Constrained Optimization for Deep Learning

Wei Jiang, Xiaoqi Ma, Shuo Wen

***Abstract*—This project implements and analyzes two popular methods - projected gradient descent(PGD) and Frank-Wolfe(FWGD) algorithm in deep neural network. The effect of hyper-parameters, such as constraint shape, constraint size, network size, and step size are discussed and optimized.**

I. Introduction

In neural network, regularization is usually introduced to extract important features and produce sparsity. The most straightforward way is to provide a constraint to the parameters. The training objective becomes

$$\min_{w \in \chi} L(w) \quad , \tag{1}$$

where $L(w)$ is the loss function to parameters and χ is the constraint. Another trade-off version of Equ.1 is the *Lagrangian* version, which applies a penalty with regularization term in loss function. In this project, we focus on the Equ.1 and implement and optimize FWGD and PGD algorithms which constrain weights and bias in each layer.

The earliest Frank-Wolfe algorithm [1] analyzes for polyhedral constraint problem which uses line-search as step size. Recently there are several researches [2] [3] applied it for *matrix factorization* while rarely in deep learning. The only similar publication [4] for neural network we found is an experiment on SVM loss and $L1$ norm of FW with Cifar10 dataset.

Our project uses the *cross entropy* loss and discusses the Lp constraint problem. Also, since there is no closed-form computation of line search for step size in the deep learning, the modified and gap-based step size are applied. And the influence of constraint size $kappa$ ($\|x\|_p \leq kappa$) is also studied in this project.

II. Method

A. Projected gradient descent

For the projected gradient descent which combined with SGD, we project the parameters out of constraint onto euclidean projection ball ($L1$-ball and $L2$-ball in this project). Specifically, given an vector y which is out of constraint, we output a vector x by solving the following optimization problem:

$$x = \arg\min_{x \in s} \|x - y\|_2 \tag{2}$$

$$s = x \in R^n : \|x\|_1 \leq kappa \tag{3}$$

According to the Lagrangian and KKT system [5], we get the solution of this optimization problem as following:

$$x_i = sgn(y_i)(|y_i| - \lambda)_+ \tag{4}$$

where λ means the threshold of soft threshold function. As for the projection to the $L2$-ball [6], we get the solution as following:

$$x = kappa \cdot y \cdot min(1, 1/\|y\|_2^2) \tag{5}$$

B. Frank-Wolfe algorithm

By finding a s_t within the constraint and knowing that w_t is in the constraint, FW ensures w_{t+1}, which equals to $\gamma s_t + (1-\gamma)w_t$, also locates in the constraint. And we should note that w_{t+1} can be surely feasible only when the constraint is a convex set. In this project, we use $L1$-ball, $L2$-ball, and L_{inf}-ball as the constraint of FW.

This project achieves the algorithm in [2] which has complexity of linear time($O(n)$). The algorithm is as the following: Start with $w_0 \in \chi$. For $t = 0, 1, 2, ..T$:

1) Compute $s_t = argmin_{s \in \chi} \langle s, \nabla f(w_t) \rangle$
2) Set $w_{t+1} = (1-\gamma_t)w_t - \gamma_t s_t$
3) Update $\gamma_t = \frac{2}{t+2}$

Especially when coming to $L1$ constraint, we will have

$$s_t = -sign(\nabla f(w_t)_i \mathbf{e}_i, \quad i := \arg\max_i(\nabla f(w_t)_i) \tag{6}$$

Equ.6 means s_t only updates the parameters in one dimension which result in very slow convergence. In this project, in order to accelerate it and improve the performance, we let s_t updates ten parameters in each step by finding the top-10 maximum gradients in $\nabla f(w_t)$.

III. Experiment

A. Data

This project uses MNIST data set with one-channel images which have size 14*14. We generated 60000 images for training process and 10000 for validation from torchvision package for digit classification task.

B. Models

Using pytorch, four-layer MLP which 128 hidden units is built for exploring the constraint effect. Different activation functions are tried such as $relu$, $sigmoid$, $softmax$ and $tanh$. Among them, $relu$ does not work in FW algorithm because of gradient vanishing, while $sigmoid$ and $softmax$ converge slowly for the small gradient value. $tanh$ performs exceptionally well so we used it as activation function.

C. Frank-Wolfe

1) Step size

Step size is an important hyper-parameter in Frank Wolfe algorithm since the algorithm can be very slow to converge with an improper step size. The oblivious step size shown above does not perform well in our experiment, which requires improvement. We tried two different step sizes as follows.

- **Modification**: Updating model parameters by using extreme points on the constraint, FW needs a large step size at the first several steps but it should decrease sharply in the following steps. In the experiment, we found the first few step sizes are too large that is nearly 1 in the default step size which results in slow convergence. In practice, we found the step size that is smaller at the beginning and decreases more quickly works better, which is

$$\gamma_t = \frac{2}{t^{1.1} + 3} \tag{7}$$

- **Gap-based search**: According to the algorithm proposed in the paper [3], this adaptive step size method depends on the choice of (a lower bound on) the Lipschitz constant L. Inspired by the discovery of Lipschitz constant is the $L2$ norm of weights in deep neural network in [7], we use the $L2$ norm of weights and bias in each layer as the constant for calculating step size.

$$d_t = s_t - x_t \tag{8}$$
$$g_t = -\langle \nabla f(x_t), d_t \rangle \tag{9}$$
$$\gamma_t = \frac{g_t}{L \left\| d_t \right\|^2}, \quad L = \left\| W_t \right\|_2 \tag{10}$$

The result of different choice of step size applied in $L1$-ball constraint is showed in Fig.1. The modified and adaptive (gap-based) step size both outperform the default one, accelerating the loss convergence and achieving higher accuracy in shorter time. Noticeably, the modified step size significantly accelerates the convergence in the beginning while the adaptive step size improves more in accuracy. In terms of performance, adaptive step size is much slower than the others because of heavy computation of parameter norm.

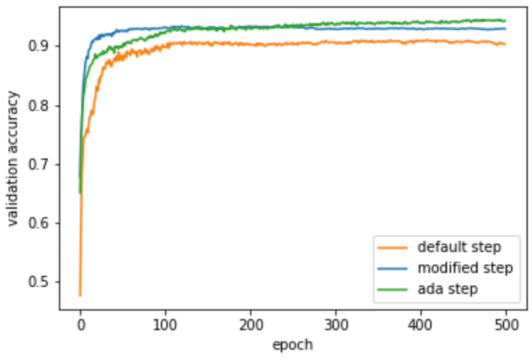


Fig. 1: Comparison of different step sizes

2) Constraint shape

We explore the performance of $L1$, $L2$, L_{inf} norm in Frank-Wolfe algorithm. In terms of $L1$ and L_{inf} which are polygons, the project realized the method in [2], while for $L2$ norm, we used the algorithms showed in [8]. By using the adaptive step size, the result of three regular norm is showed in Fig.2. We can see $L1$ norm performs the best since it extracts important features and removes outliers in the images. The choice of $kappa$ depends on the distribution of parameter Lp norm in SGD.

3) Constraint size

Kappa is also an important hyper-parameter in Frank Wolfe algorithm since s_t lies on the margin of the constraint. For example, in $L1$ regularzation, s_t locates on one of the corners of $L1$-ball. Here we modify $kappa$ and the result is shown in Fig.7 in *Appendix*. It shows that when $kappa$ is too small,

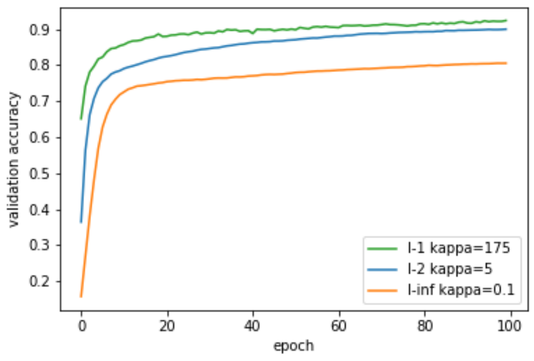
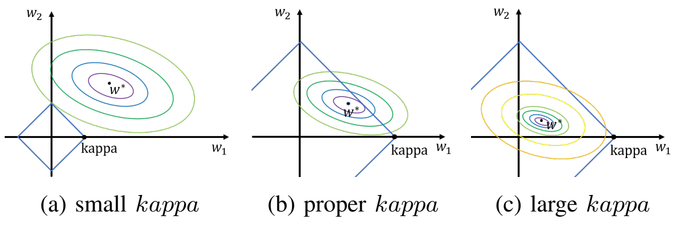


Fig. 2: The different l-p norm performance

model cannot achieve high accuracy, while when $kappa$ is too big, the model can be slower to converge to high accuracy.

The result can be explained by Fig.3: when $kappa$ is too small, the constraint does not contain the best weight w^* as shown in Fig.3a, so the model cannot achieve the best performance; when $kappa$ is too large, as in Fig.3c, since s_t lies on the corners of $L1$-ball and learning rate γ is nearly 1 in the first few steps and decreases sharply in the following steps, it will take long steps to converge to w^*; only when $kappa$ is proper, which means $L1$-ball contains w^* and w^* is not too far away the margin of $L1$-ball as in Fig.3b, the model can achieve the best performance quickly.



(a) small $kappa$ (b) proper $kappa$ (c) large $kappa$

Fig. 3: Three relations between $kappa$ and w^*

4) FW network

In $L1$ regularzation, Frank Wolfe algorithm only updates fixed small number of model parameters each step as described in Sec.II-B, and this may lead to a result that the algorithm cannot used on large size network. So we change the size of the network (by changing the number of hidden units) and evaluate the performances. The result is shown in Tab.I.

TABLE I: Network Comparsion

Hidden Units	$Kappa$	Accuracy	Epoch	Sec / Epoch	Sparsity
64	125	92.1%	150	1.23	76.5%
128	175	92.9%	300	1.52	88.9%
256	200	93.3%	230	1.90	95.3%
512	250	93.7%	375	4.49	98.4%
1024	300	93.7%	350	13.39	99.4%

We calculate the sparsity by dividing number of parameters which smaller than 10^{-6} by the total number of parameters. The reason we set 10^{-6} as the threshold is based on the distribution of the parameters. Less than 0.1% parameters distribute in the range of $(10^{-6}, 10^{-3})$, and this gap means parameters smaller than 10^{-6} are too small to play a role in the model compared to 10^{-3}.

Tab.I shows that Frank Wolfe still works on large networks since the sparsity can increase with the increasing size of network. In other words, the epochs needed for convergence is not proportional to the size of network since the parameters which need to be updated is not proportional to the size of network.

D. *Projected gradient descent*

1) *Learning rate*

Learning rate is important to both convergence rate and accuracy. It will be slow to converge when using small learning rate, while big learning rate prevents the model from converging to the best parameters. In this project, we find the most appropriate learning rate to be 0.01 for PSGD. The performance of different learning rate is shown in Fig.6 in *Appendix*.

2) *Constraint shape*

We use $L1$-ball and $L2$-ball as the constraint. We project the data into $L1$-ball with $kappa$ equals to 500 and $L2$-ball with $kappa$ equals to 5. Fig.4 shows that SGD with $L2$ ball converges faster than $L1$-ball and SGD. Projected SGD achieve higher validation accuracy than traditional SGD.

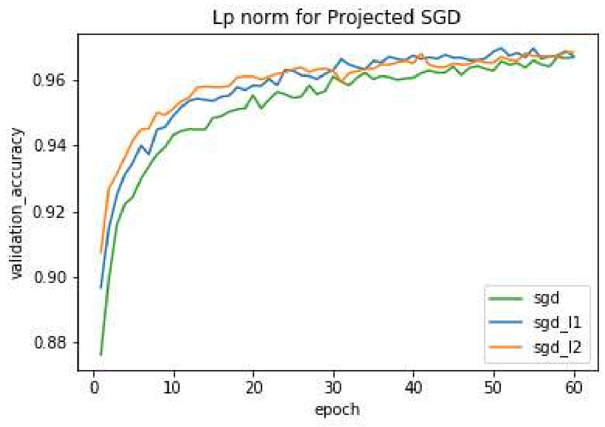


Fig. 4: Lp projected comparison

3) *Optimizers comparison*

Using $L1$ constraint, we compare different optimizers, which are SGD, Momentum SGD, AdaGrad, RMSprop and ADAM, to observe the influence of optimizers on the convergence rate. From Fig.5 we find that ADAM and Momentum SGD converge faster than SGD in the first 30 epoch, however, SGD achieves the highest accuracy in the end. There are two possible explanations: learning rate becomes too small for adaptive learning algorithms after few epochs; the introduction of momentum may cause the optimizer 'stop' when hitting the convex part of the loss function, and 'surmount' when hitting the concave part.

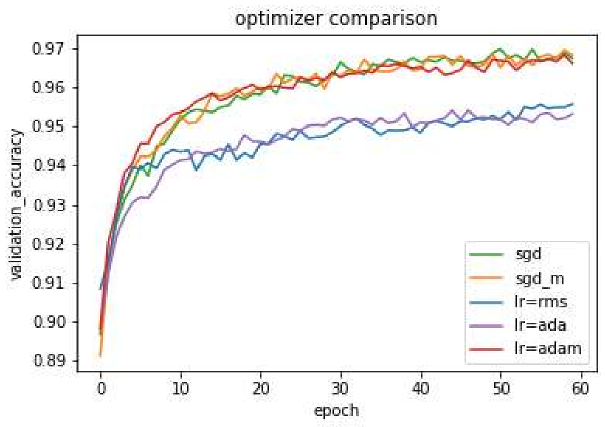


Fig. 5: Projected optimizer comparison

E. *Result comparison*

- **Sparsity**. Referring to the histogram of model parameters, i.e, weights and bias, we founf the main distribution range varies a lot in PSGD and FWGD. The sparsity is measured by the range of parameters that is recorded in II. We can see under the same $kappa$, the result of FW is much more sparse than PSGD. This is because FWGD only updates one or several parameters each step while PSGD updates almost all the parameters as SGD and then projects into $L1$-ball.

TABLE II: Parameter range of SGD, PSGD, FW

Method	90% range	80% range	$Kappa$	Epoch
SGD	(-1.0e-1, 1.0e-1)	(-7.6e-2, 7.6e-2)	/	100
PSGD	(-3.4e-2, 3.3e-2)	(-9.4e-3, 9.4e-3)	175	100
FW	(-5.2e-4, 6.0e-4)	(-1.5e-7, 1.6e-7)	175	100

- **Performance**. The performance of SGD, PSGD, FW are compared in table III. By looking at epoch time, we can see FWGD is much faster than PSGD because of $O(n)$ computation complexity. Theoretically it should also converge in linear time but in practice, FW can be slower to converge to high accuracy because it only updates limited number of parameters in each step.

TABLE III: Performance of SGD, PSGD, FW

Method	Maximum Accuracy	Epoch*	Epoch Time**	$Kappa$***
SGD	97.4%	221	0.65s	/
PSGD	96.9%	69	2.48s	175
FW	94.5%	467	1.29s	175

* the first epoch where maximum accuracy appeared
** average of 500 epochs for training run in CPU
*** use $L1$-ball and the same $kappa$ as constraint for PSGD and FW

IV. Conclusion

Comparing two constrained optimization algorithm, we find PSGD converges to a higher accuracy within fewer epochs, while FW costs less time each epoch and induce more sparsity. Also, FW can be very slow to converge to high accuracy especially when the hyper-parameters are not suitable.

References

[1] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. Naval Research Logistics Quarterly, 3(1-2):95–110, 1956.

[2] Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, Proceedings of the 30th International Conference on Machine Learning, volume 28 of Proceedings of Machine Learning Research, pages 427–435, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[3] Fabian Pedregosa, Geoffrey Negiar, Armin Askari, and Martin Jaggi. Linearly convergent frank-wolfe with backtracking line-search. 2020.

[4] Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Deep frank-wolfe for neural network optimization. International Conference on Learning Representations, 2019.

[5] Andersen Ang. Projection onto unit l1 ball. 2018.

[6] Andersen Ang. Projection onto unit l2 ball. 2018.

[7] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 3835–3844. Curran Associates, Inc., 2018.

[8] Ryan Tibshirani. Lecture notes in frank-wolfe method. Carnegie Mellon University, 2018.

Appendix

Appendix A. Different learning rate of PSGD

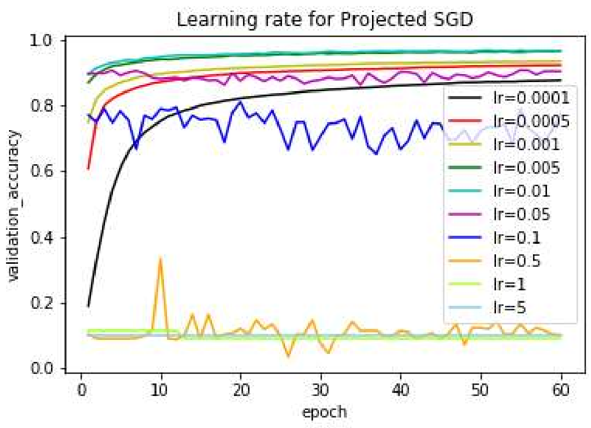


Fig. 6: learning rate of PSGD

Appendix B. Different Kappa of FW

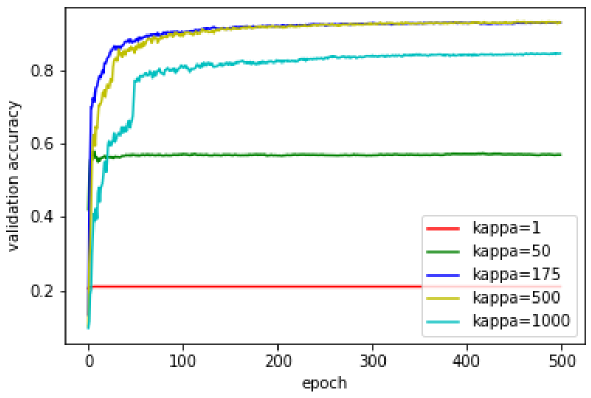


Fig. 7: Different Kappa of FW