j) - y \right\|_2^2 \tag{19}$$

Where $g(.)$ is still the ReLU activation function, $A \in R^n$ is input matrix.

The objective function has multiple global minimum points, because if we permute x_j from different orders, they are all global optimum solutions. If $n > d$,we can obtain more equations than true parameters, thus, gradient descent still works in this case. If $n \leq d$, in this case, we cannot compute all the true parameters. Gradient descent may fail to converge to the global optimum. Thus, the importance of convex relaxation will show up.

If we denote the residual as $r = \sum_{j=1}^m g(Ax_j) - y$, the gradient can be denoted as:

$$\nabla x_j f_m(X) = 2A^T D_j r \tag{20}$$

Apply first-order condition (i.e $\nabla x_j f_m(X) = 2A^T D_j r = 0 \ \forall j = 1..m$), if A is assumed to be full rank, then the optimal condition is achieved if and only if $D_j r = 0 \ \forall j = 1..m$. However, D_j is a low rank diagonal matrix, if the corresponding element is 0, then the corresponding index in r can be arbitrary number, which indicates gradient descent may fail to converge to the optimal value.

In order to overcome the problem taken by the low rankness of D_j, the modified gradient descent algorithm is:

$$\nabla x_j f_m(X) = 2A^T D_j r + 2A^T \prod_{j=1}^m D_j^c r \tag{21}$$

Where $D_j^c = I_n - D_j$. With this modification, we can guarantee that each index of residual has a positive multiplicative factor so that $\nabla_{x_j} f_m(X) = 0$ implies $r = 0$.

The corresponding function satisfies the modified gradient is:

$$f_{mp}(X) = \left\| \sum_{j=1}^m g(Ax_j) - y \right\|_2^2 - 2y^T \prod_{j=1}^m D_j^c A \sum_{j=1}^m x_j \tag{22}$$

It is proved that the relaxation of original objective function is convex (Ergen & Pilanci, 2019) and it preserve the same optimal value as the original objective function f_m. Thus, applying gradient descent with proper step size, it can guarantee modified gradient descent converge to the unique optimal value.

2.3.1. Numerical Experiments

The author generates the input and output data and initialize the parameter vector/matrix according to standard normal distribution. Then, the author uses shallow network with single neuron and multiple neurons to show the training and test performance,which are in Figure 1 and Figure 2 (Ergen & Pilanci, 2019).When the number of sample (n) is larger than d, during training process, both objective value of original function and relaxed function will achieve to 0, which

confirms the property that convex relaxation doesn't change the optimal value. When n is smaller than or equal to d, it indicates the regularized gradient descent converges faster than gradient descent algorithm. It possibly confirms the reason why gradient descent may fail even in shallow network with non-convex objective function. However, this convex relaxation only applies to shallow network, which is quite limited. Further research can be done to test whether we can achieve such convex relaxation on Deep ReLU network.

2.4. Revealing the Structure of Deep Neural Networks via Convex Duality

This paper mainly uses the existing convexity of dual problem for objective function and uses the extreme point condition in constraint to compute the explicit solution of each layer weight matrix of neural networks. As a result, the strong duality holds for regularized training problem and its weight coincides with the iterative methods such as SGD. Recall the general optimization problem:

$$\begin{aligned} \min_{x} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \\ & h_i(x) = 0 \end{aligned} \tag{23}$$

The lagrangian function is defined as:

$$L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \tag{24}$$

And the dual problem is $D^* = \max \inf L(x, \lambda, \nu)$. No matter whether f(x) is convex or not, the $\inf L(x, \lambda, \nu)$ is a concave function. Thus, maximize a concave function is a convex optimization problem. (Boyd & Vandenberghe, 2004). This is why idea how to connect duality with convexity and apply them into neural network training problem.

2.4.1. TWO-LAYER LINEAR NETWORK

In order to give a more precise explanation for how to use duality to compute the weight matrix of neural network. The minimum norm variant problem is first considered as the following:

$$\min_{\theta \in \Theta} \|W_1\|_F^2 + \|w_2\|_2^2 \;\; s.t. \;\; f_{\theta,2}(X) = y \tag{25}$$

Where $W_1 \in R^{d \times m}$ is the weight matrix connecting input layer and hidden layer, $w_2 \in R^m$ is the weight vector connecting hidden layer and output layer. $X \in R^{n \times d}$ is the input matrix and $y \in R^n$ is the output respectively. By Lemma A.1 (Tolga Ergen, 2020), it is equivalent as:

$$P^* = \min_{\theta \in \Theta} \|w_2\|_1 \;\; s.t \;\; f_{\theta,2}(X) = y, \;\; w_{1,j} \in B_2, \forall j \tag{26}$$

Using duality scenario, the dual problem (with strong duality holds) is given by:

$$\begin{aligned} P^* &= D^* \\ &= \max_{\lambda} \lambda^T y \;\; s.t. \;\; \left\|(XW_1)^T \lambda\right\| \leq 1 \;\; \forall \|w_{1,j}\|_2^2 \leq 1 \forall j \\ &= \max_{\lambda \in R^n} \lambda^T y \;\; s.t. \;\; \max_{w_1 \in B_2} \left|\lambda^T X w_1\right| \leq 1 \end{aligned} \tag{27}$$

Here B_2 is the unit ball constraint with squared norm (i.e $\|u\|_2 \leq 1$.The optimal solution is obtained on the boundary of the constraint $\max_{w_1 \in B_2} \left|\lambda^{*T} X w_1\right| \leq 1$.
Assume that there exists a planted parameter w^* such that $Xw^* = y$ holds, then apply Singluar Value Decomposition(SVD) on original input matrix X, the problem can be reformulated as:

$$\begin{aligned} \max_{\widetilde{\lambda}} \quad & \widetilde{\lambda} \Sigma_x \widetilde{w}^* \\ \text{s.t.} \quad & \left\|\Sigma_x \widetilde{\lambda}\right\|_2 \leq 1 \end{aligned} \tag{28}$$

Where $\widetilde{\lambda} = U_x^T \lambda$ and $\widetilde{w^*} = V_x^T w^*$. When $\Sigma_x^T \widetilde{\lambda} = c_1 \widetilde{w}^*$, the optimal neuron (i.e, each column of W_1) should satisfy the following condition:

$$w_1^* = \frac{V_x \Sigma_x \widetilde{\lambda}}{\left\|V_x \Sigma_x \widetilde{\lambda}\right\|_2} = \frac{V_x \widetilde{w}_r^*}{\|\widetilde{w}_r^*\|_2} = \frac{P_{X^T}(w^*)}{\|P_{X^T}(w^*)\|_2} \tag{29}$$

The above scenario can be summarized as: first, derive the dual problem, which has strong duality property and it is also a standard convex optimization problem; second, the optimal solution should be the extreme points of the constraint.
Following this scenario, the primal and dual regularized training problem for two-layer linear network are:

$$\begin{aligned} P^* &= \min_{\theta \in \Theta} \|f_{\theta,2}(X) - y\|_2^2 + \beta \|w_2\|_2^2 \;\; s.t. \;\; w_{1,j} \in B_2 \\ D^* &= -\frac{1}{2} \|\lambda - y\|_2^2 + \frac{1}{2} \|y\|_2^2 \;\; s.t. \;\; \max_{w_1 \in B_2} \left|\lambda^T X w_1\right| \leq \beta \end{aligned} \tag{30}$$

Strong duality also holds in this case(Theorem 2.2 (Tolga Ergen, 2020). Solving above dual objective function by optimal condition ($\lambda = y$) and plug it into the constraint to satisfy the extreme point condition, the explicit for w_1 is:

$$w_1^* = \frac{X^T P_{X,\beta}(y)}{\|X^T P_{X,\beta}(y)\|_2^2} \tag{31}$$

Where $P_{X,\beta}$ projects to $\left\{u \in R^n \middle| \left\|X^T u\right\|_2 \leq \beta\right\}$

2.4.2. TWO-LAYER LINEAR NETWORK WITH VECTOR OUTPUTS

Unlike the weight vector w_2 for previous problem, in this case, $W_2 \in R^{m \times n}$ will become a matrix satisfying

$f_{\theta,2}(X) = XW_1W_2, Y \in R^{n\times K}$, the primal and dual of the minimum norm variant problem is:

$$\begin{aligned} P^* &= \min_{\theta\in Theta} \|W_1\|_F^2 + \|W_2\|_F^2 \;\; s.t. \;\; f_{\theta,2}(X) = Y \\ &= \min_{\theta\in Theta} \sum_{j=1}^{m} \|w_{2,j}\|_2 \;\; s.t. \;\; f_{\theta,2}(X) = Y, \;\; w_{1,j} \in B_2 \\ D^* &= \max_{\Lambda} tr(\Lambda Y) \;\; s.t \;\; \left\|\Lambda^T X w_1\right\|_2 \le 1, \forall w_1 \in B_2 \end{aligned} \tag{32}$$

Strong duality holds for above equation(Theorem 2.3(Tolga Ergen, 2020)).Suppose there exists a planted parameter W^* such that $Y = XW^*$,The solution for optimal neuron is still the extreme points of constraint$\left\|\Lambda^T X w_1\right\|_2 \le 1, \forall w_1 \in B_2$, which is the subset of first $rank(W^*)$ largest right singular vectors of $\Lambda^T X$.

Similarly, for the regularized training problem with vector output, the primal and dual problem is:

$$\begin{aligned} P^* &= \min_{\theta\in\Theta} \frac{1}{2}\|f_{\theta,2}(X) - Y\|_F^2 + \beta\sum_{j=1}^{m}\|w_{2,j}\|_2 \;\; s.tw_{1,j} \in B_2 \\ D^* &= \max_{\Lambda} -\frac{1}{2}\|\Lambda - Y\|_F^2 + \frac{1}{2}\|Y\|_F^2 \;\; s.t\sigma_{max}(\Lambda^T X) \le \beta \end{aligned} \tag{33}$$

Strong duality holds for above problems(Theorem 2.4 (Tolga Ergen, 2020)).The optimal solutions for dual problem is the subset of maximal right singular vectors of $P_{X,\beta}(Y^T)X$, where $P_{X,\beta}(.)$ projects vector into the set $\left\{U \in R^{n\times K} | \sigma_{max}(U^T X) \le \beta\right\}$.

2.4.3. Deep Linear Networks

Unlike the usual expression for deep linear networks (i.e $f_{\theta,L} = XW_1W_2...\mathbf{w_{L-1}}w_L$, the layer weight matrix is setting as: $f_{\theta,L} = \sum_{j=1}^{m} XW_{1,j}...w_{L,j}$,then for the training problem:

$$\min_{\{\theta_l\}_1^L} \frac{1}{2}\sum_{j=1}^{m}\sum_{l=1}^{L}\|W_{l,j}\|_F^2 \;\; s.t. \;\; f_{\theta,L} = y \tag{34}$$

Theorem 1 *Optimal layer weights for (33) is:*

$$W_{l,j}^* = \begin{cases} t_j^* \frac{V_x \widetilde{w}^*}{\|\widetilde{w}_r^*\|_2}\rho_{1,j}^T, & l = 1 \\ t_j^* \rho_{l-1,j}\rho_{l,j}^T, & 1 < l \le L-2 \\ \rho_{L-2,j}, & l = L-1 \end{cases}$$

Where $\rho_{l,j} \in R^{m_l}$ such that $\|\rho_{l,j}\|_2 = 1 \;\; \langle \rho_{l,j}, \rho_{l,k}\rangle = 0 \;\; \forall l \in \{1....L-2\} \;\; \forall j,k \in \{1....m\}$. w_r^* preserve the first r element and the rest equals to 0, where r is the rank of X.Besides, the strong duality also holds for dual of (34).

However, if plugging first layer weight, we can find:$XW_1\rho 1 = cy$ holds. Thus, the rest layer actually contributes nothing to the expressive power of the network.

Similarly, for regularized training problem:

$$\min_{\{\theta\}_1^L} \frac{1}{2}\|f_{\theta,L}(X) - y\|_2^2 + \frac{\beta}{2}\sum_{j=1}^{m}\sum_{l=1}^{L}\|W_{l,j}\|_F^2 \tag{35}$$

Theorem 2 *Optimal layer weights for (34) is:*

$$W_{l,j}^* = \begin{cases} t_j^* \frac{X^T P_{X,\beta}(y)}{\|X^T P_{X,\beta}(y)\|_2}\rho_{1,j}^T, & l = 1 \\ t_j^* \rho_{l-1,j}\rho_{l,j}^T, & 1 < l \le L-2 \\ \rho_{L-2,j}, & l = L-1 \end{cases}$$

Where $P_{X,\beta}$ projects to the set $\left\{u \in R^n | \left\|X^T u\right\|_2 \le \beta t_j^{*2-L}\right\}$. Strong duality also holds for (35) (Tolga Ergen, 2020).

2.4.4. Deep Linear network with vector output

In this case, the L-layer deep linear network model should be described as:$f_{\theta,L}(X) = \sum_{j=1}^{m} XW_{1,j}....\mathbf{w_{L,j}^T}$. Strong duality also holds in this case and the optimal neuron can be computed explicitly similarly as deep linear network with scalar output. The key difference in expression is the first layer weight matrix. For regularized training problem, instead using $\frac{X^T P_{X,\beta}(y)}{\|X^T P_{X,\beta}(y)\|_2}$, we need to compute the maximal right singular vector of $P_{X,\beta}(Y)^T X$, which is extremely similar as the analysis for two-layer linear network with vector output.

2.4.5. Deep ReLU Networks

Here, we consider a L-layer ReLU network with the output function $f_{\theta,L}(X) = A_{L-1}w_L$, where $A_{l,j} = (A_{l-1,j}W_{l,j})_+, A_{0,j} = X, \;\; \forall l,j$ and $(x)_+ = max(0,x)$ (definition of ReLU). The minimum norm variant problem can be formulated as:

$$\min_{\{\theta_l\}_1^L} \sum_{j=1}^{m}\sum_{l=1}^{L}\|W_{l,j}\|_F^2 \;\; s.t \;\; f_{\theta,L}(X) = y \tag{36}$$

Theorem 3 *Let X be rank-one matrix such that $X = \mathbf{ca_0^T}$, where $\mathbf{c} \in R_+^n$ and $a_0 \in R^d$, then strong duality holds and optimal weights are:*

$W_{l,j} = \frac{\phi_{l-1,j}}{\|\phi_{l-1,j}\|}\phi_{l,j}^T, \;\; \forall l \in [L-2], \;\; w_{L-1,j} = \frac{\phi_{L-2,j}}{\|\phi_{L-2,j}\|_2}$ *where $\phi_{0,j} = a_0$ and $\{\phi_{l,j}\}_{l=1}^{L-2}$ a set of vectors such that $\phi_{l,j} \in R_+^{m_l}$ and $\|\phi_{l,j}\|_2 = t_j^*$*

2.4.6. Regularized problem with vector output

Now, the last layer has multiple neurons thus the shape of output matrix $Y \in R^{n\times K}$. if we focus on the minimum norm variant problem, the results stated in last theorem still holds for vector output case. However, from above results,

when the model has L layers, we can only compute L-1 layers explicitly.
In order to compute the full layer weights explicitly, additional assumptions are needed:

Theorem 4 *Let $\{X, Y\}$ be a dataset such that $XX^T = I_n$ and Y is one-hot encoded, then a set of optimal solutions for the following regularization training problem:*

$$\min_{\theta\in\Theta} \frac{1}{2} \|f_{\theta,L}(X) - Y\|_F^2 + \frac{\beta}{2} \sum_{j=1}^{m} \sum_{l=1}^{L} \|W_{l,j}\|_F^2 \quad (37)$$

can be formulated as:

$$W_{l,j} = \begin{cases} \frac{\phi_{l-1,j}}{\|\phi_{l-1,j}\|_2} \phi_{l,j}^T, & if \ \ l \in [L-1] \\ \left(\|\phi_{0,j}\|_2 - \beta\right)_+ \phi_{l-1,j} e_r^T, & if \ \ l = L \end{cases}$$

where $\phi_{0,j} = X^T y_j$, $\{\phi_{l,j}\}_{l=1}^{L-2}$ is set of vectors such that $\phi_{l,j} \in R_+^{m_l}$, $\left\|\phi_{l,j} = t_j^*\right\|$ and $\phi_{l,i}^T \phi_{l,j} = 0, \ \ \forall i \neq j$, Moreover, $\phi_{L-1,j} = e_j$ is the j^{th} ordinary basis vector.

Strong duality also holds for optimization problem (37).

2.4.7. Numerical Experiments and Conclusion

During the mathematical deduction, there are some interesting properties which are not covered in previous part. The first is for deep linear network, the rank of first (L-1) layer weight matrix is dependent on β. The second is for deep linear network, the Frobenius norm of first (L-2) layers is the same.At first, the author uses the data generated from standard normal distribution to verify above claims and also test the strong duality property. Then, the author use real dataset CIFAR-10 (Krizhevsky et al.) and MINIST to test the computed parameter with layer L=3,4,5. From Figure 4(Tolga Ergen, 2020), as the epoch increases, the training and test error computed by SGD will finally converge to the result computed by convex duality, which proves the result in theory part. However, this paper has strict assumptions for input data and output data when the structure comes into deep ReLU case, which is not practical in reality. It still needs to be expanded into more general assumptions.

2.5. Global Optimality Beyond Two layers: Training Deep ReLU Networks via Convex Programs

In this paper, the idea for "Convex Program" is still from the convexity in duality. However, unlike deep linear network or deep ReLU network considered in the last paper. In this paper, the neural network structure consists of K L-layer sub-networks. In particular, 3-layer structure is used in sub-network structure for deduction(Figure2).

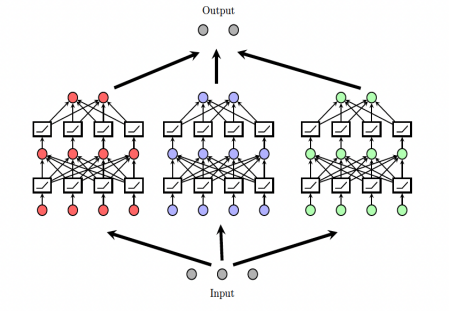


Figure 2. Architecture of sub-networks

In this article, the author uses $X \in R^{n\times d}$ to denote the input matrix. $\theta = \left\{\{W_{lk}\}_{l=1}^{L}\right\}_{k=1}^{K}$, where l represents the l^{th} layer in sub-network, k represents the k^{th} sub-network respectively. W_{lk} is the weight matrix of layer l in k^{th} sub-network.
A natural question arises why this type of structure is important. Consider the structure of Residual block (Figure 2 in (He et al., 2015)), its structure can be characterized in sub-network form:$W_{11} = W_1$, $W_{21} = W_2$, $W_{12} = W_{22} = I$, $W_{31} = W_{32} = w_3$. Thus, once the results of training deep ReLU networks can be achieved by convex optimization, it has important applications in many neural network structures.
Given dataset $\{\mathbf{X}, \mathbf{y}\}$ with $L = 3$, the regularized training problem can be formulated as:

$$P^* = \min_{\theta\in\Theta} \frac{1}{2} \|f_\theta(X) - y\|_2^2 + \frac{\beta}{2} \sum_{k=1}^{K} (\|w_{2k}\|_2^2 + w_{3k}^2) \quad (38)$$

Where $\Theta = \{\theta : \|W_{1k}\|_F \leq 1 \ \ \forall k \in [K]\}$ and $f_\theta(X) = \sum_{k=1}^{K} f_{\theta,k}(X)$. The above problem is equivalent as:

$$P^* = \min_{\theta\in\Theta_p} \frac{1}{2} \|f_\theta(X) - y\|_2^2 + \beta \|w_3\|_1 \quad (39)$$

Where $\theta_p = \{\theta : \|W_{1k}\|_F \leq 1, \|W_{2k}\|_F \leq 1, \forall k \in [K]\}$. Take the dual with respect to w_3 and change the order of min-max to obtain the dual problem:

$$\begin{aligned} P^* \geq D^* &= \max_v -\frac{1}{2} \|v - y\|_2^2 + \frac{1}{2} \|y\|_2^2 \\ s.t \ \ &\max_{\theta\in\Theta_p} \left|v^T ((XW_1)_+ w2)_+\right| \leq \beta \end{aligned} \quad (40)$$

Where $(.)_+$ is the representation of ReLU activation. Then, we also derive the bidual of (39), that is, taking dual form with respect to v in (40), the bidual can be formulated as:

$$\begin{aligned} P_B^* &= \min_\mu \frac{1}{2} \left\| \int_{\theta\in\Theta_p} ((XW_1)_+ w_2)_+ \right\|_2^2 + \beta \|\mu\|_{TV} \\ &= \min_{\theta\in\Theta_p} \frac{1}{2} \left\| \sum_{i=1}^{K^*} f_{\theta,i}(X) - y \right\|_2^2 + \beta \|w_3\|_1 . \end{aligned} \quad (41)$$

The above equivalence can be shown by Caratheodory's theorem. And (Ergen & Pilanci, 2021) shows that strong duality holds for $P_B^* = D^*$. Because (41) is the same as (39) with $K^* \geq K$. Thus, strong duality also holds $P_B^* = P^* = D^*$. However, due to the existing of ReLU function in constraint of D^*, this problem cannot be solved by standard convex optimization solver directly. It needs to be represented into a standard convex constraint. Following the results in the paper(Ergen & Pilanci, 2021), the constraint $\max_{\theta \in \Theta_p} \left| v^T ((XW_1)_+ w_2)_+ \right| \leq \beta$ is equivalent as :

$$\max_{\substack{i\in[P_1]\\ l\in[P_2]}} \max_{\substack{t_j\geq 0\\ \mathbf{1}^T\mathbf{t}\leq 1\\ \mathcal{I}_j\in\{\pm 1\}}} \max_{\substack{\|\mathbf{w}'_{1j}\|_2^2/w'_{2j}\leq t_j\\ \mathbf{1}^T\mathbf{w}'_2\leq 1\\ \mathbf{w}'_2\geq 0}} \mathbf{v}^T \mathbf{D}_{2l} \sum_{j=1}^{m_1} \mathcal{I}_j \mathbf{D}_{1ij} \mathbf{X} \mathbf{w}'_{1j}$$
$$\text{s.t. } (2\mathbf{D}_{1ij} - \mathbf{I}_n)\mathbf{X}\mathbf{w}'_{1j} \geq 0,\ \forall i, j,$$
$$(2\mathbf{D}_{2l} - \mathbf{I}_n) \sum_{j=1}^{m_1} \mathcal{I}_j \mathbf{D}_{1ij} \mathbf{X} \mathbf{w}'_{1j} \geq 0,\ \forall i, l,$$

Where $D_{1ij} \in R^{n\times n}$ and $D_{2l} \in R^{n\times n}$ are the sequence of diagonal mask matrix, which aims to simulate the behavior of ReLU. If the value after ReLU activation is still itself, then the corresponding diagonal value of D_{1ij} or D_{2l} is 1, otherwise it is 0.(see (Ergen & Pilanci, 2021) for details, the deduction for notation is omitted here.) $w'_{1j} = \sqrt{w'_{2j}} w_{1j}$ and $w'_{2j} = w^2_{2j}$, they are all the j^{th} column of W_1 and W_2 respectively. Use this constraint, the derived bidual form P_B^* can be formulated as:

Theorem 5 *The non-convex training problem (38) can be equivalent stated as a convex problem as follows:*

$$\min_{w,w'\in C} \frac{1}{2} \left\| \widetilde{X}(w' - w) - y \right\|_2^2 + \beta(\|w\|_{2,1} + \|w'\|_{2,1}) \tag{42}$$

where $\|.\|$ is a dimensional group norm operator such that given a vector $u \in R^{dp}$, $\|u\|_{2,1} = \sum_{i=1}^{P} \|u_i\|_2$, where u_i's are ordered d dimensional partitions of u. Moreover, $\tilde{X} \in R^{n\times 2dm_1P_1P_2}$ and C are defined as:

$$\mathcal{C} := \Big\{ \mathbf{w} \in \mathbb{R}^{2dm_1P_1P_2} :$$
$$(2\mathbf{D}_{1ij} - \mathbf{I}_n)\mathbf{X}\mathbf{w}^+_{ijl} \geq 0,\ (2\mathbf{D}_{1ij} - \mathbf{I}_n)\mathbf{X}\mathbf{w}^-_{ijl} \leq 0,$$
$$(2\mathbf{D}_{2l} - \mathbf{I}_n) \sum_{j=1}^{m_1} \mathbf{D}_{1ij}\mathbf{X}\mathbf{w}^{\pm}_{ijl} \geq 0, \forall i, j, l, \pm \Big\}$$
$$\tilde{\mathbf{X}} := \begin{bmatrix} \tilde{\mathbf{X}}_s & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{X}}_s \end{bmatrix},$$

where $w, w' \in R^{2dm_1P_1P_2}$ are the vectors constructed by concatenating $\left\{ \left\{ \left\{ \left\{ w^{\pm}_{ijl} \right\}_{i=1}^{P_1} \right\}_{j=1}^{m_1} \right\}_{l2=1}^{P_2} \right\}_{\pm}$ *and* $\left\{ \left\{ \left\{ \left\{ w'_{ijl}{}^{\pm} \right\}_{i=1}^{P_1} \right\}_{j=1}^{m_1} \right\}_{l2=1}^{P_2} \right\}_{\pm}$ *respectively, and* $\tilde{X}_s = [D_{21}D_{111}X ... D_{2l}D_{1ij}X ... D_{2P_2}D_{1P_1m_1}X]$

In order to solve the problem stated in Theorem 5. We need first to simulate the series of $\left\{ \{D_{1ij}\}_{i=1}^{P_1} \right\}_{j=1}^{m_1}$ and $\{D_{2l}\}_{l=1}^{P_2}$, where P_1 is the number of positive sign after $(Xw), w \in R^d (w$ is the connecting layer between input data and first layer of sub-networks), P_2 is the number of positive sign after $((XW_1)_+ w_2)$.They are all in polynomial time complexity with respect to input dimension n and d (Ergen & Pilanci, 2021).

To achieve simulation, generate $\mathbf{u_{ij}}$ from $N(0, \mathbf{I_d})$ for P_1 times and let $D_{ijl} = diag(\mathbf{1}[\mathbf{X}\mathbf{u_{ij}} \geq 0)$. Similarly, generate $\mathbf{U_1} \in R^{d\times m_1}$ and $\mathbf{u_2} \in R^{m_1P_2}$ times and set $D_{2l} = diag(\mathbf{1}[(\mathbf{X}\mathbf{U_{11}})_+ \mathbf{u_{2l}}] \geq 0)$, Then, use the standard solver CVX to solve this problem. According to the complexity of interior point methods, the paper (Ergen & Pilanci, 2021) also proves that this problem can be trained in polynomial time with respect to n and d,which indicates this problem is a practical problem for computing.

After introducing the formulation of the solution, from the series of diagonal matrix, it leads us to rethink about the role of ReLU.For this subnetwork, it has two ReLU-layer, which can be interpreted as a high-dimensional feature selection method due to convex group sparsity regularization (Ergen & Pilanci, 2021).Thus, this results reveals the impact of having additional layers and its implication on the expressive power of a network.

2.5.1. Numerical Experiment

The author first conducts experiment to verify the strong duality holds between (38) and (42). Input data matrix X is generated by standard Gaussian distribution with (n,d)=(5,2). The 3-layer sub-network has structure $m_1 = 3$ and $K = 2, 5, 15$.Then, the author uses SGD to train (38) and use convex program to train (42). It is found that when K is small, the SGD may be easily get stuck in local minimum. However, with K increases, this phenomena will disappear, SGD will converge to the optimal value, which is same as the optimal value obtained by convex program. The result is shown in Figure 4 (Ergen & Pilanci, 2021)

At last, the author conduct experiment on CIFAR-10(Krizhevsky et al.) and Fashion-MNIST dataset (Xiao et al., 2017). The sub-network structure is $L = 3$, $m_1 = 100$,$K = 40$. The result is shown in Figure 5 (Ergen & Pi-

lanci, 2021). It is shown that the accuracy obtained by convex program is higher than SGD training. For details,please see Figure 5 in (Ergen & Pilanci, 2021).

3. Conclusion

The goal of this literature is to discover different existing techniques of convex optimization of neural network. Optimization of infinitely wide neural networks is asymptotically convex. The first two papers apply L-1 regularization to make this infinite variable problem finite. Convex neural network solves an exact or approximate linear classifier at each step. Global convergence bound can be achieved for L1 regularized neural network using Frank Wolfe algorithm.The third paper utilize one global minimum property of convex function, which eliminate the possible failure of gradient descent in non-convex setting. The fourth and fifth paper utilize the convexity in dual problem and transform the original non-convex problem into a convex problem, which is practical in time complexity and also applicable because of its better performance. There are also many interesting convex problems in Neural Network training that don't contain in this literature review. For example, one can use convex optimization to compute the sparse representation of original weight matrix, which will improve the efficiency of training a neural network, using dual convexity to describe the mechanism of back-propagation. Further combination of convex optimization and neural network may be focused more on generalizing these results into more complicated neural network structures with less restrictive assumptions. Hopefully, with the assistance of convex optimization, training neural network will not be a "black-box" problem soon.

References

Bengio, Y., Le Roux, N., Vincent, P., Delalleau, O., and Marcotte, P. Convex neural networks. *Advances in neural information processing systems*, 18:123, 2006.

Boyd, S. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

d'Aspremont, A. and Pilanci, M. Global convergence of frank wolfe on one hidden layer networks. *arXiv preprint arXiv:2002.02208*, 2020.

Ergen, T. and Pilanci, M. Convex optimization for shallow neural networks. pp. 79–83, 2019.

Ergen, T. and Pilanci, M. Global optimality beyond two layers: Training deep relu networks via convex programs, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Hettich, R. and Kortanek, K. O. Semi-infinite programming: theory, methods, and applications. *SIAM review*, 35(3): 380–429, 1993.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar(canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

Tolga Ergen, M. P. Revealing the structure of deep neural networks via convex duality. *arXiv preprint arXiv:2002.09773*, 2020.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.