CSE592 Convex Optimization
Pre-conditioning strategy for neural networks

Boyu Wang, Yang Wang

Abstract

In this project, we investigate several preconditioning strategies for a simple family of neural networks, namely, ridge regression. The first strategy is to leverage all the training data to compute a preconditioning transformation P. This leads to much smaller conditional number and better optimization result. The second strategy is to compute P using a mini-batch approach. The third strategy is to learn P via a regularization loss. The latter two approaches are promising as they can be easily incorporated into the current neural network design and optimization framework. Code is available at:`https://github.com/Boyu-Wang/preconditioning`.

1 Introduction

Neural networks tend to be ill-conditioned. In this project, we investigate several preconditioning strategies for neural networks. The first strategy is to leverage all the training data to compute a preconditioning transformation P on the input data. The second strategy is to compute P using a mini-batch approach. The third strategy is to learn P via a regularization loss. The latter two approaches are more promising as they can be easily incorporated into the current neural network design and optimization framework. Detailed analysis and experimental results for all three strategies are presented in the following sections.

2 Preconditioning for Rigde Regression

2.1 Ridge Regression: a simple neural network

Our experiments are carried out upon a very simple family of neural networks, namely, ridge regression. The formulation of ridge regression is as follows.

$$\begin{aligned}\min_{w\in\mathbb{R}^d} L(w) &= \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(w^T x_i - y_i)^2 + \frac{\lambda}{2}\|w\|^2 \\ &= \frac{1}{2N}\|X^T w - Y\|^2 + \frac{\lambda}{2}\|w\|^2 \\ &= \frac{1}{2}w^T(\frac{1}{N}XX^T + \lambda I)w - \frac{1}{N}Y^T X^T w + \frac{1}{2}Y^T Y\end{aligned}$$

where $X \in \mathbb{R}^{d\times N}$ is the input data or feature, $Y \in \mathbb{R}^N$ is the target value or label, $w \in \mathbb{R}^d$ is the parameter to optimize.

Ridge regression is a quadratic optimization problem with closed form solution. However, in this project, we consider it as a form of neural network which has exactly one linear layer and can be optimized using batch or mini-batch gradient descent. The hyper-parameters for optimization includes the step size η and weight decay λ. In this way, the strategies investigated and developed for ridge regression as a neural network, hopefully, can also be extended to other families of neural networks.

2.2 Condition Number Analysis

The correlation matrix C for input data X is defined as:

$$\begin{aligned} C :&= \frac{1}{N}\sum_{i=1}^{n} x_i x_i^T \\ &= \sum_{i=1}^{n} \lambda_i \mu_i \mu_i^T = U\Lambda U^T \end{aligned}$$

where λ_i, μ_i are corresponding eigen values and eigen vectors of the correlation matrix, and they are sorted in the descending order of eigen values (λ_1 is the largest eigen value and λ_d is the smallest eigen value).

The average condition number for this problem is:

$$\begin{aligned} \hat{\kappa}(C+\lambda I) &= \frac{tr(C+\lambda I)}{\lambda_d(C+\lambda I)} \\ &= \sum_{i=1}^{d} \frac{\lambda_i + \lambda}{\lambda_d + \lambda} \geq d \end{aligned}$$

2.3 Proposed Preconditioning Transformation

The preconditioning approach is trying to find a transformation (matrix) P such that for the transformed input data $\tilde{X} = PX$, the average conditional number of the ridge regression problem becomes d or close to d. There are three immediate ways to do preconditioning.

- $P = U\Lambda^{-\frac{1}{2}}U^T$. In this case, the new correlation matrix is $\tilde{C} = P(XX^T)P^T = U\Lambda^{-\frac{1}{2}}U^T U\Lambda U^T U\Lambda^{-\frac{1}{2}}U^T = I_d$. The new condition number is $\hat{\kappa}(\tilde{C}+\lambda I_d) = d$
- $P = \Lambda^{-\frac{1}{2}}U^T$. In this case, the new correlation matrix is $\tilde{C} = P(XX^T)P^T = \Lambda^{-\frac{1}{2}}U^T U\Lambda U^T U\Lambda^{-\frac{1}{2}} = I_d$. The new condition number is $\hat{\kappa}(\tilde{C}+\lambda I_d) = d$
- $P_k = \Lambda_k^{-\frac{1}{2}}U_k^T$, where Λ_k is a diagonal matrix with the largest k eigen values and the columns of U_k are the corresponding k eigen vectors. In this case, the new correlation matrix is $\tilde{C} = P_k(XX^T)P_k^T = \Lambda_k^{-\frac{1}{2}}U_k^T U\Lambda U^T U_k\Lambda_k^{-\frac{1}{2}} = I_k$. The new condition number is $\hat{\kappa}(\tilde{C}+\lambda I_k) = k$

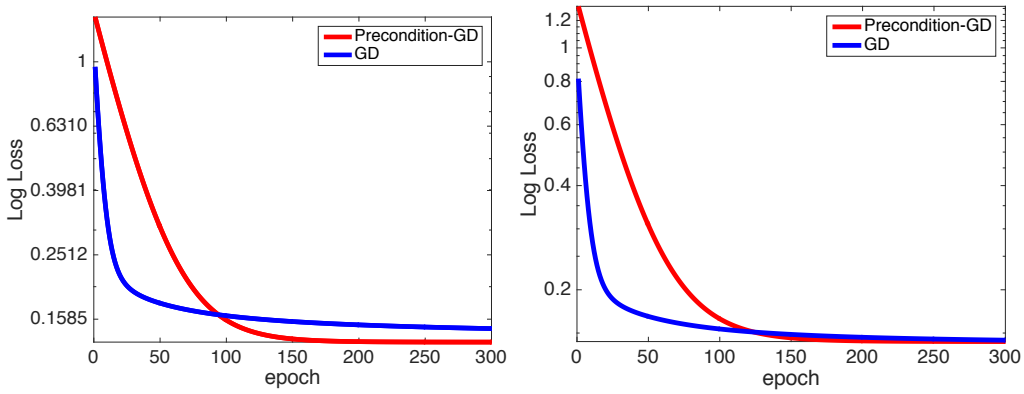
For this project, we choose the third method, where $\tilde{X} = \Lambda_k^{-\frac{1}{2}}U_k^T X$. There is a simple yet elegant interpretation for this precondition matrix, that is, U_k performs Principal Component Analysis on the input data, and Λ_k performs scale normalization on every principal direction.

3 Experiments

Following [1], we perform experiments on a subset of MNIST dataset, which consists of images of digit 4 and 7. Here, $N = 12107, d = 28 \times 28 = 784$. Our implementation is in PyTorch. As mentioned before, we perform experiments using three different preconditioning strategies: 1) Batch optimization; 2) mini-batch optimization; 3) learning precondition matrix via regularization. Details of these experiments are as follows.

3.1 Batch Optimization

In this experiment, the preconditioning matrix P is pre-computed only once using the entire training set. We compare the proposed strategy with the baseline approach where preconditioning are not applied. Optimization algorithms are implemented in Pytorch. We choose $k = 300, \lambda = 10^{-3}$. The learning rates of the two methods are both set to be 0.01. Figure 1 shows the convergence rate of using preconditioning and not using preconditioning on both training set and test set. The loss curve on the training set (which is used for optimization) clearly shows that using precondition helps improve the optimization performance.



(a) Training loss (b) Testing loss

Figure 1: **Convergence rate for batch optimization between using preconditioning and not using preconditioning** (a): Training loss (b) Testing loss. This curve clearly shows that using precondition helps improve the optimization performance.

3.2 Mini-batch Optimization

For online learning, the dataset might not be available before hand, so pre-computing the preconditioning matrix on the whole dataset is not possible. For neural networks, it is also the common practice to use mini-batch optimization for faster convergence. In this case, we propose another method which computes the preconditioning matrix P using a mini-batch approach. We have written in PyTorch a neural network layer called 'minibatch precondition', which keeps an running estimate of the correlation matrix of the input data, and use the corresponding top k eigen values and vectors to perform input preconditioning on each minibatch. The details of the algorithm is in Algorithm 1.

Algorithm 1: Minibatch Precondition

Initialize running correlation matrix C as Identity matrix, $\mu = 0.9$

Input : mini batch input: X_t

At training time, calculate the correlation matrix C_t of input mini-batch X_t. $C_t = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$

$C = (1-\mu)C_t + \mu C$

Perform eigen decomposition on C, and use the corresponding top-k eigen values and vectors to get precondition matrix $P = \Lambda_k^{-\frac{1}{2}} U_k^T$

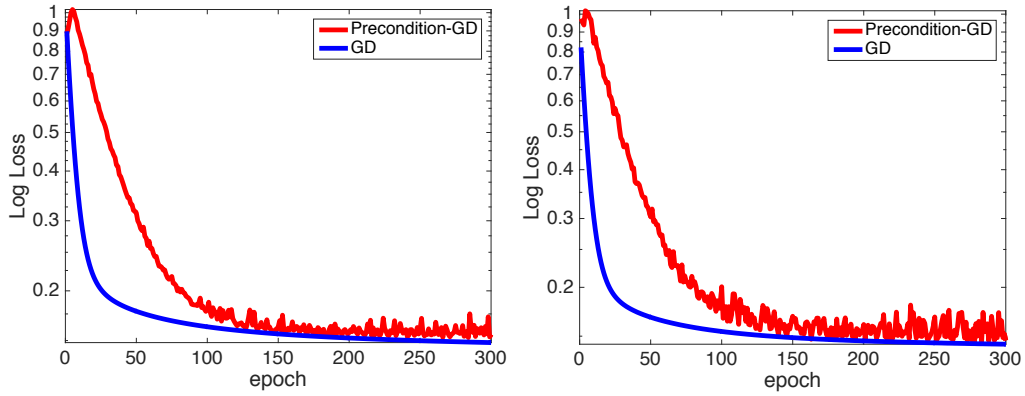
Compute $\tilde{X} = PX$

For this experiment, we use the same parameters as in the batch optimization case: $k = 300, \lambda = 10^{-3}, \eta = 0.01$. We use stochastic gradient descent as the optimizer. Figure 2 shows the convergence rate for minibatch optimization of using preconditioning and not using preconditioning on both training set and test set. As can be observed from Figure 2, due to the noisy estimation of the correlation matrix, it introduce noise to the optimization process.

3.3 Learning P via regularization

In this experiment, instead of computing the closed-form preconditioning matrix P, we propose to learn it via regularization. Our goal is to learn a transformation matrix $P \in R^{k \times d}$ such that the correlation matrix for the transformed input, $\tilde{C} = \frac{1}{N}\tilde{X}\tilde{X}^T = \frac{1}{N}PXXP^T$, is close to identity matrix I_k. With this analysis, we introduce an additional regularization loss in the learning process to regularize the the new correlation matrix to be close to identity matrix. Figure 3 demonstrates the process.

For this experiment, we set the weight of the new regularization loss to be 20. Other parameters are kept the same: $k = 300, \lambda = 10^{-3}, \eta = 0.01$. We use stochastic gradient descent as the optimizer. Figure 4 shows the convergence rate of using preconditioning and not using preconditioning on both



(a) Training loss (b) Testing loss

Figure 2: **Convergence rate for minibatch optimization between using preconditioning and not using preconditioning** (a): Training loss (b) Testing loss

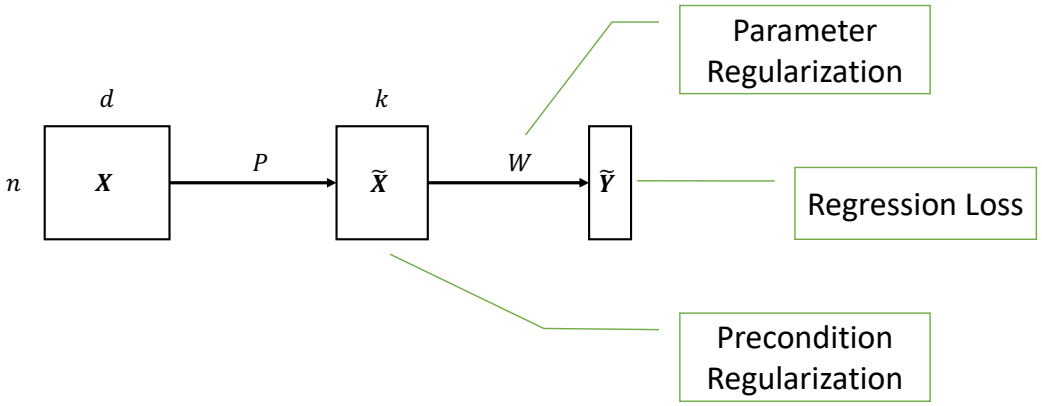


Figure 3: **Illustration of the third strategy: learning preconditioning matrix via regularization.** We introduce an additional regularization loss in the learning process to regularize the the new correlation matrix to be close to identity matrix .

training set and test set. So far, this strategy has not been able to achieve clear improvement on the optimization process.

4 Other works

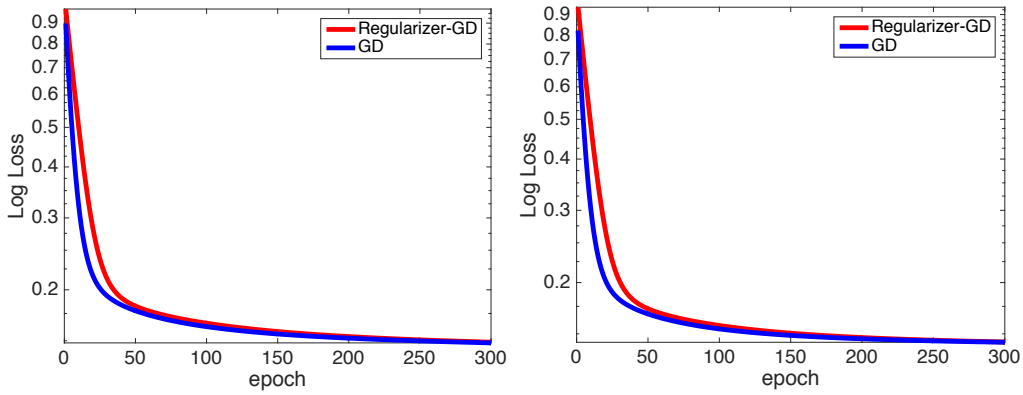
In addition to the three strategies mentioned in the previous sections, we also made an attempt to reimplement the proposed method in [1].

Again, the Hessian of ridge regression $L(w)$ is $H = \frac{1}{N}XX^T + \lambda I$. If we substitute each $\hat{x_i} = H^{-\frac{1}{2}}x_i$, then the loss becomes:

$$\min_{w \in \mathbb{R}^d} L(w) = \frac{1}{2}w^T(\lambda + 1)Iw - \frac{1}{N}w^T H^{-\frac{1}{2}}\hat{X}Y + \frac{1}{2}Y^T Y$$

The average condition number of new loss function become d.

However solving H is as hard as solving the original problem. [1] proposed sketched conditioning as a fast approximation of H.



(a) Training loss (b) Testing loss

Figure 4: **Convergence rate for learning preconditioning matrix via regularization.** (a): Training loss (b) Testing loss

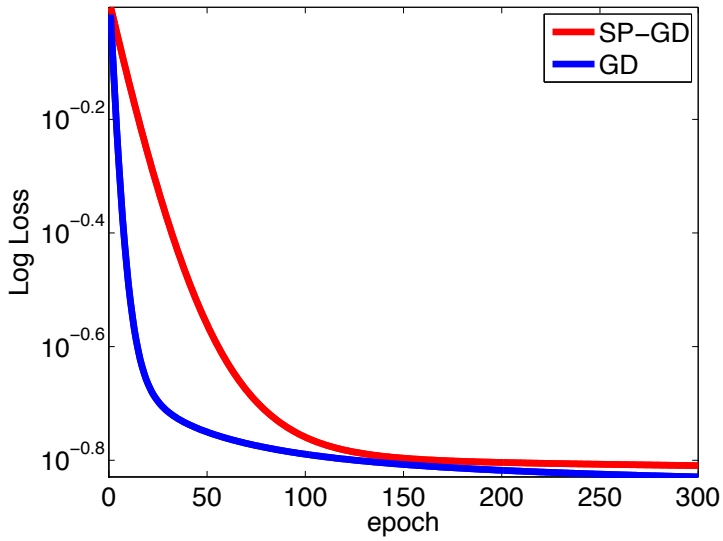


Figure 5: Convergence of Sketched Preconditioned Gradient Descent vs Gradient Descent

[1] used sketched preconditioning with SVRG algorithm. Since Stochastic Gradient Descent algorithm is popular for neural network, in this project, we combine sketched preconditioning with Gradient Descent in the first step. Our algorithm is described 2

Algorithm 2: Sketched Preconditioned Gradient Descent

Input : $X \in \mathbb{R}^{d \times N}, Y \in \mathbb{R}^N$

1 $\bar{X} = \frac{1}{\sqrt{N}} X$

2 Run SVD with $\bar{X}$ to obtain $\tilde{X}_k = \tilde{U}_k \tilde{\Sigma}_k \tilde{V}_k$

3 Use Equation (11) in [1] to get $P^{-\frac{1}{2}}$

4 Compute $\tilde{X} = P^{-\frac{1}{2}} X$

5 Solve $L(w)$ use Gradient Descent

We perform experiments on the same subset of MNIST dataset, which contains digits 4 and 7. We use $k = 30, \lambda = 10^{-3}$ for Sketched Preconditioning. We compare it with the baseline method which does not use preconditioning. Figure 5 shows the convergence rate of using preconditioning and not

using preconditioning on the training set. We did not make the Sketched Precondition method work properly. The reimplementation is not successful. The reason might be that we have used a different SVD solver from the original paper, which could lead to noisy matrix decomposition and undesirable result.

References

[1] Alon Gonen, Francesco Orabona, and Shai Shalev-Shwartz. Solving Ridge Regression using Sketched Preconditioned SVRG. *ICML 2016*