EVALUATING CRITICAL PERIODS IN DEEP NEURAL NETWORKS

McKell Stauffer and Peizhu Qian
Rice University
mjs32@rice.edu, pqian@rice.edu

December 19, 2019

ABSTRACT

Critical periods refer to the limited time window during early development when infants and children are particularly sensitive to the effects of the environment and conditions. During critical periods, the brain of infants and children exhibits high level of neuroplasticity. As we get older, neuroplasticity becomes significantly less prominent. Recent research shows evidence that critical periods may be present in neural networks. In this paper, We find that critical periods may not exist for all neural network architectures and datasets. We also measure the plasticity of an AlexNet on CIFAR-10 and find that its plasticity correlates with the model's critical period.

1 Introduction

Critical periods are a biological phenomena where an organism must be exposed to a certain stimulus within an explicit time frame in order to develop a specific ability. For example, newborn mice must experience whisker sensations within the first few days of life, or they will develop abnormal tactile sensitivity [1]. In addition, cats must have normal visual inputs within the first 3 months of life, and monkeys need consistent social contact for the first six months of life in order to develop normal functions.

Related to critical periods is the concept of plasticity. In neurology, plasticity can be defined as the ability of the nervous system to respond to intrinsic or extrinsic stimuli by reorganizing its structure, function and connections [2]. It has been commonly believed that neuroplasticity peaks at a young age and then decreases over time, but recent studies have shown that lifelong neuroplasticity may be possible [3].

In this research, we investigate whether the concepts of critical periods and plasticity extend to the realm of artificial neural networks. Gaining greater insight into the initial training periods of neural networks is remarkably important as they continue to be applied in many areas including medical imaging, signal processing [4], finance [5], geological sciences [6], etc. In regards to lifelong plasticity, [7] and [8] called it one of the main long-standing challenges for current state-of-the-art networks. As we gain insight into how new information disseminates throughout a network over time, we will have an increased capacity to achieve lifelong learning in neural networks.

We intend to explore the following questions in the realm of image classification:

1. Do neural networks have critical training periods?
2. Does their plasticity change over time?
3. Do critical training periods correlated with the amount of plasticity in the network?

Our novel contributions are:

1. We scrutinize the results of [9] and show that critical training periods may not exist for all architectures.
2. We compare the plasticity of an AlexNet to its critical training period.

2 Related Work

To the best of our knowledge, only a few papers have touched on critical periods on neural networks [10][9][11][12]. [10] explores learning stages with regularization. In particular, they find that if regularization is not applied within the first few epochs, the network acts as if it were never regularized. In addition, if regularization is taken out after the first few epochs, the generalization accuracy increases. [9] shows that a temporary stimulus deficit (affecting low-level statistics) at the beginning of training can permanently impair a network's ability to

learn a task. They find that neural networks demonstrate a phenomena called “Information Plasticity". In particular, they notice that information rises rapidly in early stages of training. It then decreases, and obstructs the distribution of information. [11] believes that networks demonstrate an information theoretic bottleneck that splits learning into two phases, pattern detection and compression. These phases can also be thought of as memorization and generalization stages [13]. Finally, [12] shows that there exists small subnetworks, "winning tickets", of a larger neural network that can be found early in training and can be trained to the same accuracy as the larger network. They validate the claim that key connectivity patterns of neural networks emerge early. The main limitations of these works are the credibility of the results of [9] (as will be further discussed in Section 4) and that no work provides mechanisms for determining when different periods start and end.

3 Methods

To provide insights into the enumerated questions in Section 1, we conducted the following experiments respectively:

1. Blur images for the first n epochs of training on the MNIST and CIFAR-10 datasets using a convolutional neural network (CNN), fully-connected network (FC) and residual network (ResNet). After the first n epochs, the blurry images are removed and normal images are utilized for another 160 epochs of training. Then identify the effects of the initial blurring has on the final test accuracy.
2. Measure the plasticity of an AlexNet on a condensed CIFAR-10 by adding a new class at n epochs and observe the final testing accuracies.
3. Compare the critical period and plasticity results of the AlexNet.

All hyperparameters for the CNN, ResNet and FC models come from those detailed in [9]. The code for all models differ from that of [9] in the following ways:

1. [9]’s model contained dropout layers while ours did not as this was not mentioned in the paper (we compare the code).
2. Our model puts the output through an extra softmax as we did not realize that the cross entropy loss function in PyTorch already has softmax implemented into it.
3. We use a fixed learning rate and no weight decay for all experiments so that they would not be confounding variables. We use a learning rate of 0.005.
4. We used an average pooling size of 2 with no padding whereas [9] used an average pooling size of 8 and a padding of 1 (it was not specified in their paper).

Our critical period experiments try to exactly mimic those in [9] but due to computational limitations (we used Google Colab for our GPU resources), we do not train our models for as extensive of time periods as theirs. Particularly, we train a model on blurry images for $n \in \{0, 20, 40, 60, 80, 100, 120, 140\}$ epochs and then on normal images for another 160 epochs. We blur the images by downsampling them to an 8x8 image and then upsampling them through bilinear interpolation to their original size. We compare the average (of five runs) final testing accuracies for each experiment, whereas [9] presents only one experiment. We train using the cross entropy loss function.

Plasticity is the ability of a network to respond to new stimuli by reorganizing its structure, functions and connections [2]. Thus for our plasticity experiment, we study the ability of a neural network to learn new classes of training data, while not forgetting old ones. Again due to computational limitations, for our plasticity experiment we condense the CIFAR-10 training dataset down to 10,000 samples using random sampling. We train with the Adam optimizer on an AlexNet. In this experiment, we remove all truck images from the training dataset for the first $n \in \{0, 20, 40, 60, 80, 100\}$ epochs of training. We then continue training on the full dataset for the next 100 epochs of training. We compare the average (of three runs) final testing accuracies for each experiment. To compare the curves associated with critical periods and plasticity, we also run the above critical period experiment on the AlexNet architecture.

4 Results

We first explore our critical period experiment evaluated on the MNIST dataset. As shown in Figure 1, critical periods (as defined by the experiment) do not seem to exist for the MNIST dataset. This could be due to the intrinsic simplicity of MNIST. For comparison, we include the results of [9] in Figure 2. Due to the small time scale over a large amount of epochs, we claim that Figure 2 is misleading and that it does not show that critical periods exist on the MNIST dataset nor on a fully-connected network. [9] did not present results of a fully-connected network on the CIFAR-10 dataset but we include them for our models in Figure 3. This figure suggests that the initial blurry images work more as a pre-training than as a hindrance to the initialization of the network.

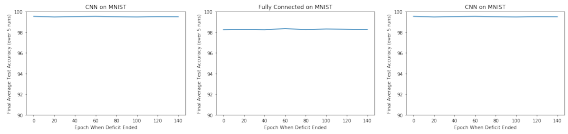


Figure 1: The critical period experiments on MNIST. The mean and variance (over 5 trials) of the final testing accuracies are plotted for various values of n. Each value of n (along the x-axis) represents the number of epochs that the blurry images were present for.

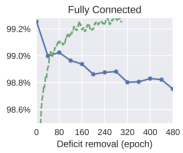


Figure 2: [9]'s fully connected results. Note that the difference on the y-scale is incredibly small. In addition, note this experiment contains many more epochs than ours.

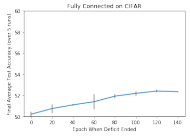


Figure 3: The critical period experiment of a fully-connected network on MNIST.

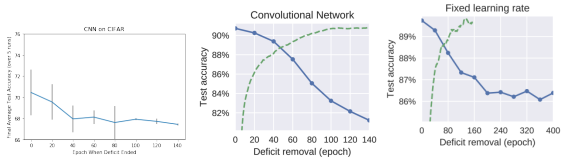


Figure 4: The results of the critical period experiments using a CNN on CIFAR-10. On the left is our results. The middle is [9]'s results with a learning rate schedule. And on the right is their results with a fixed learning rate.

In contrast, our experiments with the CNN (4) and ResNet (5) on the CIFAR-10 dataset demonstrate an existence of a critical period. After initially training the network with blurry images for the first 140 epochs, the final testing accuracies decrease 3-6%, wheras [9] report about a 13% decrease in accuracy. This large discrepancy could be due to the learning rate decay. In [9]'s fixed learning rate experiment with the CNN on CIFAR-10, the decrease is only about 3%, similar to our findings, except that they had to increase the initial training period out 240 more epochs (a total of 400 epochs) to get this result. For the CNN, our results report a 4% variance for some epochs. As [9] does not report variances, these large variances could make their results more suspicious.

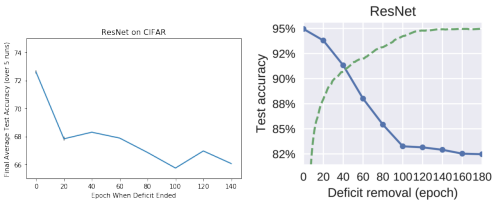


Figure 5: The results of the critical period experiments using a ResNet on CIFAR-10. On the left is our results and on the right is [9]'s results. Note that our results are not the average of 5 runs due to computational limitations.

Figure 6 contains the results of the same critical period experiments for an AlexNet. For this experiment, the average final testing accuracy did decrease about 7%. The plasticity results for the exact same implementation of an AlexNet can also be found in Figure 6. This experiment showed a similar 7-9% decrease in accuracy. Although the plasticity experiment showed a sharper initial decrease, both curves appear to be similar.

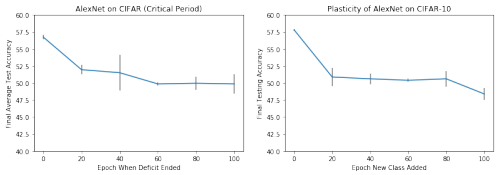


Figure 6: The left contains the critical periods experiment for an AlexNet. The right contains the plasticity experiment for the same AlexNet. On the left, the x-axis represents the n where the deficit was removed. On the right, the x-axis represents the n where the new class was added. Each result shows the average test accuracies across 3 trials.

5 Conclusion

Critical periods may not exist for all architectures and datasets, although there are evidences for their existence. The CNN, ResNet and AlexNet on the CIFAR-10 dataset showed a decrease in average final test accuracy when initially trained on blurry images. Whereas the Fully-Connected network showed an increase in average final test accuracy when trained on CIFAR-10. All architectures trained on MNIST showed no change in final test accuracies. We also found evidence that critical periods could be correlated with the amount of plasticity in a network at a certain time. This is important as plasticity could be a way to measure the start and end points of critical periods.

Future work would include exploring this relationship and finding other ways to determine the amount of information flow in a network. The ultimate goal would be to prove the existence of critical periods under certain conditions and to be able to determine when these critical period are. When critical periods of a network are understood, one could experiment with changing training schemes during or after these periods, which could lead to more robust and efficient training of models.

References

[1] Zohraida Sibtain Karim. *Let Me Be Me*. Dorrance Publishing, 2017.

[2] Steven C. Cramer, Mriganka Sur, Bruce H. Dobkin, Charles O'Brien, Terence D. Sanger, John Q. Trojanowski, Judith M. Rumsey, Ramona Hicks, Judy Cameron, Daofen Chen, and et al. Harnessing

neuroplasticity for clinical applications. *Brain*, 134(6):1591–1609, Jun 2011.

[3] Lisa Pauwels, Sima Chalavi, and Stephan P. Swinnen. Aging and brain plasticity. *Aging*, 10(8):1789–1790, Aug 2018.

[4] A. S. Miller, B. H. Blott, and T. K. Hames. Review of neural network applications in medical imaging and signal processing. *Medical Biological Engineering Computing*, 30(5):449–464, 1992.

[5] Bo K Wong and Yakup Selvi. Neural network applications in finance: A review and analysis of literature (1990–1996). *Information Management*, 34(3):129–139, 1998.

[6] Vladimir M. Krasnopolsky and Helmut Schiller. Some neural network applications in environmental sciences. part i: forward and inverse problems in geophysical remote measurements. *Neural Networks*, 16(3-4):321–334, 2003.

[7] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[8] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. *CoRR*, abs/1706.08947, 2017.

[9] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. *CoRR*, abs/1711.08856, 2017.

[10] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. *CoRR*, abs/1905.13277, 2019.

[11] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

[12] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Yingyan Lin, Zhangyang Wan, and Richard Baraniuk. Drawing early-bird tickets: Towards more efficient training of deep networks. *CoRR*, abs/1909.11957, 2019.

[13] Guan-Horng Liu and Evangelos Theodorou. Deep learning theory review: An optimal control and dynamical systems perspective. *CoRR*, abs/1908.10920, 2019.

COMP514 Final Project
Batch Size and Generalization Gap

Anonymous Authors[1]

Abstract

Deep learning models are often trained using random gradients. These methods use the gradients of mini-batches of the training data to update the model's parameters. One situation that has been observed is that when using large batches, generalization will continue to decline - known as the "generalization gap" phenomenon. Identifying the root cause and closing the gap are two problems we need to consider. In this report, we first make a review of the causes of large batch size generalization gaps. This paper lists and discusses the possible causes such as overfitting, saddle points, and sharpness. Later, we also studied some methods that can improve generalization, such as using ghost batch normalization and increasing the number of training iterations. Finally, using combination strategy of the ghosts batch normalization and sufficient updating iterations, we found that the performance of large batch sizes is not weaker than small batch sizes.

1. Introduction

Stochastic Gradient Descent (SGD) has been the workinghorse of many deep learning models which nowadays play an important role on a wide variety of applications like Artificial Intelligence, Computer Vision, and Natural Language Processing etc. SGD is an iterative algorithm that, within each loop, helps the network to update weight parameters in the direction such that the overall loss function would decrease the most. However, unlike the traditional Gradient Descent methods that utilize the whole gradient of loss function which is computation-costly, SGD only selects portions of training data (called the training batch) and compute a "partial gradient" of selected data portion to update the parameters without computing the whole gradient.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Due to the batch-divided training property of SGD method, one of the hyper-parameter to be wisely chosen is the batch size, and different batch sizes would affect significantly both the convergence and accuracy rate of the deep learning model. For this literature review, we would focus on how the deep model's accuracy is affected by batch sizes. In particular, it has been observed in practice that when using SGD to train the deep neural network, a large-batch training method would have poorer performance on test data than small-batch training method. Such degradation in model's generalizability is called "generalization gap".

In this literature review, we would summarize the state-of-art research results over the relationship between "generalization gap" and the training batch size of SGD optimizer (and its variants such as ADAM, AdaGrad) of the network. We would focus on the latest ideas and discoveries of two research directions: 1) Theories or hypothesis explaining the reason why large batch size causes the generalization gap. 2) Strategies improving the large-batch training method to reduce or eliminate the generalization gap.

2. Motivation

Even though that large-batch training would result in generalization gap and worse test outcomes, people are still looking for possibilities to apply large-batch training method to neural networks more widely due to its training advantages: Training the data in a larger batch size provides more parallelism to the learning pipeline. When using GPUs in training the neural networks, large-batch training would benefit more in parallel computing, and thus speeds up the training process much more significantly compared to small-batch training methods. (Goyal et al., 2017) have been utilized the speedup of large-batch training to train the ImageNet in less 1 hour. However, such drastic increase in training time did pay the price of worse testing accuracy when using larger batch size.

Therefore, to achieve both higher accuracy and faster training speed of the deep models, it becomes crucial to find out the reasons that could explain the generalization gap observed in large-batch training, and come up with methods that could eliminate the gap accordingly. How to achieve

such a jointly optimized network model is still somehow an open problem. We hope that, with this literature review, people could have a more organized picture of both the state-of-art theories and methods to overcome generalization gap. We also hope that some more successful method might be inspired in the future.

3. Theories Explaining the Generalization Gap

According to (Keskar et al., 2016), in general there are four speculations of the possible causes of generalization gap in large batch training: i) LB methods over-fit the data. ii) LB methods are attracted to saddle-points. iii) LB methods lack the explorative properties of SB methods and are prone to converging to minimizers closest to the initial point. iv) SB and LB methods converge to qualitatively different minimizers with differing generalization properties. Among them we would focus on the last conjectures in our literature review, since it has been considered as more reasonable causes of generalization gap with the support of numerical results.

3.1. Generalization Gap and Overfitting

Based on numerical results of extensive experiments over large-batch training of deep networks Keskar *et al.*, the first conjecture about LB methods over-fitting the data has been disproved. The phenomenon of overfitting suggests that with increasing training epochs, the model's test accuracy would first increase then decrease due to it over-learned some idiosyncrasies of the training data. However, Keskar *et al.* have found out in their traning experiments that the test curve does not decay after a certain peak iteration and creates the gap between LB and SB methods. Instead, both the test and train curve gradually increase and converge to two asymptotic values, which incur the gap of test accuracy. Therefore, the generalization gap exists irrelevant with overfitting. As such, early-stopping heuristics aimed at preventing models from over-fitting would not help reduce the generalization gap.

3.2. Generalization Gap and Saddle Points

The conjecture about LB methods being easier to be trapped at saddle points are not supported by numerical results either. According to Yao *et al.*, with larger training batch size and same training epochs (100 epochs in their experiments), the converged minimum has a more positively larger Hessian Spectrum and total gradient.(Yao et al., 2018) Such numerical observation contradicts with the assumption that LB methods are prone to stuck at saddle points, which would results in very small total gradient and negative eigenvalues. Therefore the numerical results have suggested that LB methods do not get their degradation in performance due to converging to non-minimum.

3.3. Generalization Gap and Sharp Minimum

3.3.1. Initial Motivation

Keskar *et al.* makes the important state-of-art claim, that large-batch training methods generalize poorly is because of the fact that they converge to sharp minimums while the small-batch training methods converge to flat minimums. The sharpness of a minimum x^* is defined in the way of how rapidly could the loss function increase around a small neighborhood of the minimum x^*. A flat minimum could be described with low precision, whereas the sharp minimum requires higher precision since the function is very sensitive in changing value around x^*. Therefore, the large sensitivity of the sharp minimum, which large-batch training methods tend to converge to, negatively impacts the ability of the model to generalize on new data.

The result is inspired from the parametric plot of a linear slice of the trained loss function. It shows the one-dimensional values of the loss function along the line $\alpha x_l^* + (1-\alpha)x_s^*$ that comes across the minimum x_l^*, x_s^* of both LB and SB methods. It turns out that in the parametric plots, though x_l^*, x_s^* have the same value, x_l^* has a much sharper neighborhood than x_s^*. Such discovery drives further analysis over the relation of generalizability and sharpness in full dimension space.

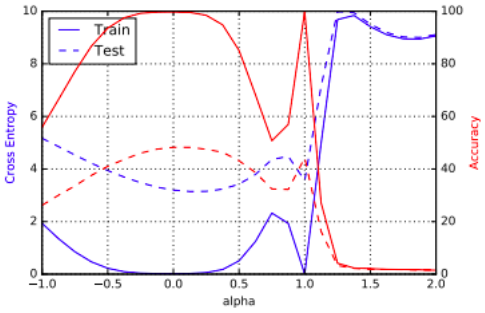


Figure 1. A linear parametric plot sample from the original paper

3.3.2. The Definition of Sharpness

Ideally, the sharpness of a minimum x in parameter space is characterized by the magnitude of eigenvalues of $\nabla^2 f(x)$, where $f(\cdot)$ is the loss function. However, due to the prohibitive cost for computing the Hessian and eigenvalues in deep learning applications, Keskar *et al.* instead employs a sensitivity measurement to define the sharpness of minimum.

The sensitivity measurement characterizes the sharpness

of a minimum by computing the largest function variation within a certain neighborhood of the minimum point $x \in R^n$. Specifically, let C_ϵ denote a small L_1 box around the minimum:

$$C_\epsilon = \{z \in R^n : -\epsilon(|x_i| + 1) \leq z_i \leq \epsilon(|x_i| + 1)\}$$
$$\forall i \in 1, 2, ..., n$$

The "ϵ - sharpness" on the entire space has form of:

$$\Phi_{x,f}(\epsilon) = \frac{(\max_{y \in C_\epsilon} f(x + y)) - f(x)}{1 + f(x)} \times 100$$

However, the sensitivity measurement above considering the entire L_1 box has the potential to be misled by the case where a large value of f is only attained in a tiny subspace of C_ϵ. To avoid such unbalanced impact on the sharpness measurement, Keskar *et al.* has also defined a modified version of "ϵ - sharpness" on random manifold. For such purpose, they introduced a matrix $A \in R^{n \times p}$, where p determines the dimension of the manifold. The columns of matrix A are randomly generated. In the paper's experiment, p is chosen as 100.

The modified version of L_1 box C_ϵ's definition:

$$C_\epsilon = \{z \in R^p : -\epsilon(|(A^\dagger x)_i| + 1) \leq z_i \leq \epsilon(|(A^\dagger x)_i| + 1)\}$$
$$\forall i \in 1, 2, ..., p$$

where $A^\dagger$ denotes the pseudo-inverse of A.

And the "ϵ - sharpness" on random manifold:

$$\Phi_{x,f}(\epsilon, A) = \frac{(\max_{y \in C_\epsilon} f(x + Ay)) - f(x)}{1 + f(x)} \times 100$$

3.3.3. Experiments results

Based on the definition of sharpness, Keskar *et al.* has conducted experiments over the correlation between test accuracy and minimum's sharpness with several toy models and popular datasets. The toy models' structures were designed to exemplify popular configurations used in pratice like AlexNet and VGGNet. Their configuration details are shown in the Table 1 (For detailed architectures please see the appendix of the original paper).

Each network is trained in both LB and SB method of ADAM optimizer with 100 epochs, where for LB method a batch size of 10% of total data size is used, and for SB method the batch size is 256. To increase the credibility, all experiments are conducted 5 times with different uniformly distributed random starting points. The mean and standard deviation of both training and test accuracy are recorded in Table 2, along with the measurement of "ϵ - sharpness" in both entire space and random lower-dimensional manifold in Table 3 and 4.

Based on the results, Keskar *et al.* has claimed that numerically LB methods would result in worse test accuracy and meanwhile converging to sharper minimum. Hence, the conjecture about generalization gap caused by minimum's sharpness is supported by numerical results.

3.3.4. The limitation of ϵ - sharpness measurement

Unfortunately, the measurement of sharpness is not perfect and has been proved by (Dinh et al., 2017) as ill-defined under certain cases. Dinh *et al.* cast their doubt on the "ϵ - sharpness" definition by pointing out the symmetric property of ReLu-based deep learning network called "non-negative homogeneity", and analyze its negative impact on the sharpness measurement.

For all ReLu-based K-layer network they have form of:

$$y = \Phi_{rect}\Big(\Phi_{rect}\big(...\Phi_{rect}(x \cdot \theta_1)...\big) \cdot \theta_{K-1}\Big) \cdot \theta_K$$

where x is the input vector, y is the output vector, θ_k is the weight parameters of the k^{th} layer, and Φ_{rect} is the ReLu activation function.

However, ReLu activation function has the special property of "non-negative homogeneity", which essentially would allow interchanging the operation order of non-negative multiplication and ReLu-activation:

$$\forall (x_i, \alpha) \in R \times R^+, \Phi_{rect}(\alpha \cdot x_i) = \alpha \cdot \Phi_{rect}(x_i)$$

Therefore, utilizing such property, one could scale down the weight parameters in one layer by α and scale up the weight of another layer by α, and the network output would still be the same:

$$\Phi_{rect}\big(x \cdot (\alpha \cdot \theta_1)\big) \cdot \theta_2 = \Phi_{rect}(x \cdot \theta_1) \cdot (\alpha \cdot \theta_2)$$

Dinh *et al.* have named such operation interchange as alpha-transformation. To be more specifically, for any two layers of a network, the configuration and test result would be same as long as the two layers' parameters are equivalent under alpha-transformation:

$$T_a : (\theta_1, \theta_2) \longrightarrow (\alpha\theta_1, \alpha^{-1}\theta_2)$$

Based on the unidentifiability, Dinh *et al.* propose a case where two network configurations having same output (test

Name	Network Type	Data set
F1	Fully Connected	MNIST
F2	Fully Connected	TIMIT
C1	(Shallow) Convolutional	CIFAR-10
C2	(Deep) Convolutional	CIFAR-10
C3	(Shallow) Convolutional	CIFAR-100
C4	(Deep) Convolutional	CIFAR-100

Table 1. Network Configurations

	Training Accuracy		Test Accuracy	
Name	SB	LB	SB	LB
F1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.3\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

Table 2. Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

	$\epsilon = 10^{-3}$		$\epsilon = 5 \times 10^{-4}$	
Name	SB	LB	SB	LB
F1	1.23 ± 0.83	205.14 ± 69.52	0.61 ± 0.27	42.90 ± 17.14
F2	1.39 ± 0.02	310.64 ± 38.46	0.90 ± 0.05	93.15 ± 6.81
C1	28.58 ± 3.13	707.23 ± 43.04	7.08 ± 0.88	227.31 ± 23.23
C2	8.68 ± 1.32	925.32 ± 38.29	2.07 ± 0.86	175.31 ± 18.28
C3	29.85 ± 5.98	258.75 ± 8.96	8.56 ± 0.99	105.11 ± 13.22
C4	12.83 ± 3.84	421.84 ± 36.97	4.07 ± 0.87	109.35 ± 16.57

Table 3. Sharpness of Minimum in Full Space

	$\epsilon = 10^{-3}$		$\epsilon = 5 \times 10^{-4}$	
Name	SB	LB	SB	LB
F1	0.11 ± 0.00	9.22 ± 0.56	0.05 ± 0.00	9.17 ± 0.14
F2	0.29 ± 0.02	23.63 ± 0.54	0.05 ± 0.00	6.28 ± 0.19
C1	2.18 ± 0.23	137.25 ± 21.60	0.71 ± 0.15	29.50 ± 7.48
C2	0.95 ± 0.34	25.09 ± 2.61	0.31 ± 0.08	5.82 ± 0.52
C3	17.02 ± 2.20	236.03 ± 31.26	4.03 ± 1.45	86.96 ± 27.39
C4	6.05 ± 1.13	72.99 ± 10.96	1.89 ± 0.33	19.85 ± 4.12

Table 4. Sharpness of Minimum in Random Subspaces of Dimension 100

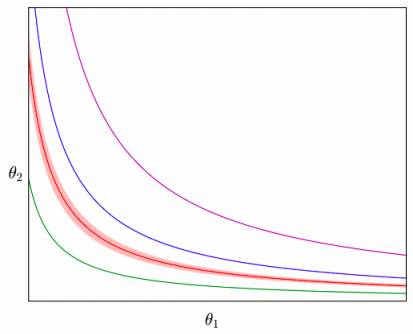


Figure 2. Illustration of effect of homogeneity. Each curve is a collection of (θ_1,θ_2) that result in same output.

accuracy) have different sharpness measurements within "ϵ - sharpness" metric. According to Figure 3 below, the two configuration points on the hyperbola have the same network output hence the same test accuracy. However, $\theta^{'}$ is very close to $\theta_2 = 0$, where one layer of the network has all its weight being zero. Such configuration would result in a constant target function, which is one of the worst approximation of the data and therefore the loss value would be extremely large. Therefore, the non-deterministic feature here of ϵ - sharpness suggests that such metric needs to be refined in order to better characterize the relationship between network generalizability and minimum sharpness.

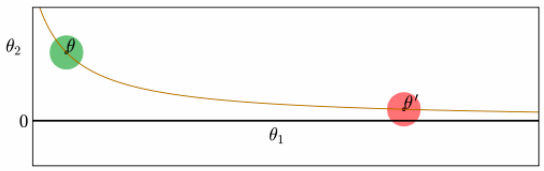


Figure 3. Illustration of the ill-performance of ϵ - sharpness metric. Configuration θ and $\theta^{'}$ have same test accuracy yet different sharpness

In general, Keskar *et al.* have defined the notion of ϵ - sharpness and conducted experiments that showed generalization accuracy and minimum sharpness are strongly correlated. On the other hand, Dinh *et al.* have shown that the definition of ϵ - sharpness becomes non-deterministic when considering the alpha-transformation of weight parameters between network layers, and therefore are not qualified enough to be the causer of generalization gap. However, there still lack theoretical work that either prove or disprove the existence of causality between minimum sharpness and generalizability. Hence, such area is still a open problem for future research. Some potential works may involve refining or re-designing the notion of sharpness that would be invariant under the symmetric transformation of network.

3.4. Summary

In general, the theoretical analysis over the causer of generalization gap between LB and SB methods is still an open problem. The previous work of experiments have disproved the naive claims like generalization gap being caused by overfitting and saddle points. One of the most popular idea on the reason of generalization gap is that LB methods converge to sharper minimum that generalizes poorly. With certain mathematical definition of sharpness (e.g. ϵ - sharpness), such idea has been supported with numerical results that show strong correlation between sharpness and test accuracy of LB and SB methods. However, current definitions of sharpness all suffer from the non-deterministic feature, which is caused by the symmetric transformation of the network weight. Therefore, the refinement over sharpness's definition and more theoretical analysis is still required in future. What's more, other theories or hypothesis are also worthwhile exploring. One of the very promising direction also mentioned in Keskar *et al.* is the relationship between the minimum-and-starting-point distance $d(x_0, x^*)$ and the test accuracy, which tries to manifest the explorativity of LB and SB methods and its relation to generalizability.

4. Strategies for reducing the Generalization Gap

Although there are still arguments from different aspects about the reasons of the generalization gaps, researchers have found some interesting tricks for closing those generalization gaps.

4.1. Ghost Batch Normalization

Batch normalization (BN) algorithm is widely applied in deep learning areas, especially in the convlutional nerual networks. BN is knwon to increase the models' robustnesses, accelerate training processes, and improve generalization. In order to overcome the drawback of BN that it is bounded to depend on the choosen batch size, Hoffer *et al.* applied the ghost batch normalization (GBN) method instead of the traditional BN in training networks (Hoffer et al., 2017). Details of the GBN algorithm are given in Algorithm 1. This algorithm reduces the generalization error substantially.

4.2. Adapting larger numbers of weight updates

Some researches argued that the poor generalization performance of large-batch training is simply because the limited iteration number of models' weights (Smith et al., 2017). For example, the number of weights updating iterations of batch size 64 is 16 times compared to the training under batch size of 1024. As shown in Figure 4, the number of iterations of the weight decreases as the batch size increases. Not only that, we can also observe that when the batch size

is large, such as 1024, the training loss curve does not seem to completely converge. In view of this situation, people argued: we can solve the problem of generalization gaps by sufficiently training a large number of iterations for large batch sizes.

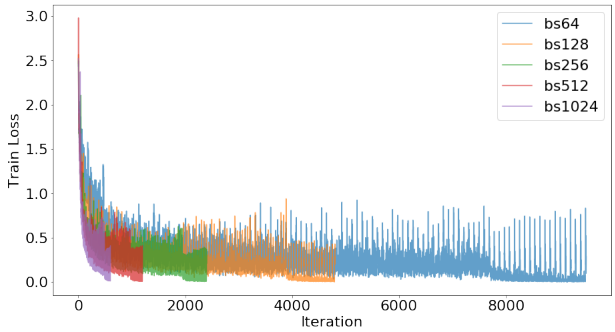


Figure 4. Training loss via number of iterations for different batch sizes

5. Experiments and Results: Closing the Generalization Gap

In this section, experiments are conducted for exploring the relationships between batch sizes and generalization gaps. A ResNet 50 model is applied in training through CIFAR 10 and CIFAR 100 dataset with SGD optimizer. The training platform is: CPU: Intel I9-9900k, GPU: Dual RTX 2080 Ti, RAM: 128 GB.

5.1. Generalization gaps of different batch sizes

In order to get better understanding of the batch sizes and generalization errors, we plot the error graph via epochs.

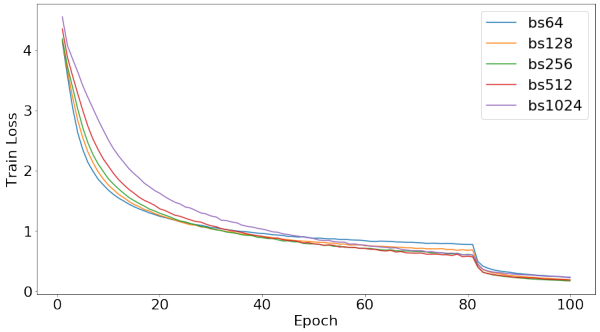


Figure 5. Training loss via number of epochs for different batch sizes

Figure 5 shows the training loss curve of different batch sizes. It can be learned that although the converging rate is different, the training loss of different batch sizes converge into the same range. However, in Figure 6, what can be observed is that as the batch size becomes larger, the

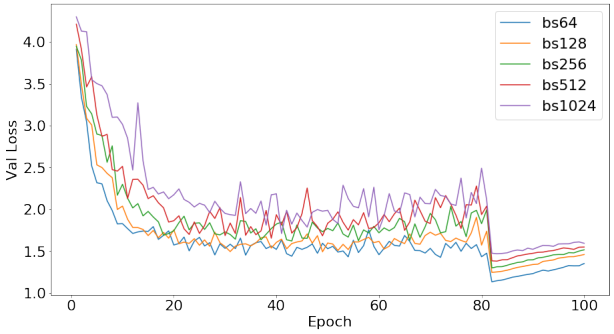


Figure 6. Validation loss via number of epochs for different batch sizes

model validation error also increases significantly. This phenomenon verifies the fact that under the condition that other variables are the same, the larger the batch size, the weaker the generalization performance of the model.

5.2. Ghost Batch Normalization

In the paper by Hoffer et al., they demonstrated that GBN can reduce generalization errors (Hoffer et al., 2017). We have emerged from this GBN test based on our previous experiment. We set the batch size to 1024 and the virtual batch size to 64 to verify the feasibility of the GBN method.

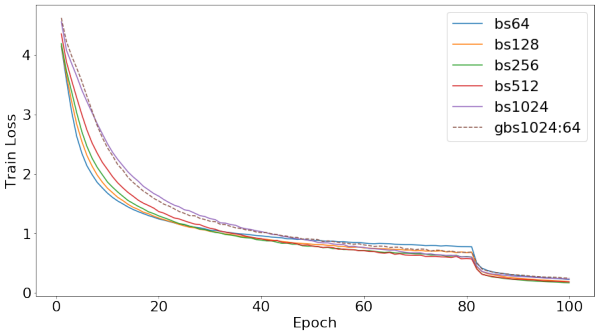


Figure 7. Training loss via number of epochs for Ghost Batch Normalization

Figure 7 8 show the train and validation loss for different batch sizes and the GBN method of 1024-64. In the training error curve, it shows that the GBN algorithm converges to the same interval as other ordinary methods; whereas we can easily find that in the test error curve, the GBN algorithm with a 1024-bit batch size has converged to a size similar to the location of batch size of 128, which is significantly lower than the batch size of 1024. This finding shows that the GBN method can indeed help the model reduce generalization errors.

Algorithm 1 Ghost Batch Normalization (GBN), applied to activation x over a large batch B_L with virtual mini-batch B_S. Where $B_S < B_L$.

Require: Values of x ovver a large-batch: $B_L = x_{1...m}$ size of virtual batch $|B_S|$; parameters to be learned: γ, β, momentum η.

Training Phase:

Scatter B_L to $\{X^1, X^2, ..., X^{|B_L|/|B_S|}\} = \{x_{1...|B_S|}, x_{|B_S|+1...2|B_S|}...x_{|B_L|-|B_S|}...m\}$

$\mu_B^l \leftarrow \frac{1}{|B_S|}\sum_{i=1}^{|B_S|} X_i^l$ for $l = 1, 2, 3...$, {calculate ghost mini-batches means}

$\sigma_B^l = \sqrt{\frac{1}{|B_S|}\sum_{i=1}^{|B_S|}(X_i^l - \mu_B)^2 + \epsilon}$ for $l = 1, 2, 3...$, {calculate ghost mini-batches std}

$\mu_{run} = (1-\eta)^{|B_S|}\mu_{run} + \sum_{i=1}^{|B_L|/|B_S|}(1-\eta)^i \eta \mu_B^l$

$\sigma_{run} = (1-\eta)^{|B_S|}\sigma_{run} + \sum_{i=1}^{|B_L|/|B_S|}(1-\eta)^i \eta \sigma_B^l$

return $\gamma\frac{X^l - \mu_B^l}{\sigma_B^l} + \beta$

Training Phase:

return $\gamma\frac{X - \mu_{run}}{\sigma_{run}} + \beta$ {scale and shift}

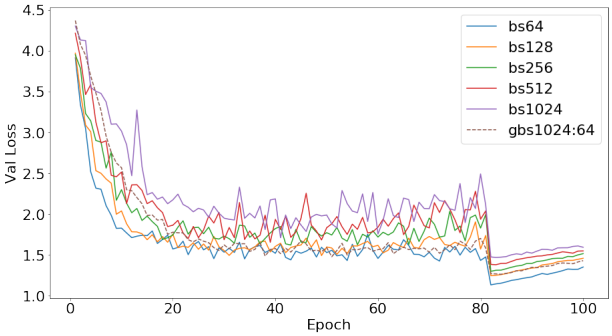


Figure 8. Validation loss via number of epochs for Ghost Batch Normalization

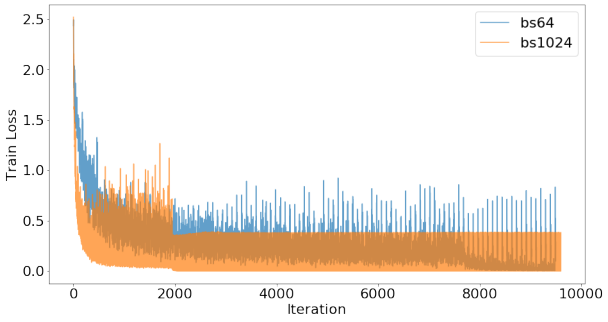


Figure 9. Training loss for batch size of 64 and 1024 with same weights updating iterations

5.3. Larger weights updating iterations

As discussed in the previous section 4.2, researchers discussed the reason that there is a generalization error for large batch sizes simply because the number of iterations of the parameter update is not enough. In order to fully verify this view, we try to make the same number of iterations for training a model with large batch as for a model with small batch.

From Figure 9, it gives that despite the different convergence rates, under the same number of iterations, a large batch size model and a small batch size model converge to very close places. The results of this experiment show that insufficient number of iterations is indeed a factor affecting the generalization of a model trained through large batch size.

5.4. Performances Summary

So far, we have conducted the experiments of using different batch sizes in training, applying ghost batch normalization (GBN), training with larger epochs when the batch size is large (LE). The performances of different settings are compared in this section.

Dataset	SB	LB	GBN	LE	GBN + LE
CIFAR 10	92.78%	90.32%	91.48%	92.72%	**94.13%**
CIFAR 100	69.89%	65.26%	68.03%	69.42%	**71.04%**

Table 5. The validation performances of small batch size and batch size with different settings

As shown in Table 5, we can find that under normal circumstances, SB performs better than LB. After using GBN, such a gap is closed; and when we iterate the same number of parameters for LB and SB training, the performance of the two has almost reached a level. Finally, we combined GBN and LE, the result is that large batch sizes outperform small batch sizes.

6. Conclusion

In this project, we investigated the relationship between SGD-like optimization using different batch sizes and generalization errors. It gives that larger batch sizes lead to

greater generalization gaps. We start by reading the paper by Keskar et al to find out explanations for this phenomenon, and they claim generalization gap is due to that large batch size will introduce the optimization function into the sharp minimum. The sharpness of LB minimum will then affect the generalization of the model. However, the theoretical explanation for sharpness-generalizability relation is still lacked, and other researches have shown that the metric for sharpness used in Keskar et al. is not perfect and will be ill-defined under some test cases. Hence the theoretical analysis in such field is still open for future. Fortunately, practically speaking, we found some ways to reduce the generalization error. Through GBN and LE methods, we can solve the generalization problem caused by large batch size.

References

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. *CoRR*, abs/1703.04933, 2017. URL http://arxiv.org/abs/1703.04933.

Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pp. 1731–1741, 2017.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Yao, Z., Gholami, A., Lei, Q., Keutzer, K., and Mahoney, M. W. Hessian-based analysis of large batch training and robustness to adversaries. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 4949–4959. Curran Associates, Inc., 2018.

Role of Margin in Neural Networks Generalization

Shabnam Daghaghi [1]

Abstract

In recent years deep neural networks demonstrated widespread success in a broad variety of tasks such as computer vision, speech recognition, and natural language processing. It is shown that neural networks are able to generalize relatively well while they have enormous capacity to overfit any given dataset. A very popular line of research recently is focused on explaining generalization behavior of networks. It has been shown that generalization performance of neural networks depends on training data as well as model and optimization parameters, therefore classic complexity measures based on parameter counting such as VC dimension are not applicable. In this work we focus on explaining network generalization through the lens of margin and margin distribution. The importance of margin in designing classifiers are presented and margin distribution is introduced. Several very recent methods to improve neural network margin are elaborated and the relationship between margin distribution and generalization gap is presented.

1. Introduction

Deep learning models demonstrate different generalization behavior comparing to classic machine learning models and generalize well despite their substantial complexity (number of parameters). This behavior is shown in Figure 1 which is sometimes interpreted as implicit regularization i.e. the effective capacity of a network is implicitly controlled and constrained by combination of network architecture, its parameters and training data (Neyshabur et al., 2014; 2017b).

Recently in efforts to explain and theorize generalization behavior of neural networks, several novel complexity measures such as weight norm, unit capacity (weight distance to initialization), margin (along with margin distribution) and sharpness are proposed (Bartlett et al., 2017; Golowich et al., 2017; Neyshabur et al., 2018; Jiang et al., 2018; Keskar et al., 2016; Hochreiter & Schmidhuber, 1995). Complexity measures for neural networks should be able to explain improvement of generalization with over-parameterization (Neyshabur et al., 2018). Neural networks differ from classic machine learning models in terms of the impact of over-parameterization. It is shown that generally as the size of network increases test error improves (Figure 2) which demands exploring novel complexity measures and generalization bounds (Neyshabur et al., 2018).

One category of the proposed network complexity measures is based on margin and margin distribution (Garg & Roth, 2003; Elsayed et al., 2018; Jiang et al., 2018). Classic hinge loss promotes larger margin and can result in relatively better generalization comparing to other classification loss functions. However, it has been shown that the extreme definition of margin (distance of closest points to the boundary) does not effectively represent generalization behavior (Garg & Roth, 2003). For an improved notion of margin, all data points should contribute in calculation of margin which is referred as margin distribution. Garg & Roth (2003) showed that margin distribution highly correlates with generalization error. Following the same idea, Jiang et al. (2018) proposed to utilize margin distributions of all network layers to predict generalization gap. Additionally, novel large margin losses based on margin distributions are proposed to improve generalization behavior of neural networks (Elsayed et al., 2018).

Existing loss functions e.g. Softmax loss can reach zero loss on training data that is not necessarily an indication of good generalization error (e.g. in case of random data), therefore we need to design new loss functions (or modify existing ones) to better reflect the generalization behavior. For instance since margin distribution correlates well with generalization gap (error), the new improved loss should also include some information about the margin (Jiang et al., 2018). Hence with this new loss, the training loss of a random labeled data cannot reach zero indicating that the generalization performance is going to be poor.

The popular Softmax loss has some drawbacks such as overconfidence in probability and unbounded penalty for outliers. Several improvements are suggested to improve discriminative power of Softmax loss and increase its margin. Among these improved losses are bi-tempered Softmax loss (Amid

[1]Department of Electrical and Computer Engineering, Rice University.

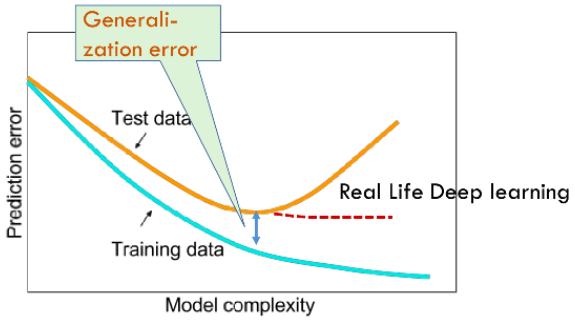


Figure 1. Generalization behavior of deep learning models vs. classic machine learning models (https://desh2608.github.io/2018-07-27-deep-learning-theory-2/).

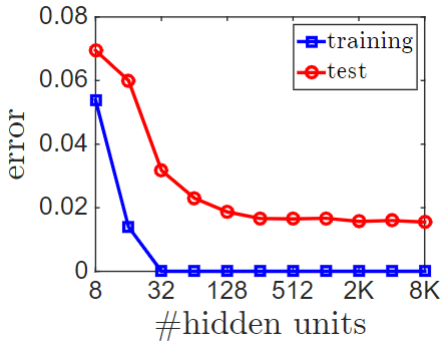


Figure 2. How neural networks generalize when number of parameters increases (Neyshabur et al., 2017a).

et al., 2019) and angular margin Softmax losses (Liu et al., 2017; 2016; Deng et al., 2019; Wang et al., 2018).

This report first outlines definition of margin and margin distribution and their properties in Section 2. Next, in Section 3 novel loss functions to improve neural network generalization are introduced. Finally in Section 4 conclusions and discussions are presented.

2. Margin Definition and Properties

Margin is often defined based on either discriminant function (score) values or distance of a point to decision boundary. The difference of the target class score and maximum score of other classes can be defined as margin[1]:

$$M(f, y) = f_y - \max_{i \neq y} f_i \tag{1}$$

[1] https://www.stat.berkeley.edu/~bartlett/talks/201710MLaB.pdf

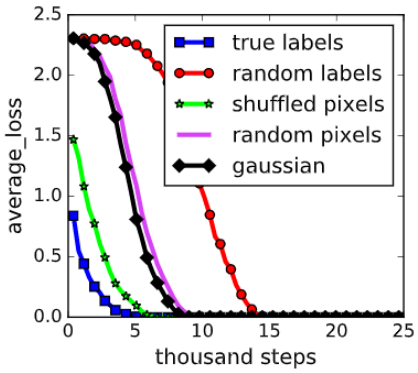


Figure 3. Capability of deep networks to fit random data due to their enormous capacity (Zhang et al., 2016).

The popular method to calculate margin is the geometric interpretation based on the closest distance of data points to decision boundaries. However this distance calculation is intractable for general nonlinear boundaries e.g. in neural networks, so some approximations are proposed to obtain the margin. Elsayed et al. (2018); Jiang et al. (2018) applied first order Taylor series approximation to calculate margin and margin distributions. Elsayed et al. (2018) explored obtaining margin of all layers of neural network and proposed a new large margin loss (utilizing margin distribution of all layers) based on multi-class hinge loss to improve generalization of networks. Their new large margin loss guarantees robustness against noisy labels and outliers close and far away from the boundary. The distance (margin) of data point x to the boundary of classes i, j is defined as:

$$d_{f,x,\{i,j\}} \triangleq \min_{\delta} \|\delta\|_p \quad \text{s.t.} \quad f_i(x+\delta) = f_j(x+\delta) \tag{2}$$

Elsayed et al. (2018) proposed to obtain margin via linearization:

$$\tilde{d}_{f,x,\{i,j\}} \triangleq \min_{\delta} \|\delta\|_p \quad \text{s.t.} \quad f_i(x) + \langle \delta, \nabla_x f_i(x) \rangle = f_j(x) + \langle \delta, \nabla_x f_j(x) \rangle \tag{3}$$

Finally margin can be approximated using the discriminant function values and their gradients:

$$\tilde{d}_{f,x,\{i,j\}} = \frac{|f_i(x) - f_j(x)|}{\|\nabla_x f_i(x) - \nabla_x f_j(x)\|_q} \tag{4}$$

The simplified geometric interpretation of Equation 4 can be explained by Figure 4 where a schematic of a three class

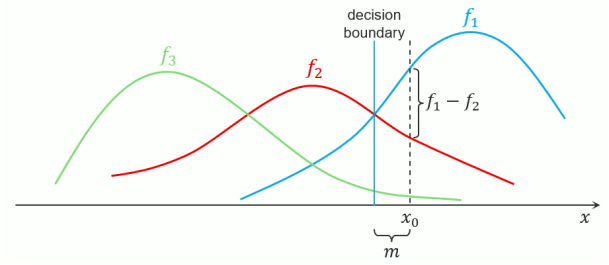


Figure 4. Relationship of margin and confidence while linearizing discriminant functions.

classification problem with one dimensional feature space is shown. It demonstrates the relationship of margin and classification scores as follows which is the simplified form of Equation 4:

$$m = \frac{f_1 - f_2}{f_1' - f_2'} \tag{5}$$

Most large margin methods are only applicable to shallow models (classic machine learning models as opposed to deep neural networks) and consider large margin only in a preset feature representation (input space) and do not guarantee large margin in the intermediate representation spaces (Elsayed et al., 2018; Jiang et al., 2018). Deep neural networks include several layers leading to several intermediate spaces also known as intermediate representations. Elsayed et al. (2018) shows that to enhance generalization of neural networks we need to improve margin of all representation spaces along with margin of input space and it proposes a large margin loss function based on the margin distribution of all representation spaces to improve generalization error. Figure 5 schematically shows the representation spaces along with the input space for a typical fully connected network. To obtain margin of an intermediate layer we can simply take the gradient of discriminant functions with respect to the activations of the corresponding layer, $\nabla_h f(x)$, instead of input features x in Equation 4.

This definition of margin is applicable to any network architecture. The margin distribution based loss shows considerable improvements in several difficult learning scenarios such as small training sets, corrupted labels, and adversarial examples. However, calculation of the proposed loss is computationally expensive and may increase run time by 60% due to presence of gradients (first order derivative) in the loss which requires calculation of second order derivatives during backpropagation. (Elsayed et al., 2018).

Commonly large margin machine learning methods do not utilize margin distribution and only rely on the margin of closest data points to the decision boundary. Garg & Roth (2003) proposed to improve generalization of linear classifiers by considering margin distribution in form of a weighted margin term in the loss function. In the context of linear classification they showed that the correlation of test error with weighted margin of training data (margin distribution) is considerably more than the correlation of test error with regular margin. This motivates margin distribution as the superior predictor for generalization error.

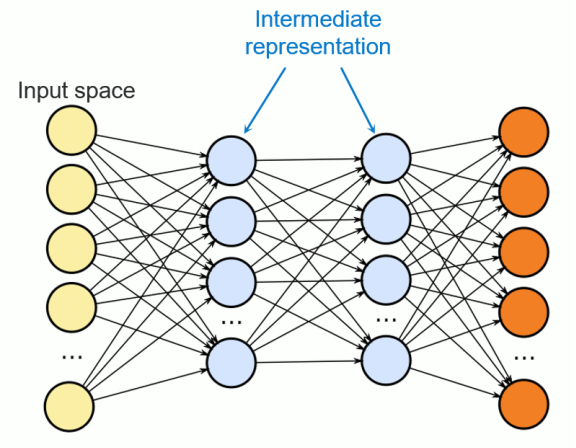


Figure 5. Networks generalization depends on the margin of input space as well as intermediate representations.

Jiang et al. (2018) empirically showed that margin distribution of all layers in a network highly correlates with the generalization behaviour. They trained three convolutional neural networks with cross-entropy loss on CIFAR-10 dataset with three different levels of label corruptions. As Figure 7 shows, as test accuracy increases, margin distributions of all layers are skewed to the right (margin increases) confirming the impact of margin of all layers in a network on its generalization error. Hence, Jiang et al. (2018) proposed utilizing margin distributions as a predictor for generalization gap. They first summarized margin distributions of all layers (e.g. moments, quartiles) as a vector θ called total signature, then they correlated θ with the generalization error with a simple relationship as follows: $\hat{g} = a \log \theta + b$

Jiang et al. (2018) evaluated their proposed generalization gap predictor on several network architectures and datasets and found out the promising agreement between actual and predicted gap as shown in Figure 8.

3. Novel Loss Functions to Improve Generalization

There are several classification loss functions such as Softmax, hinge, and exponential loss. Classification losses are usually expressed in terms of margin as shown in Figure

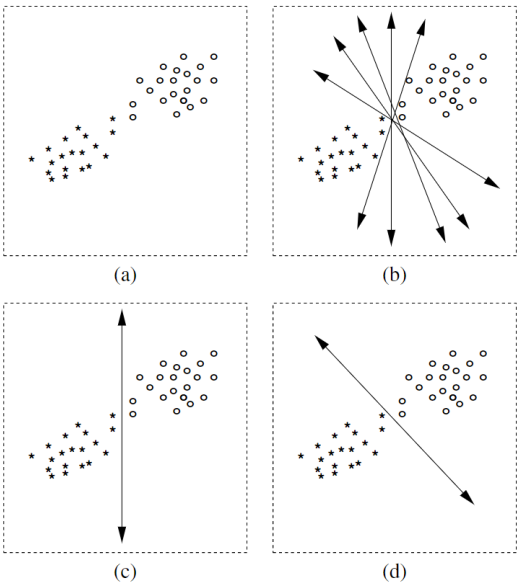


Figure 6. (a) A schematic binary classification problem in 2D. (b) Linear classifiers with zero training error. (c) The hyperplane learned by a large margin classier such as SVM. (d) This hyperplane maybe a better classifier than SVM, since more data points are further apart from the decision boundary (Garg & Roth, 2003).

9. The trivial classification loss is 0-1 loss which simply penalizes all incorrect classifications equally and does not penalize correct classifications. Training a classifier with 0-1 loss is not practical because its minimization is an NP-hard (combinatorial) optimization. Therefore, we approximate 0-1 loss with its smooth versions to have a computationally feasible loss.

The most popular classification loss for deep learning is cross-entropy loss which is also known as Softmax loss, logistic loss or log loss. Softmax loss for each sample is as follows:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) = \log(1+e^{-v}) \qquad (6)$$

where we obtain the last equality by plugging in the predicted probabilities using Softmax function. Usually we express classification loss in terms of margin or signed discriminant function represented by v. One of the important weaknesses of Softmax loss is that it does not provide enough intra-class compactness and inter-class separability (specially for fine-grained classification). This is due to Log and Exp functions in Softmax loss. Log function causes unbounded loss for incorrect classifications and makes it sensitive to noise and Exp function leads to overconfidence in probabilities, hence Softmax loss does not guarantee large margin. Therefore there are several variants of Softmax loss to promote large margin and better generalization. A simple way to improve the margin of Softmax loss is controlling the overconfidence of Softmax function ,similar to temperature Softmax (Hinton et al., 2015), via introducing γ parameter:

$$L_i = \log(1+e^{-\gamma v}); \quad 0<\gamma<1 \qquad (7)$$

where γ acts as a regularizer and smaller γ enforces more regularization and better generalization and leads to a better margin. If $\phi(v)$ represents the loss function, *loss margin* μ is defined as $-\frac{\phi'(0)}{\phi''(0)}$ which has strong relationship with generalization behavior (Masnadi-Shirazi & Vasconcelos, 2015). As shown in Figure 10, improved Softmax with γ parameter has larger loss margin comparing to standard Softmax.

Another simple way to improve the margin and discriminative power of Softmax loss is to directly penalize the probabilities for incorrect classes as well correct classes. The modified Softmax loss will become as follows:

$$-\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log p_j(x_i) - (1-y_{ij})\log(1-p_j(x_i)) \qquad (8)$$

Contrastive loss (Hadsell et al., 2006) and triplet loss (Schroff et al., 2015) are also proposed to enforce a distance constraint to encourage extra intra-class compactness and inter-class separability (based on the idea of pairing samples). However, a subsequent problem is that the number of training pairs and triplets can increase dramatically.

It is shown that weighted margin distribution is a better complexity measure than the extreme notion of margin e.g. in SVM (Garg & Roth, 2003). Authors suggest that all points should contribute to the (generalization) error and the relative contribution of a point should decay as a function of its distance from the boundary. The suggested weight function to build the weighted margin distribution is as follows:

$$W(x_i) = \begin{cases} e^{-\alpha f^2(x_i)} & \text{if } y_i f(x_i) \geq 0 \\ e^{-\beta y_i f(x_i)} & \text{if } y_i f(x_i) < 0 \end{cases} \qquad (9)$$

By plugging in the weighted margins for correct and incorrect classifications we can obtain the final loss function:

$$L(x) = \sum_{i=1}^{n} I(\hat{y}_i = y_i) e^{-\alpha f^2(x_i)} + \sum_{i=1}^{n} I(\hat{y}_i \neq y_i) e^{-\beta y_i f(x_i)} \qquad (10)$$

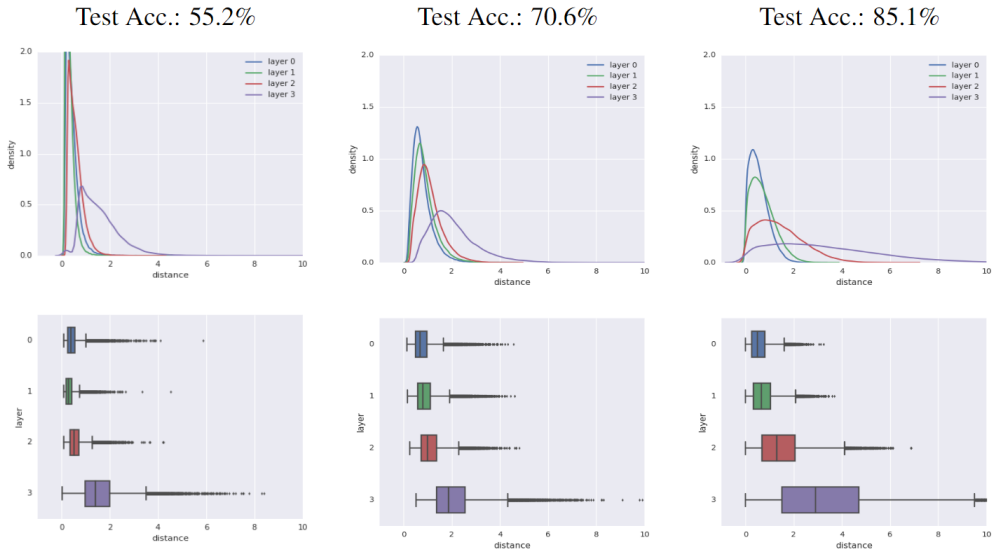


Figure 7. Impact of margin distributions of all layers in network generalization error. This is the result of training CNN on CIFAR-10 with different levels of corrupted labels. The label corruption decreases from left to right and as the consequence the margin distributions are more pulled to the right demonstrating that the margin is becoming larger. As the result the test accuracy increases and the generalization performance improves. All these networks have almost 100% training accuracy. Margin distributions are highly correlated with generalization error and can be used to predict generalization gap (Jiang et al., 2018).

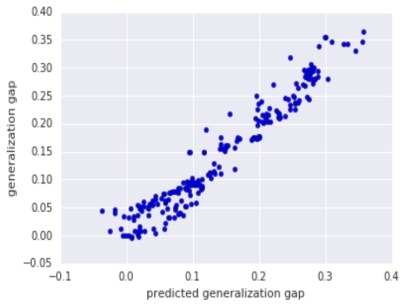


Figure 8. Actual vs. predicted generalization gap. The predicted generalization gap using the signatures of margin distributions shows very good agreement with the actual generalization gap (Jiang et al., 2018).

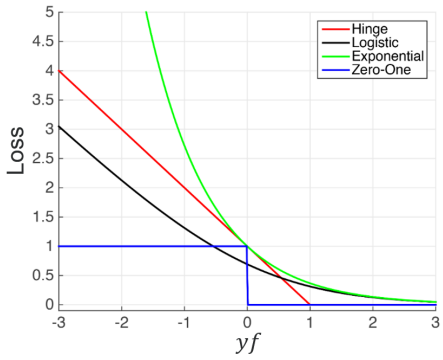


Figure 9. Popular classification loss functions commonly expressed in terms of margin.

where $I(.)$ represents the indicator function. The new loss function shown in Figure 11 will result in an overall better margin and generalization behavior comparing to standard large margin losses e.g. hinge loss. As seen from the shape of this loss function, it is a trade-off between 0-1 loss and hinge loss. Garg & Roth (2003) found out significant correlation between weighted margin on train data with test (generalization) error which suggests margin distribution as a promising predictor of generalization gap.

Softmax cannot handle large and small margin noises (outliers close to or far away from the boundary) which impacts generalization behaviour negatively. This issue rises because of loss unboundedness due to Log function and overconfidence in probabilities due to Exp function. Amid et al. (2019) addresses these issues by replacing Log and Exp functions with their corresponding *tempered* versions shown in Equations 11 and 12. The proposed bi-tempered Softmax loss is bounded which handles large-margin outliers and has a heavy tail which handles small-margin mislabeled examples. As the result training is more robust to noise while two tunable temperature parameters are introduced.

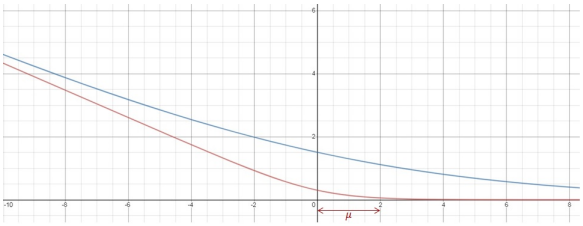


Figure 10. The margin of logistic or Softmax loss can be increased by slightly modifying the loss equation. Red curve (Equation 6) shows the standard logistic loss and blue curve (Equation 7) is the increased margin logistic loss. μ which is called *loss margin* is directly related to generalization behavior (Masnadi-Shirazi & Vasconcelos, 2015).

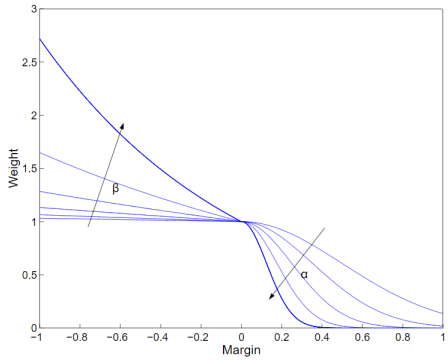


Figure 11. The new classification loss considering the margin distribution. The weight given to the data points as a function of their margin (Garg & Roth, 2003).

$$\log_t(x) := \frac{1}{1-t}(x^{1-t} - 1) \quad 0 \le t < 1 \tag{11}$$

$$\exp_t(x) := [1 + (1-t)x]_+^{\frac{1}{1-t}} \quad t > 1 \tag{12}$$

In face recognition applications (fine-grained classification), a proposed improvement to Softmax loss is enhancing the angular margin. Deng et al. (2019) proposes an additive angular margin loss (ArcFace) to improve the discriminative power of face recognition models. Dot product between (CNN extracted) feature and last fully connected layer weight is interpreted as cosine distance (after normalization), therefore angular penalties can be incorporated to cosine distance to improve Softmax loss. Standard Softmax loss is as follows:

$$-\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}} \tag{13}$$

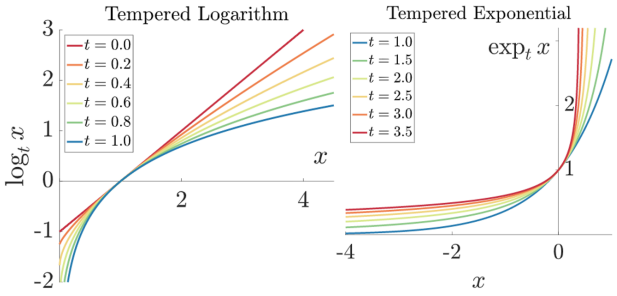


Figure 12. Tempered versions of Log and Exp functions to improve generalization of Softmax loss (Amid et al., 2019).

where $W_j^T x_i$ is replaced by $||W_j||\,||x_i||\cos\theta_j$ to show the impact of angular margin θ_j. After weight normalizing we have:

$$-\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos\theta_{y_i}}}{e^{s\cos\theta_{y_i}} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos\theta_j}} \tag{14}$$

Additive angular margin penalty m is added between feature and weight to simultaneously enhance intra-class compactness and inter-class discrepancy:

$$-\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(\cos(\theta_{y_i}+m))}}{e^{s(\cos(\theta_{y_i}+m))} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos\theta_j}} \tag{15}$$

The impact of angular penalty in improving the discriminative power of classifier is shown schematically in Figure 14. For instance in case of multiplicative angular penalty Figure 13 shows the gradual improvement of class separability as angular penalty increases.

To improve Softmax loss three kinds of angular margin penalties are proposed: SphereFace/L-Softmax (Liu et al., 2017; 2016), ArcFace (Deng et al., 2019), and CosFace (Wang et al., 2018) that are based on multiplicative angular margin m_1, additive angular margin m_2 and additive cosine margin m_3, respectively. Equation (16) shows the final improved Softmax loss when we combine all three angular penalties. Angular margin penalties, no matter added on angle or cosine space, all enforce intra-class compactness and inter-class diversity by penalizing target logit.

$$-\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(\cos(m_1\theta_{y_i}+m_2)-m_3)}}{e^{s(\cos(m_1\theta_{y_i}+m_2)-m_3)} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos\theta_j}} \tag{16}$$

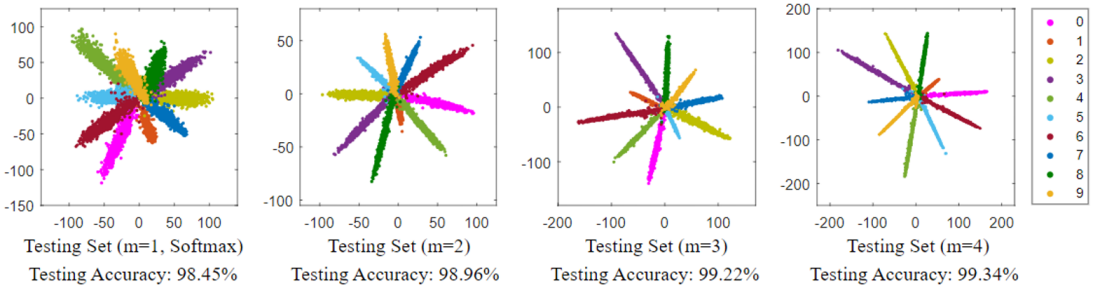


Figure 13. The impact of different multiplicative angular margins. As margin multiplier increases, class separations further improve (Liu et al., 2016).

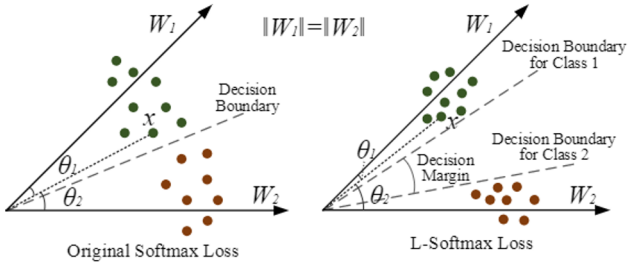


Figure 14. A schematic representation to show the impact of angular margin. Incorporating angular margin enforces better class separation (Liu et al., 2016).

4. Discussion

It has been shown that classic measures of complexity fail to explain neural networks generalization behaviour. As the result, novel measures such as norm, margin and sharpness are introduced. Proper complexity measures should include the impact of model parameters and training data. In this report, we focused on margin and margin distribution as effective measures to capture network generalization gap. And the importance of ensuring large margin in all representation spaces of neural network is presented. New loss functions that properly reflect network generalization behavior are outlined and in particular recent improvements of Softmax loss are reported.

References

Amid, E., Warmuth, M. K., Anil, R., and Koren, T. Robust bi-tempered logistic loss based on bregman divergences. *arXiv preprint arXiv:1906.03361*, 2019.

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pp. 6240–6249, 2017.

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4690–4699, 2019.

Elsayed, G., Krishnan, D., Mobahi, H., Regan, K., and Bengio, S. Large margin deep networks for classification. In *Advances in neural information processing systems*, pp. 842–852, 2018.

Garg, A. and Roth, D. Margin distribution and learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 210–217, 2003.

Golowich, N., Rakhlin, A., and Shamir, O. Size-independent sample complexity of neural networks. *arXiv preprint arXiv:1712.06541*, 2017.

Hadsell, R., Chopra, S., and LeCun, Y. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pp. 1735–1742. IEEE, 2006.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hochreiter, S. and Schmidhuber, J. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pp. 529–536, 1995.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. Predicting the generalization gap in deep networks with margin distributions. *arXiv preprint arXiv:1810.00113*, 2018.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Liu, W., Wen, Y., Yu, Z., and Yang, M. Large-margin softmax loss for convolutional neural networks. In *ICML*, volume 2, pp. 7, 2016.

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 212–220, 2017.

Masnadi-Shirazi, H. and Vasconcelos, N. A view of margin losses as regularizers of probability estimates. *Journal of Machine Learning Research*, 16(85):2751–2795, 2015. URL http://jmlr.org/papers/v16/masnadi15a.html.

Neyshabur, B., Tomioka, R., and Srebro, N. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*, 2014.

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pp. 5947–5956, 2017a.

Neyshabur, B., Tomioka, R., Salakhutdinov, R., and Srebro, N. Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*, 2017b.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. The role of over-parametrization in generalization of neural networks. 2018.

Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z., and Liu, W. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5265–5274, 2018.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

A Review of Matrix Sensing

Benjamin Coleman [1] * **Gaurav Gupta** [1] *

Abstract

Matrix sensing is the well studied problem of recovering a low rank matrix i.e. a rank r optimal solution $X^\star \in \mathbb{R}^{n\times n}$, given a set of linear measurements. The problem belongs to a class of rank-constrained optimization problems, which includes related problems such as low-rank matrix completion, low-dimensional Euclidean embedding problems and image compression. Matrix sensing has applications in quantum state tomography, system approximation and identification, neuron activity recovery from μECoG and video background subtraction. In this report we present a summary of important results in the matrix sensing literature. We also implement and compare several optimization methods for solving the matrix sensing problem.

1. Introduction

Matrix sensing (Recht et al., 2010) is an important optimization problem in quantum state tomography, phase retrieval and other applications. The task is the matrix analogue to compressed sensing: Assuming a low-rank $X^\star \in \mathbb{R}^{n\times n}$, we wish to recover an estimate X of $X^\star$ from a set of m linear measurements of the form $y_i = \langle X, A_i\rangle$. In compressed sensing, the signal is assumed to have a sparse representation in some signal basis. In matrix sensing, the signal $X^\star$ is assumed to be sparse in terms of the nuclear norm. The problem is interesting when the number of measurements $m << n \times n$. In this regime, we can efficiently sample and recover $X^\star$ using a small set of fixed measurement operations.

*Equal contribution [1]Department of Electrical and Computer Engineering, Rice University, Houston TX, USA. Correspondence to: Gaurav Gupta <gaurav.gupta@rice.edu>, Ben Coleman <ben.coleman@rice.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

2. Matrix Sensing

Formally, the matrix sensing problem is as follows. Given a measurement mechanism $\mathcal{A}$ with m measurement matrices A_i, matrix sensing requires a solution to an optimization problem.

$$X = \arg \min_{rank(\mathbf{X})\leq r} \frac{1}{2}\sum_{i=0}^{m-1}(y_i - \langle \mathbf{A}_i, \mathbf{X}\rangle)^2$$

Where the linear measurements are y_i, $i \in [0, m]$, which is assumed to be generated by the model $y_i = \langle A_i, X^\star\rangle$, where $X^\star \in \mathbb{R}^{n\times n}$. Without constraints or a structural assumption on $\mathbf{X}^\star$, the problem is under-determined with infinite solutions. However, given a rank r matrix and a sufficiently large number of measurements ($\Omega(rn\log n)$), there is a unique solution that may be found via optimization.

Just as with compressed sensing, many variations on the matrix sensing problem have been studied. For instance, a common assumption is that the measurement matrices A are near-isometries (Recht et al., 2010), but non-RIP matrices have also been shown to work (Zhong et al., 2015). A special class of measurement matrix A_i (called Pauli's observable) is a Knonecker product of Pauli measurement operators, also carry RIP property. These matrices are of particular practical interest due to their importance in quantum state tomography (Liu, 2011). For simplicity, results are often presented for the simple case where $X^\star$ is positive semidefinite (PSD) and the measurements are noiseless, but the role of measurement noise is well-known for several optimization methods. As a whole, the matrix sensing field offers a mature set of tools and theoretical guarantees that are suitable for a wide range of applications.

3. Motivation

Since the measurements in the matrix sensing problem are assumed to be the output of a linear system that measures the hidden variable matrix X, many practical systems can be studied using matrix sensing. One such example is neuron activity recovery from μECoG electrode signals (Ajayi et al., 2018). When a set of neurons receives an input stimulus, the neurons respond with an electrical potential. Electrode recording techniques can capture aggregations over the set

of neurons for a short time period. If we construct a matrix by placing the response of neuron n at time t at index n, t, this experimental procedure yields valid matrix sensing measurements. For hours of experiments, the sample size m is in millions (Markram et al., 2015). The goal is to recover the membrane potential matrix $X \in \mathbb{R}^{n \times m}$, which is exactly the objective of the matrix sensing problem. In this context, the measurement matrices are determined by the low-pass filter delay and amplitude attenuation characteristics of the physical media in which the μECoG electrode operates. The μECoG recordings here can be viewed as the linear mapping of the membrane potential of the neurons.

Another direct application is in Quantum state tomography (QST). Here we report the density matrix $X^* \in \mathbb{R}^{2^q \times 2^q}$ of the quantum circuits, also called q-cubit state. The measurements are the expected value of q-cubit Pauli's observables $A_i \in \mathbb{C}^{2^q \times 2^q}$. The measurement vector $y \in \mathbb{R}^m$ is:

$$y = Tr(A_i.X^*) + e_i, \;\; i = 1, ...m$$

for some error e_i.

Currently the best computer in the world is using $q = 53$. This makes the size of X^*, very very very large! With these any brute force method (even existing matrix sensing methods) are very time consuming and not feasible. (Ajayi et al., 2018) have done matrix sensing experiment (using Factored Gradient Descent, section 6) for $q = 6$.

4. Preliminaries

In this section, we will briefly define some existing concepts and and tools that we will use.

4.1. Singular vector decomposition

For a rectangular matrix $X \in \mathbb{R}^{m \times n}$, the singular vector decomposition is defined as

$$X = U \Sigma V^T$$

where $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times n}$. The singular values are always positive. The rank of a matrix is equivalent to the number of nonzero singular values. For a square matrix, the time complexity of the full SVD is $O(n^3)$.

4.2. Matrix Norms

The **nuclear norm** of a matrix is represented by the summation of singular values.

$$\|X\|_* = \sum_{i=0}^{r} \sigma_i$$

Note that the nuclear norm is the L1 norm of the vector of singular values.

The **Frobenious norm** of a matrix X is represented by

$$\|X\|_F = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} X_{ij}^2}$$

Note that the Frobenius norm is also equal to the Euclidean, or L2, norm of the vector of singular values.

$$\|X\|_F = \sqrt{\langle X, X \rangle} = \sqrt{Trace(X^T X)} = \sqrt{\sum_{i=1}^{r} \sigma_i^2}$$

4.3. Positive Semi-Definite Matrix

A matrix M is positive semi-definite if an only if

$$x^T M x > 0 \;\; \forall x \in \mathbb{R}^n \setminus \mathbf{0}$$

It is important to note that symmetric real matrices are positive semi definite.

4.4. Restricted Isometry Property

We use the Restricted Isometry Property (RIP) definition from (Recht et al., 2010). A linear operator $A : \mathbb{R}^{n \times n} \rightarrow R^m$ satisfies the RIP on rank-r matrices, with parameter $\delta_r \in (0, 1)$, if the following holds for all rank-r X:

$$(1 - \delta_r)\|X\|_F^2 \leq \|A(X)\|_2^2 \leq (1 + \delta_r)\|X\|_F^2$$

Isometry is a property which says that the all the distances are preserved after the transformation by matrix A on X. Restricted isometry is relaxed version of isometry where the distances are preserved with an error of $(1 \pm \delta_r)\|X\|_F^2$. A measurement matrix is useful for the matrix sensing problem if it is a near-isometry (satisfies RIP).

4.5. Restricted Strong Convexity

A convex function f is (m, r)-restricted strongly convex (RSC) (Bhojanapalli et al., 2016a) if

$$f(Y) \geq f(X) + \langle \nabla f(X), Y - X \rangle + \frac{m}{2}\|Y - X\|_F^2$$

for any rank r matrices $X, Y \in \mathbb{R}^{n \times n}$. We can think of restricted strong convexity as being strongly convex, but only in some restricted set of directions. The set of directions is governed by rank r of matrix X and Y.

5. Projected Gradient Descent

The first provable solution to the matrix sensing problem came via a connection with compressed sensing, convex relaxation and projected gradient descent (Recht et al., 2010).

For a matrix function, the gradient is a matrix of gradients, one for the relationship between each cell in the input matrix and the function output. Gradient descent involves iteratively updating X_t with the gradient matrix.

$$X_{t+1} = X_t - \eta \nabla f(X)$$

The projected gradient method requires the extra step of projecting X_t onto the given constraint. The matrix sensing problem is a convex objective with the non-convex constraint that $\text{rank}(X) = r$. Although projected gradient descent works, it is quite computationally expensive as it requires computing the SVD for each iteration.

5.1. Iterative Hard Thresholding

One can project the iterate X_t onto a rank r space by computing the SVD at each gradient descent step and truncating to keep only the singular vectors with the r largest singular values. The gradient updates take the form:

$$X_{t+1} = \prod_{rank(X) \leq k} X_t - \eta \nabla f(X)$$

This procedure, while effective, requires that we calculate the SVD of X and sort the singular values for each iteration. While this is feasible for small matrices, we found in our experiments that the projection updates are no longer feasible on general-purpose hardware when $n > \approx 500$.

5.2. Convex Relaxation

Inspired by compressed sensing methods that solve a non-convex $L0$ norm constraint via relaxation onto a $L1$ constraint, (Recht et al., 2010) proposed a similar convex relaxation for the matrix sensing problem. Using the nuclear norm as a surrogate for sparsity in the singular values, projected gradient descent provably converges to the correct solution provided that the measurement matrices satisfy the RIP. Since the nuclear norm ball is a convex constraint, and is the best convex approximation of the rank function over the unit ball of matrices with norm less than one. The procedure, which we will refer to as nuclear norm minimization (NMM) updates X_t as follows:

$$X_{t+1} = \prod_{\|X\|_* \leq \lambda} (X_t - \eta \nabla f(X))$$

Formally, their result is that when the sensing matrices satisfy the RIP with $\delta_{2l} < 1$, $l \geq 1$, the solution to the matrix sensing objective is unique. Given a sufficiently strong RIP condition ($\delta_{5l} < 0.1$), gradient descent on the convex relaxation converges to the correct solution.

6. Factored Gradient Descent

Both methods for solving the matrix sensing problem are undesirable in practice because they require an SVD computation for each gradient descent step. The SVD requires $O(n^3)$ time for a matrix $X \in \mathbb{R}^{n \times n}$. While most matrix multiplication algorithms are also $O(n^3)$, the SVD calculation is non-trivial to parallelize and is substantially slower in practice. The boundary for computational infeasibility on a modern laptop is very easy to reach and is near $n \approx 500$ based on our experiments.

Algorithm 1 Factored gradient descent (FGD)

Input: Function f, target rank r, # iterations T.
Initialise $X = X^0$
Set $U \in \mathbb{R}^{p \times r}$ such that $X_0 = UU^\top$
Set step size η
for $t = 0$ **to** $T - 1$ **do**
  $U_{t+1} = U_t - \eta \nabla f(U_t U_t^\top)$
end for
Output: $X = UU^\top$

Another method to solve the matrix sensing problem is to encode the low rank constraint into the objective and perform unconstrained gradient descent on the reformulated problem. While this method has been used as a hueristic for a long time, the (Tu et al., 2015). This can be done by factorizing the matrix, $X = UV^\top$. Here, $U \in \mathbb{R}^{n \times r}$ and $V \in \mathbb{R}^{n \times r}$. For the rectangular matrix $X \in \mathbb{R}^{n_1 \times n_2}$, $U \in \mathbb{R}^{n_1 \times r}$ and $V \in \mathbb{R}^{n_2 \times r}$.

For the special case of a PSD matrix, it is much more advantageous to use the factorization $X = UU^\top$. With this factorization, the matrix sensing problem becomes

$$\min_{U \in \mathbb{R}^{n \times r}} \frac{1}{2} \sum_{i=1}^{m} f(X)^2$$

Where $f(X) = (y_i - \langle A_i, UU^T \rangle)$. The gradient descent step becomes:

$$U_{t+1} = U_t - \eta \nabla f(U_t U_t^T)$$

FGD eliminates the non-convex constraint and costly projection step, but induces problems by causing the factorization objective to become non-convex. It is easy to show that there are multiple non-connected solutions to the problem. For the factorization $X = UU^\top$, $UU^\top = UR^\top RU^\top$ is also a solution, where R is an orthonormal matrix: $R \in \mathbb{R}^{r \times r} : R^T R = I$. Therefore, the factorization transfers the complexity of the constraint into the optimization problem itself.

6.1. Convergence and Initialization

Despite this concerning fact, the factorized matrix approach is very successful in practice since the extra complexity for solving a harder optimization problem is not nearly as computationally expensive as the SVD for each iteration. In

terms of smoothness and convexity, the problem is determined by the measurement matrices $\mathcal{A}$. Under reasonable assumptions on $\mathcal{A}$, we have either an L-smooth objective with a sub-linear convergence rate or a RSC objective with a linear convergence (Bhojanapalli et al., 2016a). Specifically, with a step size

$$\eta = \frac{1}{16(M\|X^0\|_2 + \|\nabla f(X^0)\|_2)} \tag{1}$$

and a correctly-chosen initialization, one can converge with rate $O(1/T)$ (sublinear) given an M-smooth matrix sensing objective and rate $O(\alpha^T)$ with RSC. This is in contrast with the classic convex optimisation intialisation where $\eta \propto 1/M$, the step size depends on the spectral information of the matrix X.

Rather than express convergence in terms of RSC or conviteexity, we may also specify properties of the measurement mechanism since the two ideas are closely related. If **A** is RIP, this implies linear convergence because RIP implies that the objective obeys an equivalent RSC condition. However, it should be noted that the RIP condition is stronger than RSC condition. Since the Hessian of the objective is given by $A^\star A$, restricted strong convexity implies that:

$$\|A(Z)\|_2^2 \geq C\|Z\|_F^2$$

where $C > 0$ is small constant. This implies that the quadratic function has a defined lower bound. It is strongly convex over restricted set of directions given by Z. On the other hand, the RIP property is:

$$(1-\delta_r)\|X\|_F^2 \leq \|A(X)\|_2^2 \leq (1+\delta_r)\|X\|_F^2$$

Since the RIP has both an upper and lower bound and X is drawn from restricted set, RIP on the measurement matrices is stronger than a RSC assumption on the objective.

Regardless of the convergence properties, the initialization procedure becomes critical to finding a good solution now that the factorized objective is now non-convex. A variety of initialization schemes have been proposed. While random initialization is common for many optimization methods, it is not particularly well-suited to matrix sensing.

If we assume that FGD is initialized with good starting point (a point close to low rank solution), then the FGD converges (Bhojanapalli et al., 2016a).

- If f is M-smooth convex function and the initialisation U^0 is :

$$DIST(U^0, U_r^*) \leq \rho\sigma_r(U_r^*) \;\; \rho = \frac{\sigma_r(X^*)}{100\sigma_1(X^*)}$$

  Then the FGD converges sub-linearly-

$$f(X^T) - f(X_r^*) \leq \frac{\frac{5}{\eta}.DIST(U^0, U_r^*)^2}{T + \frac{5}{\eta}.\frac{DIST(U^0,U_r^*)^2}{f(X^0)-f(X_r^*)}} = O(1/T)$$

- If f is M-smooth and (m, r)-restricted convex function and the initialisation U^0 is :

$$DIST(U^0, U_r^*) \leq \rho'\sigma_r(U_r^*) \;\; \rho' = \frac{\sigma_r(X^*)}{100\kappa\sigma_1(X^*)}$$

  Then the FGD converges linearly-

$$DIST(U^0, U_r^*)^2 \leq \alpha.DIST(U^0, U_r^*)^2 + \beta\|X_* - X_r^*\|_F^2$$

Where the distance between any arbitrary matrix U and V, $DIST$ is defined as:

$$DIST(U, V) = \min_{R:R\in O} \|U - VR\|_F$$

O is the set of $r \times r$ orthonormal matrices R, such that $R^T R = I$

Note that the initialisation is spectral and based on true U_r^*, however it is easy to obtain, often with just one eigen value decomposition (Bhojanapalli et al., 2016a).

6.2. Regularization

Since U and V are non-unique, a common step in practice is to regularize the objective so that U and V are distinct from each other. This regularizer takes the following form.

$$g(U, V) = \lambda\lambda||U^\top U - V^\top V||_F^2$$

This form of regularization improves the convergence rate (Park et al., 2018) and is extensively used in practice, although there is some evidence that it may not be necessary when spectral initialization is used (Ma et al.). For a more complete discussion of this issue, see (Chi et al., 2019).

7. FGD with Momentum

The lower bounds of convergence for L smooth objective is $O(1/T^2)$ and for L smooth and strong convex objective is $O(c^{2T})$. Momentum based methods are proved to reach this lower bounds for convex function. However these methods also works really well in practice for non-convex optimisation settings.

Momentum can provably accelerate FGD over PSD matrices (Ajayi et al., 2018). By including previous estimates in the gradient update, one can obtain a faster linear convergence rate provided an initialization that is reasonably close to a local minimum and RIP measurement mechanisms. The size

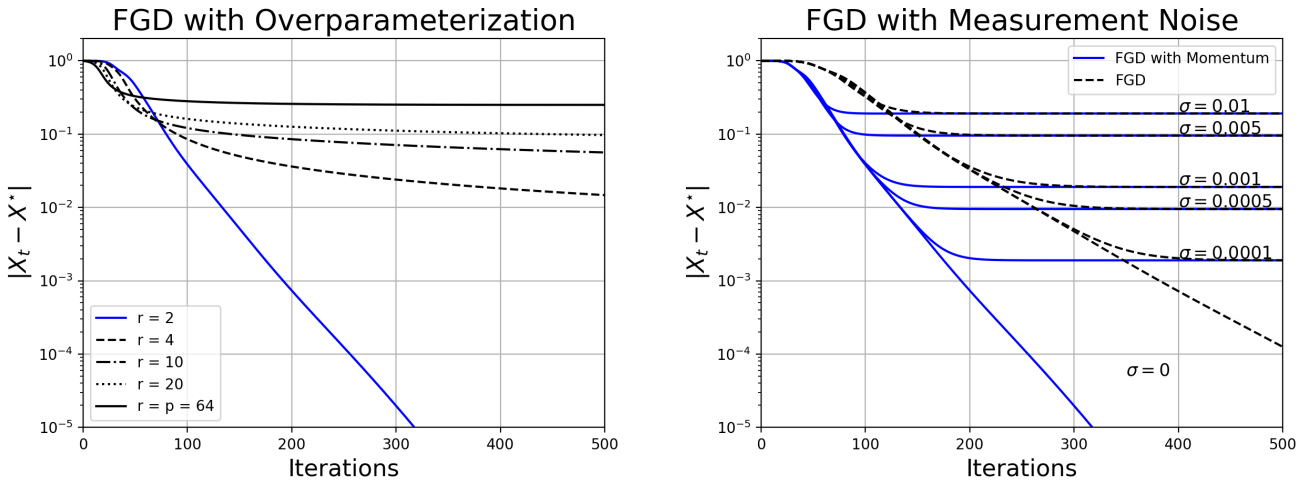


Figure 1. Individual effects of over-parameterization (left) and noise (right). The FGD variant used on the left includes momentum. The results without momentum (not shown) are essentially the same with a slightly slower rate of linear convergence.

of the valid initialization region is inversely proportional to the condition number of $X^\star$, so momentum is not an appropriate choice when the trailing singular values, which would otherwise be folded into the measurements as noise, are important for the application.

8. Noisy Measurements

When $X^\star$ is a PSD matrix, the matrix sensing problem is inherently easier because the non-convex objective behaves well. Any local solution to the objective function in the factorized space produces a result $X = X^\star$ (Bhojanapalli et al., 2016b). However, if we introduce noise to the measurements, the results change. In particular, the introduction of measurement noise causes the recovered solution X to be within a noise-dependent factor of the optimal solution $X^\star$. With the measurement model $y_i = \langle X^\star, A_i \rangle + \mathcal{N}(0, \sigma)$ with Gaussian noise, we have a bound on how far X is allowed to deviate from $X^\star$.

$$||UU^\top - X^\star||_F \leq \sqrt{\frac{\log n}{m}}\sigma$$

Here, n is the dimension of the problem and m is the number of measurements. Intuitively, we may think of the measurement noise process as adjusting the objective function. Such an adjustment produces sets of measurements that would correspond to noiseless measurements on non-PSD matrices. This introduces some weakly non-optimal local minima by distorting the matrix that gradient descent attempts to recover. To improve the recovery guarantees in the presence of noise, one may take more measurements or reduce the noise variance.

Algorithm 2 Factored gradient descent (FGD) with Momentum

Input: Function f, target rank r, # iterations T, rate η, momentum μ.
Initialise $X = X^0$
Set $U \in \mathbb{R}^{p \times r}$ such that $X_0 = UU^\top$
$Z_0 = U_0$
for $t = 0$ **to** $T - 1$ **do**
  $U_{t+1} = Z_t - \eta \nabla f(Z_t Z_t^\top)$
  $Z_{t+1} = U_{t+1} + \mu(U_{t+1} - U_t)$
end for
Output: $X = UU^\top$

9. Over-Parameterization

In many practical situations, the exact rank of $X^\star$ is not known. This is a problem for existing methods like FGD and SVD based IHT, since we must know r to construct a matrix. Fortunately, the $UU^\top$ factorization ($U \in \mathbb{R}^{n \times n}$) has an implicit regularizing effect when $\mathcal{A}$ is RIP. (Li et al., 2017) show that when properly initialized, FGD converges to the correct solution even when provided with a $U_0 \in \mathbb{R}^{n \times n}$. This implicit regularization is dependent on the initialization, since only some of the local minima in the overparameterized case lead to $X^\star$ and avoid overfitting. Orthogonal low-norm initialization of U is appropriate and comes with strong convergence guarantees in the overparameterized setting (Li et al., 2017). In practice, a good initialization is

$$U_0 = \alpha \tilde{I}$$

where $\tilde{I}$ is a truncated identity matrix and α is a small value ($\alpha < 1/\sqrt{r}$). For the general $UV^\top$ factorization, spectral initialization has weak guarantees (Bhojanapalli et al., 2016a) but is often sufficient in practice. Spectral

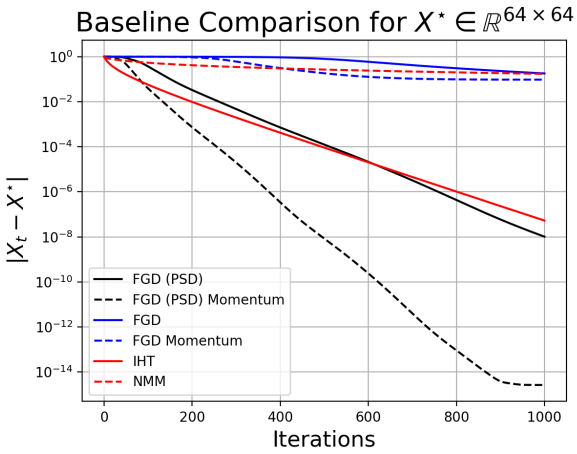


Figure 2. Comparison of all methods for the synthetic low-rank PSD matrix recovery problem.

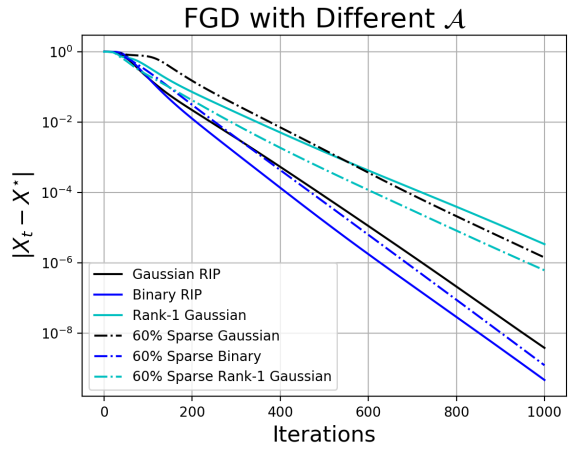


Figure 3. Comparison of several measurement matrices for the synthetic low-rank PSD recovery problem.

initialization requires computing a SVD, but only once at the start of the algorithm.

$$U_0 = X\Sigma^{1/2}, V_0 = Y\Sigma^{1/2}$$

where X and Y are the left and right singular vectors of $\nabla f(\mathbf{0})$.

By ensuring that U_0 has orthogonal columns and a small Frobenius norm, (Li et al., 2017) prove convergence to $\mathbf{X}^\star$ for PSD matrices in the overparameterized scenario.

10. Experiments

We performed experiments on real and synthetic data to supplement our discussion of the matrix sensing problem. We implemented the following methods:

1. **FGD with $UU^\top$:** Standard factored descent for PSD matrix recovery. We use small orthogonal initialization (low Frobenius norm with orthogonal columns).
2. **FGD with $UU^\top$ and momentum:** Accelerated descent method for PSD matrix recovery using momentum to incorporate previous estimates into the gradient update.
3. **FGD with $UV^\top$:** This is the general low-rank factorization for all matrices. We use the spectral method for initialization by computing the SVD of $-\nabla f(\mathbf{0})$.
4. **FGD with $UV^\top$ and momentum:** We applied momentum to the update procedure for both U and V to determine whether the $UV^\top$ factorization can also be accelerated.
5. **Iterative hard thresholding:** Each iteration, we use the SVD to project the iterate onto the space of low-rank matrices.
6. **Convex relaxation:** We minimize the nuclear norm subject to an L_1 constraint.

Our goal from the experiments is to understand the conditions and theoretical results from the matrix sensing literature in a practical context. Given a real-world matrix sensing problem, we would like to understand how to apply FGD methods and intuition from the theoretical results presented so far. We provide three sets of experiments.

10.1. Experiment Setup

First, we perform experiments on a random rank-2 PSD matrix in $\mathbb{R}^{64\times 64}$. For these experiments, we use $\eta = 0.1$ and $\mu = 0.5$ unless otherwise specified. We use 640 Gaussian RIP measurements and do not vary the number of measurements n. We run each method to convergence.

Our second set of experiments are for a sketch of the covariance matrix of the covertype dataset from UCI, which has 54 features. This covariance matrix is (approximately) low-rank, with a sharp drop-off after the first 10 singular values. We use matrix sensing to recover a rank-15 approximation of the covariance matrix from a set of 1620 Gaussian RIP measurements and display the results.

Finally, we want to understand how the measurement matrices affect the convergence and quality of the recovered solution. We implement three matrix sensing mechanisms: Gaussian RIP, binary (± 1) RIP, and the rank-1 Gaussian measurements from (Zhong et al., 2015). In practice, sparse random projections and measurement matrices are incredibly effective for many applications in databases and signal processing because they considerably reduce the computational burden without degrading the results (Achlioptas, 2001). Therefore, we also implement and test highly sparse

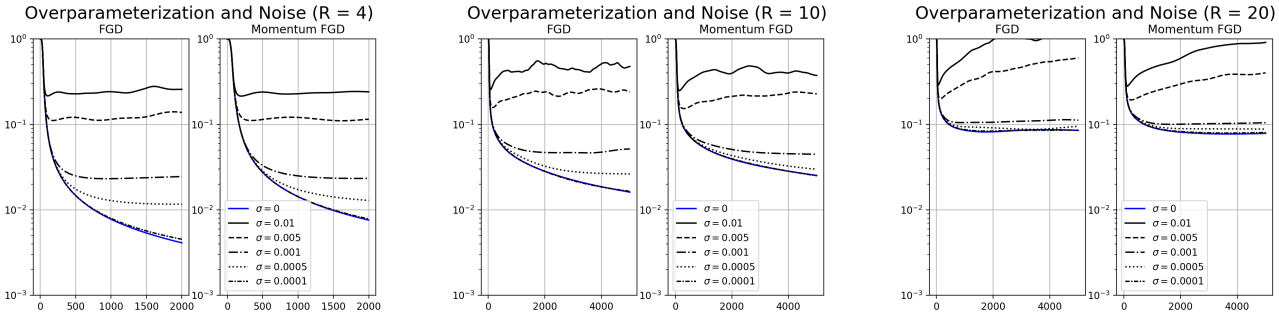


Figure 4. Effect of both overparameterization and noise on the PSD matrix sensing problem. We observe that overparameterized factorizations are less tolerant of measurement noise.

versions of the three measurement mechanisms described above to determine whether a similar idea holds true in matrix sensing. We use the same synthetic problem in $\mathbb{R}^{64\times 64}$ used previously, but this time we use ordinary FGD with $\eta = 0.8$, 1000 iterations and a variety of measurement matrices.

10.2. Results

Figure 2 shows the performance of all methods on the noiseless synthetic PSD recovery problem. We attempted to tune the step size and momentum parameter so that the $UV^\top$ factorization worked as well as $UU^\top$ but were unsuccessful, likely due to the fact that our implementation did not include the regularizer. This is possibly also due to the weak guarantees afforded by the spectral initialization method, in contrast with the strong linear convergence guarantees for the PSD case. As expected, accelerated FGD was substantially faster than ordinary FGD. However, depending on the value of the momentum parameter μ, the resulting solution is within a bounded error of the true solution. We chose $\mu = 0.5$ because it yields a quality solution, but we found that increasing μ quickly gives a solution with higher error.

Figure 1 shows the individual effects of overparameterization and noise on the matrix sensing problem. To obtain these results, we generated matrix sensing measurements and added Gaussian noise with variance σ, then performed the recovery. We observe results consistent with the theory. In the noiseless case, all local minima are global minimizers for the PSD problem. However, for noisy measurements the local minima may produce a result that deviates from the optimum by a constant value that increases with the noise variance.

For overparameterization (Figure 1), we observe that while overparameterized factorizations eventually converge to the correct solution, they do so at a much slower rate than when the rank is correctly specified. When we attempt overparameterized recovery on noisy measurements (Figure 4), we observe that it is critical in practice to have a reasonable idea of the rank of the matrix. The results in Figure 4 show that highly overparameterized factorizations are very sensitive to noise. We observed that with $R = p$, FGD actually diverges for noise levels that converge properly when $R << p$. Here R is the rank of the factorization.

When we attempted to recover a covariance matrix from a matrix sensing sketch, we found that it did a reasonably good job of preserving important information (Figure 5). However, matrix sensing tends to introduce some undesired patterns in the covariance matrix. Finally, we compared several different sensing mechanisms in Figure 3. We found that the efficient rank-1 measurements from (Zhong et al., 2015) were just as effective as Gaussian and ± 1 RIP matrices without needing as much storage space. We also explored the possibility of using sparse versions of each measurement matrix and found that FGD still converges to the correct solution. However, it seems to do so at a slower rate, where the slowdown is proportional to the sparsity in our projections.

10.3. Conclusion

Matrix sensing is an important problem for several application areas. We have presented a survey of matrix sensing methods and provided a detailed discussion of factorized gradient descent techniques. We also implemented a set of common methods and experimentally verified the theoretical results.

References

Achlioptas, D. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 274–281. ACM, 2001.

Ajayi, T., Mildebrath, D., Kyrillidis, A., Ubaru, S., Kollias, G., and Bouchard, K. Provably convergent acceleration in factored gradient descent with applications in matrix sensing. *arXiv preprint arXiv:1806.00534*, 2018.

Bhojanapalli, S., Kyrillidis, A., and Sanghavi, S. Drop-

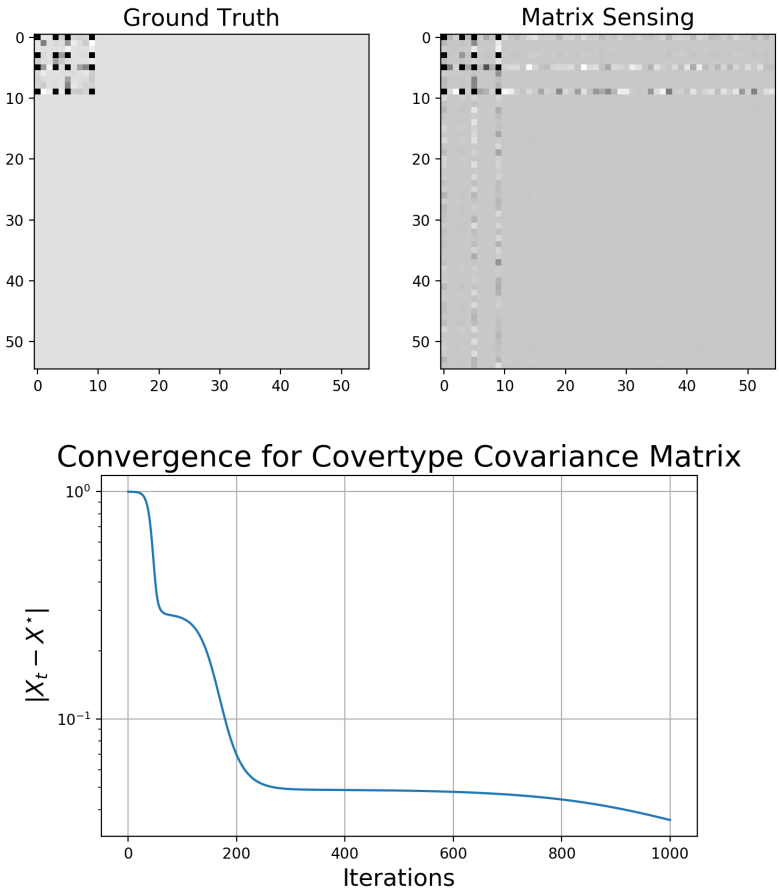


Figure 5. Convergence and recovered covariance matrix for the covertype dataset. We observe that the matrix sensing results preserve most of the important statistical relationships in the covariance matrix, but do introduce some spurious correlations.

ping convexity for faster semi-definite optimization. In *Conference on Learning Theory*, pp. 530–582, 2016a.

Bhojanapalli, S., Neyshabur, B., and Srebro, N. Global optimality of local search for low rank matrix recovery. In *Advances in Neural Information Processing Systems*, pp. 3873–3881, 2016b.

Chi, Y., Lu, Y. M., and Chen, Y. Nonconvex optimization meets low-rank matrix factorization: An overview. *IEEE Transactions on Signal Processing*, 67(20):5239–5269, 2019.

Li, Y., Ma, T., and Zhang, H. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. *arXiv preprint arXiv:1712.09203*, 2017.

Liu, Y.-K. Universal low-rank matrix recovery from pauli measurements. In *Advances in Neural Information Processing Systems*, pp. 1638–1646, 2011.

Ma, C., Li, Y., and Chi, Y. Beyond procrustes: Balancing-free gradient descent for asymmetric low-rank matrix sensing.

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., Ailamaki, A., Alonso-Nanclares, L., Antille, N., Arsever, S., et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

Park, D., Kyrillidis, A., Caramanis, C., and Sanghavi, S. Finding low-rank solutions via nonconvex matrix factorization, efficiently and provably. *SIAM Journal on Imaging Sciences*, 11(4):2165–2204, 2018.

Recht, B., Fazel, M., and Parrilo, P. A. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, 52(3):471–501, 2010.

Tu, S., Boczar, R., Simchowitz, M., Soltanolkotabi, M., and Recht, B. Low-rank solutions of linear matrix equations via procrustes flow. *arXiv preprint arXiv:1507.03566*, 2015.

Zhong, K., Jain, P., and Dhillon, I. S. Efficient matrix sensing using rank-1 gaussian measurements. In *International Conference on Algorithmic Learning Theory*, pp. 3–18. Springer, 2015.

An Overview of Optimization Methods in Game Theory

Senthil Rajasekaran [*1] **Nicolae Sapoval** [*1]

Abstract

Recent advances in machine learning have motivated active study of adversarial frameworks since the introduction of Generative Adversarial Networks (Goodfellow et al., 2014). As such, there has been a lot of interest in reinterpreting machine learning problems as game theoretic ones (Goodfellow et al., 2014; Oliehoek et al., 2017; Daskalakis et al., 2017; Ge et al., 2018; Zhang et al., 2019). The tools of game theory provide us with guarantees on the existence of optima (Nash, 1950) in game settings and admit different optimization dynamics. In order to address those unique constraints several methods of optimization have been adopted to the game theoretic setting. In our work we analyze these approaches both qualitatively and quantitatively. We describe the algorithms, provide theoretical guarantees where they have been proven to exist, and evaluate the algorithms with respect to training GANs.

1. Introduction

In this project, we will be considering optimization techniques as they apply to the game theoretic notion of solvability. This area of study has seen a lot of interest in recent years with the development of adversarial learning frameworks, in particular the generative adversarial networks (GANs) (Goodfellow et al., 2014; Hu, 2019; Daskalakis et al., 2017; Oliehoek et al., 2017; Ge et al., 2018; Zhang et al., 2019). In an adversarial network, several agents, represented by neural networks, are placed into a game like scenario in order to reach a desirable objective through iterated play.

In game theory the notion of solvability is given by the Nash equilibrium (Osborne & Rubinstein, 1994). We informally define a Nash equilibrium as a state induced by a strategy profile (a set of functions that recommends actions to an agent based on observed history) in which no agent can improve their payoff by unilaterally deviating from the strategy profile, i.e. every agent, upon knowing that the other agents will follow the Nash equilibrium actions will do no better than following the Nash equilibrium actions themselves.

In a game with a finite number of agents and a finite number of pure strategies for each agent, the existence of a mixed strategy Nash equilibrium is guaranteed by Nash's Existence theorem (Osborne & Rubinstein, 1994). A pure strategy is a function that takes the game history as input and outputs a single action. A mixed strategy takes the game history as input and returns a probability distribution over a set of possible actions. This is a very strong existence guarantee, which, in turn, under permissive conditions allows an optimal solution to exist in the GAN training setting (Goodfellow et al., 2014).

We motivate our analysis of game theoretic questions from the optimization lens by drawing a parallel between Nash equilibria and minima in objective functions in a classical optimization setting. In a game, under certain permissible conditions, best response play (optimal response to observed history) will converge to Nash equilibrium (Osborne & Rubinstein, 1994). In this way a parallel can be drawn to algorithms like gradient descent (GD) that will converge to a minimum under certain assumptions on the objective function and constraint set.

We want to examine what restrictions on the constraint set are induced by the formulation of a problem as a game. Hence, by using game theoretic algorithms and the concept of Nash equilibria, what convergence guarantees can we provide for those problems? Can we improve the existing techniques in optimization theory by using tools from game theory?

In order to reason about the links between game theory and optimization, we believe it will be helpful to phrase the setting induced by a game through the use of well known terms in optimization theory. As discussed above, we know that, given a game and its payoff functions, we are guaranteed the existence of at least one Nash equilibria in any reasonable setting. Furthermore, because the final output will be a probability vector representing the weights with which each

[*]Equal contribution [1]Department of Computer Science, Rice University, Houston, Texas. Correspondence to: Senthil Rajasekara <sr79@rice.edu>, Nicolae Sapoval <nsapoval@rice.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

pure strategy is considered, this can be viewed as doing optimization over an n-dimensional simplex - a convex set. This is the motivation behind the use of algorithms like mirror descent in the game theory setting. (Rakhlin & Sridharan, 2013)

The rest of paper will be following the layout described below. In section 2 we will give background material and some motivation for the algorithms to follow. In section 3 we will describe algorithms proposed by different authors to solve the game theoretic optimization problems. Next, in section 4 we will compare the proposed algorithms on a synthetic dataset to evaluate the effectiveness and trade-offs of each approach. Finally, in section 5 we will discuss the observed results and outline some possible directions for future work in this area.

2. Background

In this section we will briefly introduce the key concepts in game theory and optimization. We then will follow up with the problem setup in the optimization context in the next section. In the game theory subsection we define the notion of Nash equilibrium and state a corresponding existence theorem. In the optimization section we will focus on defining the gradient descent algorithm and motivating using it as a foundation for game theoretic problems.

2.1. Game Theory

Although the definition of a *game* is often tweaked to fit the specific framework it is being interpreted in, we will first start by providing a general definition of a game and explicitly state when we add other assumptions.

Definition 2.1 (Game (Osborne & Rubinstein, 1994))**.** A *game* is a quadruple

$$\langle [n], (S_1, S_2, \dots S_n), O, f \rangle$$

- $[n] = 1, 2, \dots, n$ represents the set of *players*, and $n \geq 2$
- $(S_1, S_2, \dots S_n)$ are the sets of *strategies*. Set S_i corresponds to the strategies available to player i. Furthermore we define $S = S_1 \times S_2 \dots \times S_n$ as the set of *strategy profiles*.
- O is the set of *outcomes*.
- $f : S \to O$ is a function that assigns every strategy profile an outcome.

The definition may seem opaque at first glance, but the formulation becomes much more natural when view in the context of an example.

Example. Let us try to formulate a very simple game in terms of the definition above. The game "Odds and Evens" is played by two players. Both players will simultaneously choose to play either an odd number or an even one. Player 1 wins when the sum of both numbers is odd, and Player 2 wins otherwise. The game will be played for two rounds.

- $n = 2$, since we have a two player game
- Let *odd* denote playing an odd number and *even* an even one. Then, $S_1 = S_2 = \{odd, even\} \times \{odd, even\}$. This represents the two distinct choices that can be made at each round.
- Let 1 denote a player 1 win and 2 a player 2 win. Then the set of outcomes O can be viewed as $\{1, 2\} \times \{1, 2\}$ and it represents the four possible outcomes of two rounds of the game.
- $f : S \to O$ maps each strategy to an outcome. So if player 1 uses the strategy $\{odd, odd\}$ and player 2 uses the strategy $\{even, even\}$, the function f will map this strategy profile to $(1, 1)$, as player 1 will win both rounds. The function f is defined on all other inputs from S in the same manner.

Note, that while the above formal definition of the game is required to prove the important results that follow, we often will skip the explicit definition of the parameters for the games in the following sections.

In the definition above we have only considered "pure strategies", which are deterministic recommendations of actions. A more general formulation (and one that we will need to make a guarantee for existence of a Nash equilibria, which will be defined later) is one of a *mixed strategy*.

Definition 2.2 (Mixed Strategy (Osborne & Rubinstein, 1994))**.** A *mixed strategy* of a player i is a probability distribution over the set S_i. We denote the set of mixed strategies for player i as S_i^m. A *mixed strategy profile* S^m is an element of $S^m = S_1^m \times S_2^m \dots \times S_n^m$.

Note, that even in case of games with finite number of players and pure strategies, the set of mixed strategies is infinite. This allows for much more expressive behavior of games in mixed strategies.

Finally, we need the concept of a utility function before introducing the Nash equilibrium.

Definition 2.3 (Utility (Osborne & Rubinstein, 1994))**.** A *utility function* for a player i is given by $U_i : O \to \mathbb{R}$, a function that maps each game outcome to a real number.

Player's preferences can be measured through the use of a utility function, and this is in turn will allow us to reason about what outcomes are preferred for each player. Utility

functions can be viewed as negative loss in case of machine learning. Thus each agent will seek to maximize their utility or equivalently minimize the loss.

Definition 2.4 (Nash Equilibrium (Osborne & Rubinstein, 1994))**.** A mixed strategy profile s^* in an n player game is a *Nash equilibrium* if

$$\forall(i \in [n])\forall(s \in S_i^m)U_i(f(s^*[i \mapsto s])) \leq U_i(f(s^*))$$

where $s^*[i \mapsto s]$ represents the strategy profile with the assignment for i rewritten to s.

Remark. Intuitively, the Nash equilibrium can be seen as a "no single deviance" condition. That is, given a strategy profile s^*, it is the case that for every single player, they can't unilaterally improve their own payoff by deviating from the Nash equilibrium profile. In this way, a Nash equilibrium has a certain sense of "stability" in that rational players will not attempt to individually deviate from the strategy profile.

Stronger notions of equilibrium can be defined in which we look at the notions of k players all deviating at once to improve all of their payoffs (Osborne & Rubinstein, 1994). However, these "coalition" dynamics are not within the scope of our current work given that we are mostly focusing on the two player GAN training setting in this document.

Theorem 2.1 (Existence of Nash Equilibrium). Every finite game with a finite number of players and finite number of pure strategies associated with those players has at least one Nash equilibrium(Nash, 1950).

2.2. Optimization

Many of the algorithms we will present in the methods section use the gradient descent algorithm as a foundation. As such, this section will center around introducing the gradient descent algorithm.

Gradient descent is an iterative, first order optimization algorithm used to find a minima of an objective function $f(x)$. The iterations are given by the following iteration rule.

$$x_{t+1} = x_t - \eta_t \nabla f(x_t)$$

Here, η_t represents a step size (which may vary over index) and ∇ represents the gradient operator. At each iteration, the approximation x_t is updated by moving in the opposite direction of the gradient. Intuitively, this can be seen as "sliding down the slopes" of the function until a minimum point is reached. Gradient descent can be seen at stopping at "stable" points in which the gradient is 0 and the algorithm makes no further progress.

In the optimization setting, two common restraints on objective functions are used to guarantee the existence of global minima.

Definition 2.5 (Convex Function)**.** A function is *convex* if

$$\forall(x, y)f(x) \geq f(y) + \langle \nabla f(y), y - x \rangle$$

where $\langle x, y \rangle$ represents the inner product of x and y and ∇ represents the gradient operator.

Definition 2.6 (Strongly Convex Function)**.** A function is μ*-strongly convex* (with respect to the Euclidean norm) if

$$\forall(x, y)f(x) \geq f(y) + \langle \nabla f(y), y - x \rangle + \frac{\mu}{2} \|x - y\|_2^2$$

In the same way that convex optimization uses the global minima guarantees of these restrictions, algorithms in the game theory setting have the guarantee of a Nash equilibrium. In the game setting, using the gradient to move in a direction has the analogous concept of best response play (in which a player moves in a way that best responds to the observed history) which similarly makes no further progress at Nash equilibrium points. This can be intuitively seen by unpacking the definition of a Nash equilibrium. If it were the case that a player had a best response that deviated away from the Nash equilibrium, this would contradict the no deviance property of Nash equilibria.

In the next section we will discuss several algorithms designed to find such equilibria given this existential guarantee.

A popular modification of gradient descent is Nesterov's acceleration, given by the following updated iteration rule.

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) + \beta(x_t - x_{t-1})$$

(Nesterov, 1983) The term $\beta(x_t - x_{t-1})$ is the momentum step amplified by a hyperparameter β, which biases the direction of the algorithm towards the previously taken direction. The acceleration exhibits faster convergence for convex functions (Nesterov, 1983) and the algorithm is considered to be one of the most important ideas in the theory of optimization.

3. Methods

We present several optimization algorithms that attempt to solve the problem of finding Nash equilibria in games.

3.1. Stochastic gradient descent

Stochastic gradient descent is a modification of gradient descent originally introduced to deal with very large data sets. It updates the original iteration rule to the following for an n dimensional x vector:

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \longrightarrow x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t), i_t \in [n]$$

At each step, the algorithm uniformly chooses a sample from the set $[n] = \{1, 2, \dots n\}$ It then performs the gradient descent update by only considering that coordinate. In this way full gradients need not be computed, which may be expensive when considering high values of n.

In practice, typically instead of choosing one sample uniformly from $[n]$, a small subset of samples called a batch is chosen. Furthermore, the random selection procedure is replaced with cycling through all batches in sequence.

In a game setting, stochastic gradient descent could be used by having each player minimize their loss function based on the stochastic gradient descent update rule. We know of no convergence guarantees for stochastic gradient descent in either the general game setting or the GAN training setting.

3.2. Mirror descent

Mirror descent is a modification of gradient descent based on the observation that the L_1 norm is a much more suitable norm for the n dimensional simplex than the L_2 Euclidean norm we implicitly use in other algorithms. Since the game theory optimization problem is one of determining probabilities, it is one that utilizes the geometry of the simplex as well. Therefore techniques utilizing the L_1 norm are an emerging point of interest in the intersection of game theory and optimization.

Mirror descent makes use of changing the notion of distance (away from Euclidean) by introducing *Bregman divergence*

Definition 3.1 (Bregman Divergence(Bubeck, 2014))**.** Given a strongly convex function $f : D \to \mathbb{R}$ with $D \subseteq \mathbb{R}^n$ a convex set, the *Bregman divergence* $D_f : D \times D \to \mathbb{R}_+$ is given by

$$D_f(x, y) = f(x) - (f(y) + \langle \nabla f(y), x - y \rangle)$$

In the L_1 norm, the negative entropy function

$$E(x) = \sum_{i=1}^{n} x_i \log x_i$$

is a strongly convex function over the set $D = \mathbb{R}^n_+$. Under these conditions, it can also serve as a *mirror map*, which we will now define.

Definition 3.2 (Mirror Map(Bubeck, 2014))**.** Let $D \subseteq \mathbb{R}^n$ be an open convex set and let f be a strongly convex function. Then f is a *mirror map* if it additionally satisfies the following conditions

- f is differentiable on D
- The range of ∇f is the entire $\mathbb{R}^n$
- As x approaches δD , the boundary of D, $\nabla f(x) \to \infty$

Furthermore, we define the notion of a *subgradient of g at a point*

Definition 3.3 (Subgradient(Bubeck, 2014))**.** If $U \subseteq \mathbb{R}^n$ is an open convex set and $g : U \to \mathbb{R}$ is convex on U, a vector $v \in \mathbb{R}^n$ is called a *subgradient of g at the point* x_0 if

$$g(x) - g(x_0) \geq \langle v, x - x_0 \rangle$$

The collection of subgradients of the function g at the point x_0 will be denoted $\delta g(x_0)$.

The *mirror descent* algorithm then has the following update rule for a mirror map f and a compact set C over the objective function g. Initially we choose

$$x_0 = \arg\min_{x \in C} f(x)$$

And we iterate by updating

$$x_{t+1} = \arg\min_{x \in C} D_f(x, x_t) + \eta \langle v_t, x \rangle$$

where $\{v_1, \dots v_t \dots\}$ represent a sequence of directions such that each $v_t \in \delta g(x_t)$ (Bubeck, 2014).

Under the conditions we have outlined, the negative entropy function qualifies as a mirror map using the L_1 norm and over the n dimensional simplex. Using this function as the mirror map is a popular way to use mirror descent when minimizing an objective function over the probability simplex. In a machine learning setting, mirror descent would be used to minimize each player's loss function based on the mirror descent update rule. We know of no convergence guarantees for this algorithm for either the general game setting or the GAN training setting.

3.3. Optimistic gradient descent

Optimistic gradient descent was introduced in (Daskalakis et al., 2017) in order to specifically reason about the GAN training framework, specifically the WGAN framework. In the GAN framework we consider 2 player, zero sum games explicitly, and the optimistic gradient descent algorithm takes this into account explicitly.

In a two player game setting, one could imagine using gradient descent on a loss function to optimize each player's actions. In the WGAN framework, the loss function is given by

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

where the two players are represented by G (the generator, who controls θ) and D(the discriminator, who controls ω). Furthermore Q represents the true distribution (which we assume we have access to in this particular case) that the generator is trying to mimic. The game is zero sum, so the generator is given this loss function and the discriminator is given the loss function $-L(\theta, \omega)$.

In this scenario, one could define a gradient descent based algorithm as follows. Let $\nabla_{\omega,t} = \nabla_\omega L(\theta_t, \omega_t)$ and $\nabla_{\theta,t} = \nabla_\theta L(\theta_t, \omega_t)$. Then we could naturally define the following update rules

$$\omega_{t+1} = \omega_t + \eta \nabla_{\omega,t}$$
$$\theta t + 1 = \theta - \eta \nabla_{\theta,t}$$

The idea behind optimistic gradient descent is to tweak the update rules to read as follows

$$\omega_{t+1} = \omega_t + 2\eta \nabla_{\omega,t} - \eta \nabla_{\omega,t-1}$$
$$\theta t + 1 = \theta - 2\eta \nabla_{\theta,t} + \eta \nabla_{\theta,t-1}$$

Optimistic gradient descent exhibits a last iterate convergence rate of $O(\frac{1}{T})$ in the WGAN training framework, where T represents the number of iterations (Daskalakis et al., 2017).

3.4. Optimistic ADAM

Optimistic ADAM was also introduced in (Daskalakis et al., 2017) in make a comparison between the well known ADAM algorithm and optimistic gradient descent. Given a step size η, decay rates for moment estimates β_1 and β_2, a loss function of weights $l_t(\theta)$ and initial parameters θ_0, the iteration step of optimistic ADAM is given by the following step.

First the stochastic gradient is computed

$$\nabla_{\theta,t} = \nabla_\theta l_t(\theta)$$

Then the biased estimates of the moments are calculated.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta,t}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla^2_{\theta,t}$$

Then the bias corrected moments are computed.

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$
$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

And then the optimistic gradient step is performed, which mirrors the form of the optimistic gradient descent algorithm.

$$\theta_t = \theta_{t-1} - 2\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \eta \frac{\hat{m}_{t-1}}{\sqrt{\hat{v}_{t-1}} + \epsilon}$$

Note that unlike optimistic gradient descent, this algorithm is just a general modification of gradient descent. Therefore it could be used in any scenario where gradient descent might used, and has no explicit ties to GANs or games. In order to use it in the context of training GANs, we would just update our parameters θ and ω using this update rule instead of simple gradient steps.

We know of no convergence guarantees for this algorithm for either the general game setting or the GAN training setting.

3.5. Hamiltonian dynamics (Symplectic gradient adjustment)

The symplectic gradient adjustment was introduced in (Balduzzi et al., 2018) and introduces it's own formalism. This technique provides a way to improve convergence speed and guarantees when multiple interacting objectives are considered, such as a game. Gradient descent runs into problems in this setting, such as entering "orbits" in which no further progress can be made. The purpose of this section is to simply briefly introduce the algorithm and its implementation without a discussion of its motivation. If the reader is interested in the motivation, (Balduzzi et al., 2018) covers this in good detail.

For this section, we redefine the notion of a game to match the definition in (Balduzzi et al., 2018).

Definition 3.4 (Game)**.** A game is a set of players $[n] = \{1, 2, \dots n\}$ and twice continuously differentiable losses $\{l_i : \mathbb{R}^d \to \mathbb{R}\}_{i=1}^n$. Parameters are $w = (w_1, \dots w_n) \in \mathbb{R}^d$ with $w_i \in \mathbb{R}^{d_i}$ where $\sum_{i=1}^n d_i = d$. Player i controls w_i

Although this definition is largely similar to the one we have introduced before, it introduces some important additional constraints, like specifying that the players actions correspond to setting values for the vector parameters and requiring that the outcome functions be twice differentiable. However, it should still be noted that this formalism is still extremely general and allows for a wide variety of scenarios to be modelled effectively.

Given a game, we define the *simultaneous gradient*

Definition 3.5 (Simultaneous Gradient)**.** Given a game as defined above, we define the *simultaneous gradient* to be

$$\xi(w) = (\nabla_{w_1} l_1 \dots \nabla_{w_n} l_n) \in \mathbb{R}^d$$

where notation follows the definition for games presented previously.

The simultaneous gradient is a small tweak on the original concept of a gradient with an eye towards the fact that certain players control specific parameters. Given this new concept of gradient, it now becomes possible to define the concept of a *game Hessian* that will mimic the structure of the well known Hessian.

Definition 3.6 (Game Hessian)**.** The *Hessian* of a game is a $d \times d$ matrix of second derivatives (d is the dimension from previous definitions) It is defined as

$$H = \begin{bmatrix} \nabla^2_{w_1} l_1 & \nabla^2_{w_1,w_2} l_1 & \dots & \nabla^2_{w_1,w_n} l_1 \\ \dots & \dots & \dots & \dots \\ \nabla^2_{w_n,w_1} l_n & \nabla^2_{w_n,w_2} l_n & \dots & \nabla^2_{w_n,w_n} l_n \end{bmatrix}$$

And again, we can see this as a slight modification of the original Hessian concept with an eye on which players own which parameters. Given a Hessian matrix, it is known that it decomposes into the sum of a symmetric and an anti-symmetric component. Let us denote the symmetric component as $S(w)$ and the antisymmetric component as $A(w)$.

Definition 3.7 (Game Classifications)**.** A game is a *potential game* if $A(w) \equiv 0$. It is a *Hamiltonian game* if $S(w) \equiv 0$.

Potential games are easy to solve, and so the paper devotes time towards the Hamiltonian case. It is for this reason they introduce the symplectic gradient adjustment.

$$\xi_\lambda := \xi + \lambda A^\top \xi$$

Where notation follows once again from previous definitions (though parameters were removed). The idea behind this algorithm is that we precompute this ξ_λ value and use it as if it were the gradient for another gradient descent algorithm. This is not an iterative algorithm. It is a precomputed adjustment that modifies the concept of a gradient in order to boost convergence in game settings for other algorithms like gradient descent.

The symplectic gradient descent algorithm's main advantage is it's more robust convergence in multiplayer settings,meaning that it converges by avoiding problems common to other similar algorithms, like gradient descent. A discussion of these cases as well as convergence rates under a few varying restrictive constraints is given in (Balduzzi et al., 2018).

4. Results

To evaluate the proposed algorithms we have used the synthetic dataset from (Metz et al.), i.e. a mixture of 16 Gaussians placed on a 4×4 grid. We chose this dataset since several of the papers we considered (Balduzzi et al., 2018; Daskalakis et al., 2017; Mertikopoulos et al., 2018) have claimed the results and ran comparisons using this dataset. We have used a total of $16 \times 5,000 = 80,000$ points (each Gaussian being equally represented) in $\mathbb{R}^2$ as our ground truth for training of the GANs. For all of the algorithms we have used batches of size 16×500 for each iteration of the algorithm. In addition, to allow for reproducible results we have fixed random seeds for both NumPy and PyTorch during the experiments. We have used the learning rate of 0.001 and 0.002 for gradient descent, 0.0002 for RMSprop, 0.0002 for Adam (with $\beta_1 = 0.5$ and $\beta_2 = 0.999$). For the optimistic versions of these algorithms we have used the same learning rates as for the original ones.

In figure 1 we see the comparison over the first 8000 iterations between the regular and optimistic versions of the algorithms discussed. We stopped these experiments after 8000 iterations since both optimistic Adam and optimistic RMSprop (bottom 2 rows in figure 1) experienced a discriminator collapse prior to reaching the 8000th iteration. Namely, the generator error became 0, and neither generator nor discriminator could learn any more. Out of all candidate algorithms only regular Adam starts converging towards the proper distribution while the other algorithms struggle to make progress.

To assess whether this is just an issue of the rate of convergence rather than non-convergence we have let gradient descent, as well as the optimistic algorithms run for more iterations. However, as can be seen from the figure 2, even after 20,000 iterations neither gradient descent, optimistic gradient descent nor optimistic Adam algorithms (top 3 rows) converge.

5. Discussion and future work

As with any emerging field, many new algorithms have appeared up to address the problem of optimization in game settings, so many so that we could not review them all here. One of the aims of this document was to gather several of these algorithms in one place and test them in as best an unbiased manner as we can manage. However, we have realized that in many cases while the theoretical foundations of the algorithms proposed were solid, no ready-to-use implementations were available. To this extent we have implemented optimistic optimizers in PyTorch and used them in our tests.

As we have seen from the results, most of the algorithms did not achieve convergence in the setting they were tested in. We suspect that conducting a through grid-search on the space of hyper-parameters could alleviate this problem. However, that approach would be somewhat contrary to the

out-of-the-box unbiased testing we were aiming for in this review. Hence, we suggest that a more thorough evaluation of these algorithms should be conducted while controlling for a wide range of hyper-parameters.

Finally, in this document we have only considered the cases of two player games. However, the framework provided in section 2 is rich enough to formulate the question of dynamics in n-player games. The two player setting gives rise to adversarial problems, therefore enriching the set of methods that can be applied and the class of problems that can be addressed. Games with more than two players introduce a make-shift coalition based dynamics in additional to adversarial principles. It is not yet well understood what optimization problems can benefit from these approaches, and in general the introduction of coalition dynamics provides a spike in complexity that may not lead to tractable algorithms. As an example, in the relatively simpler setting of modal logics, adding coalition dynamics makes the satisfiability problem PSPACE-complete as opposed to NP-complete(Pauly, 2002). However, it would be interesting to analyze what kind of machine learning problems have a natural - or even useful - formulation as an n-player game for $n > 2$.

References

Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T. The mechanics of n-player differentiable games. 2018.

Bubeck, S. Convex Optimization: Algorithms and Complexity. *arXiv e-prints*, art. arXiv:1405.4980, May 2014.

Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H. Training GANs with optimism, 2017.

Ge, H., Xia, Y., Chen, X., Berry, R., and Wu, Y. Fictitious GAN: Training GANs with historical models, 2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NeurIPS'14*, pp. 2672–2680, 2014.

Hu, R. Deep fictitious play for stochastic differential games, 2019.

Mertikopoulos, P., Lecouat, B., Zenati, H., Foo, C.-S., Chandrasekhar, V., and Piliouras, G. Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile, 2018.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. Unrolled generative adversarial networks (2016). *arXiv preprint arXiv:1611.02163*.

Nash, J. F. Equilibrium points in n-person games. *PNAS*, 36(1):48–49, 1950. doi:10.1073/pnas.36.1.48.

Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Doklady AN USSR*, 269:543–547, 1983. URL `https://ci.nii.ac.jp/naid/20001173129/en/`.

Oliehoek, F. A., Savani, R., Gallego-Posada, J., van der Pol, E., de Jong, E. D., and Gross, R. GANGs: Generative adversarial network games, 2017.

Osborne, M. J. and Rubinstein, A. *A Course in Game Theory*. The MIT Press, 1994. URL `https://ideas.repec.org/b/mtp/titles/0262650401.html`.

Pauly, M. A modal logic for coalitional power in games. *J. Logic and Comput.*, 12:149–166, 2002. doi:10.1093/logcom/12.1.149.

Rakhlin, A. and Sridharan, K. Optimization, learning, and games with predictable sequences, 2013.

Zhang, L., Wang, W., Li, S., and Pan, G. Monte Carlo neural fictitious self-play: Approach to approximate Nash equilibrium of imperfect-information games, 2019.

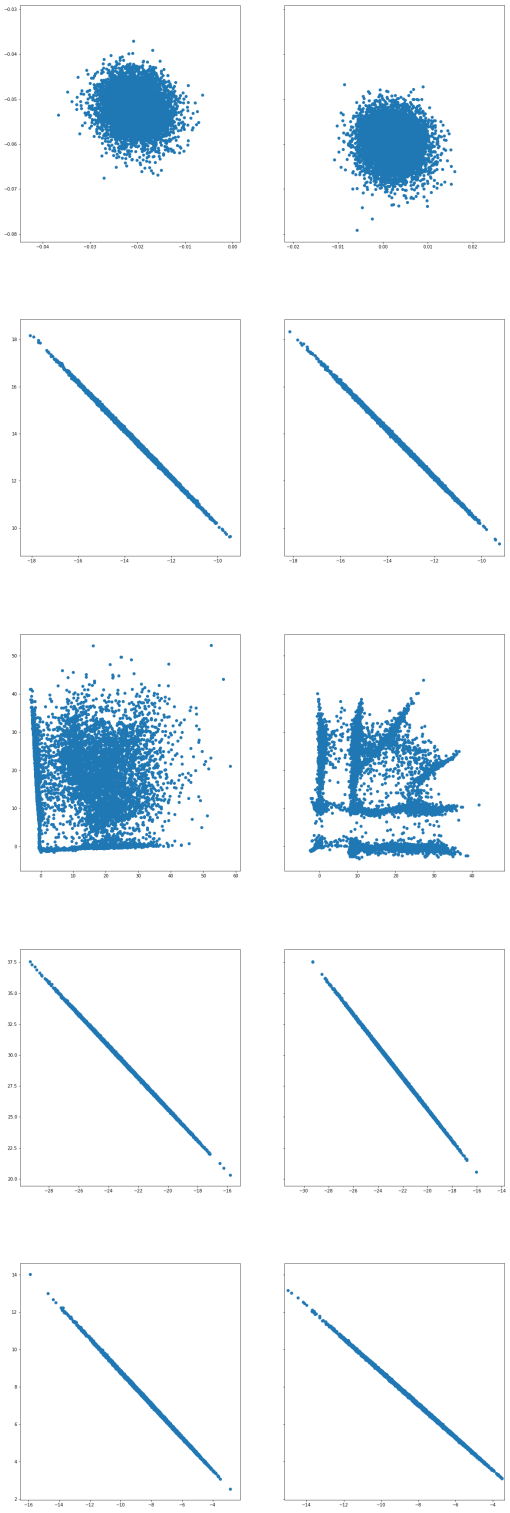


Figure 1. Comparison of gradient descent, RMSprop, Adam, optimistic RMSprop and optimistic Adam (from top to bottom) algorithms at 4000 and 8000 iterations.

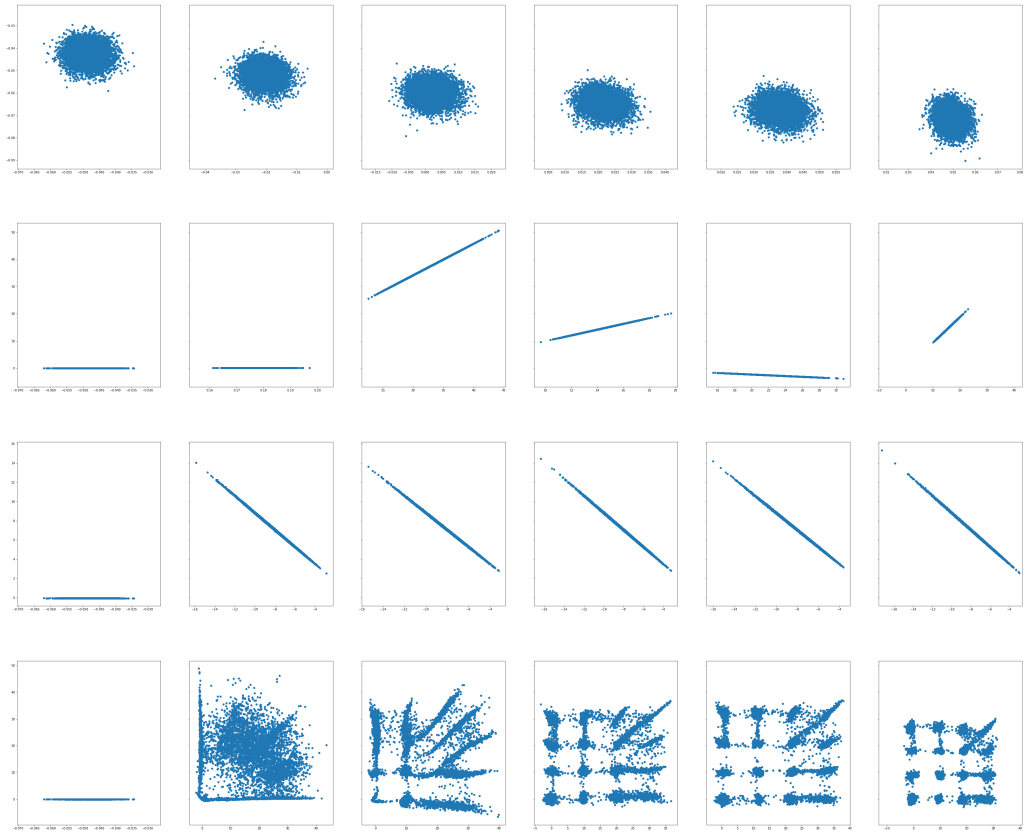


Figure 2. Comparison of gradient descent, RMSprop, Adam, optimistic RMSprop and optimistic Adam (from top to bottom) algorithms at 4000 and 8000 iterations.

Proximal + Learning Methods for Image Inverse Problems

Mary Jin [*1] **Tianyi Zhang** [*1]

Abstract

Proximal methods have been widely used in image processing and reconstruction tasks due to their modularity and efficiency. Traditionally, hand-designed priors are used in these frameworks to regularize underdetermined problems. Recently, deep learning has become state-of-the-art for many image processing tasks due to their ability to learn more accurate distributions given a set of training images. We are interested in the combination of these two approaches. In this report, we review two major classes of such techniques: unrolling and learning proximal operators, in both application and theoretical aspects. We discuss how these techniques can combine the advantages of proximal and deep learning to offer higher efficiency or reconstruction performance.

1. Introduction

Linear inverse problem is pervasive in restoration of high resolution images in applications from natural image super-resolution to CT and MRI reconstruction. The majority of these tasks can be modeled by the linear system

$$y = Ax + \omega \tag{1}$$

Given measurement $y \in \mathbb{R}^M$ and the sensing matrix $A \in \mathbb{R}^{M \times N}$, one seeks to recover the true signal $x \in \mathbb{R}^N$ stripped of the noise $\omega \in \mathbb{R}^M$. For example, in superresolution, A downsamples the high resolution x and returns the noisy, low resolution y.

However, in most real life situations, $M \ll N$ and the inverse problem is underdetermined with non-trivial null space. Thus prior knowledge of x must be used to narrow down possible solutions. There are a wealth of existing literature on inverting ill-posed imaging problems, most lie on a spectrum between hand-crafted priors and entirely data-driven recovery.

1.1. Hand-designed recovery methods

Traditionally, the prior on the true signal x are either hand-crafted or analytically derived based on empirical observations. Popular imaging priors include l_1-regularization, sparsity in wavelet coefficient, and total variation regularization.

For a given image prior, the inverse problem can be formulated as below:

$$\min_x f(x) + \phi(x; \lambda) \tag{2}$$

where ϕ is the prior function and λ is its parameters.

Since ϕ is often not differentiable, it is common to adapt the proximal gradient algorithm with proximal operator $\mathcal{P}$ and step size η to solve (2) iteratively:

$$x_{t+1} = \mathcal{P}_{\phi_\lambda, \eta}(x_t - \eta_t A^\top (y - Ax)) \tag{3}$$

Most of the popular image priors have closed form proximal operator, making the problem both interpretable and tractable. But the iterative process can take a long time to converge and depending on the prior used, might not generate the most natural image.

1.2. Data-driven recovery methods

Deep learning based techniques have been producing the state-of-the-art results in inverting ill-posed imaging problems. These methods typically use an over-parameterized neural network to approximate an inverse mapping from the perturbed image space $\mathcal{Y}$ to the true image space $\mathcal{X}$ by drawing a large amount of sample pairs x, y from $\mathcal{X}, \mathcal{Y}$ respectively.

These data-driven recovery methods generate high quality, realistic images and once trained, are much faster than iterative methods. However, for different inverse problems, the network needs to relearn the parameters with a corresponding image data set.

1.3. Combination of proximal methods and data-driven learning

In order to utilize a generalized, interpretable framework like proximal gradient while efficiently recovering photo-realistic images, the combination of proximal methods and data-driven learning has risen in popularity. These methods first use hand-designed prior to set up the algorithm and then use training to learn parameters of the prior.

In this review, we will discuss two main ways of integrating proximal methods with learning for image inverse problems: unrolled algorithms and learning proximal operators for Plug-and-play (PnP) frameworks.

2. Unrolling

One intuitive way to leverage neural network to solve image inverse problem is to take a well-explained iterative proximal algorithm and unroll it. Each iteration of the algorithm becomes a connected layer of a network and the parameters become learnable weights. Since the network only contains a few iterations, the unrolled network is usually a magnitude or more faster than the original iterative approach. And by training with ground truth x and perturbed image y pairs, these networks also offer improved reconstruction quality. We will discuss some of the most popular unrolled networks in the following sections.

2.1. From ISTA to LISTA

A popular hand-designed prior for natural images is the l_1-norm, which enforces sparsity. This gives the Lasso problem:

$$\min_x \frac{1}{2}\|y - Ax\|_2^2 + \lambda\|x\|_1 \tag{4}$$

In the simple case where the sensing matrix A is the identity matrix, the proximal operator $\mathcal{P}_{l_1}(x)$ has a close form solution, the soft-thresholding operator $\mathcal{P}_{sft,\lambda}(x)$. And for other A matrices, the Lasso problem can be solved using iterative soft thresholding algorithm (ISTA).

Given an input image y, ISTA iterates the following:

$$v_t = y - A\hat{x}_t \tag{5a}$$

$$\hat{x}_{t+1} = \mathcal{P}_{sft,\lambda}(\hat{x}_t + \beta A^\top v_t) \tag{5b}$$

until convergence. Data fidelity is enforced by (5a) while (5b) ensures sparsity.

Fast ISTA (FISTA) (Beck & Teboulle, 2009) introduces a momentum term and converges in about an order-of-magnitude fewer iterations. To further reduce the number of iterations without sacrificing reconstruction quality, Gregor and Lecun proposed Learned ISTA (LISTA) (Gregor & LeCun, 2010).

There are two major changes from ISTA to LISTA. 1) The iteration process is unrolled and truncated into a T-layer residual neural network. 2) the soft-thresholding parameter λ_t is learned for each unrolled layer $t \in [1, ..., T]$. We can rewrite ISTA as the following:

$$\hat{x}_{t+1} = \mathcal{P}_{sft,\lambda_t}(S\hat{x}_t + By) \tag{6}$$

where $B = \beta A^\top$ and $S = I_N - BA$. This non-linear, parameterized, feed-forward architecture is trained with x, y pairs to approximate optimal sparse code. Depending on the problem, one can also relax the constraint on sensing matrix A and estimate B and S.

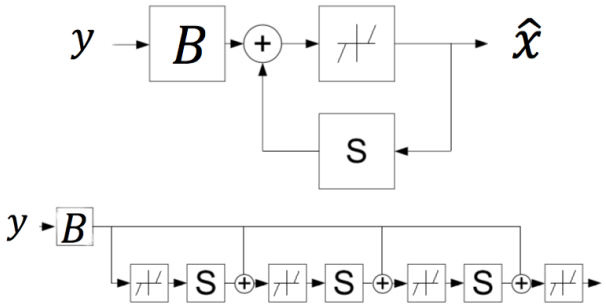


Figure 1. **Top**: block diagram of ISTA algorithm. **Bottom**: Learned ISTA truncated to 3 layers

LISTA converges in significantly fewer iterations than FISTA. It takes FISTA around 18 iterations to reach the same mean square error as a single layer LISTA network trained for 1 iteration. However, for $B \in \mathbb{R}^{N\times M}$, and $S_t \in \mathbb{R}^{N\times N}$, each iteration has a higher computational complexity at $O(TN^2)$ for a T-layer network, compared to ISTA's $O(MN)$ for $A \in \mathbb{R}^{M\times N}$. The cost per iteration can be reduced by approximating S with lower dimensional matrices $U_1, U_2 \in \mathbb{R}^{q\times M}$ such that $S = U_1^\top U_2$.

In general, LISTA massively enhances the performance of ISTA and FISTA and inspired a myriad of unrolling of other iterative algorithms.

2.2. From AMP to L-AMP

Approximate message passing (AMP) can also be used to solve l_1 problems. It has a similar formulation as ISTA:

$$v_t = y - A\hat{x}_t + b_t v_{t-1} \tag{7a}$$

$$\hat{x}_{t+1} = \mathcal{P}_{sft,\lambda_t}(\hat{x}_t + A^\top v_t) \tag{7b}$$

where $\hat{x}_0 = 0, v_{-1} = 0$ and

$$b_t = \frac{1}{M}\|\hat{x}_t\|_0 \tag{8}$$

$$\lambda_t = \frac{\alpha}{\sqrt{M}}\|v_t\|_2 \tag{9}$$

for tuning parameter α.

The main differences between AMP and ISTA are that AMP includes a "Onsager correction" term $b_t v_{t-1}$ and that the soft-thresholding parameter λ_t is now iteration dependent.

When A is a large i.i.d. sub-Guassian random matrix commonly seen in compressive sensing, the Onsager correction

reduces the problem to denoising as it assumes the input to the shrinkage function can be written as:

$$r_t = x^* + \mathcal{N}(0, \sigma_t^2 I_N) \tag{10}$$

where x^* is the true signal we are trying to recover and σ_t^2 is the known variance of the white Gaussian noise given by

$$\sigma_t^2 = \frac{1}{M}\|v_t\|_2^2 \tag{11}$$

This assumption allows AMP-l_1 to converge in much fewer iterations than ISTA. However, when A deviates from an i.i.d. sub-Gaussian matrix, AMP-l_1 often diverges.

To address the dependency on a Gaussian A, Vector AMP (VAMP) was proposed (Schniter et al., 2016). VAMP also performs denoising and Onsager correction while maintaining the same per iteration complexity as AMP. But VAMP can work with a much larger class of A, the right-rotationally invariant matrices, which retains its distribution when right multiplied by a fixed orthogonal matrix with large dimension.

For more details on the VAMP algorithm see Schniter et al.. In short, VAMP has 2 stages, a linear MMSE (LMMSE) stage and a shrinkage stage. In the first stage, MMSE is performed on $\tilde{r}_t$ with respect to $\tilde{\mathcal{P}}$ and Onsager correction on r_t. In the second stage, VAMP performs denoising on r_t with respect to $\mathcal{P}$ and then Onsager correction on $\tilde{r}_{t+1}$.

Empirically, VAMP-l_1 requires about half the iterations of AMP-l_1, which requires an order-of-magnitude fewer iterations than FISTA.

AMP and VAMP can both be unrolled into feed-forward neural networks, which will be denoted as LAMP and LVAMP respectively.

The differences between LAMP and LISTA stem from the differences between AMP and ISTA, namely the addition of Onsager correction and an iteration dependent λ_t. Furthermore, LAMP can relax the "tied" constraint $B = A^\top$ and achieve better performance when training the two variables as "untied." The network trains on x, y pairs to learn A_t, B_t and α.

Similarly, VAMP can be trained for learnable shrinkage parameter $\theta_t = \alpha$ and learnable LMMSE parameters

$$\tilde{\theta} = U, s, V, \sigma_\omega$$

where $U\text{Diag}(s)V^\top$ is the economy SVD of A and σ_ω^2 is the variance of noise ω.

To avoid over-fitting, Borgerding et al. proposes to train LAMP and LVAMP with a hybrid of "layer-wise" and global optimization. For $t = 1, ..., T$, parameter θ and $\tilde{\theta}$ is learned up to and including layer t. In the untied case, bootstrapping is used to escape bad local minimum.

The computational complexity per iteration for LAMP is $\approx 2TMN$ for a T-layer network. For i.i.d. signals and SVD-parameterized A, the complextiy of LVAMP is the same as LAMP at $\approx 2TMN$.

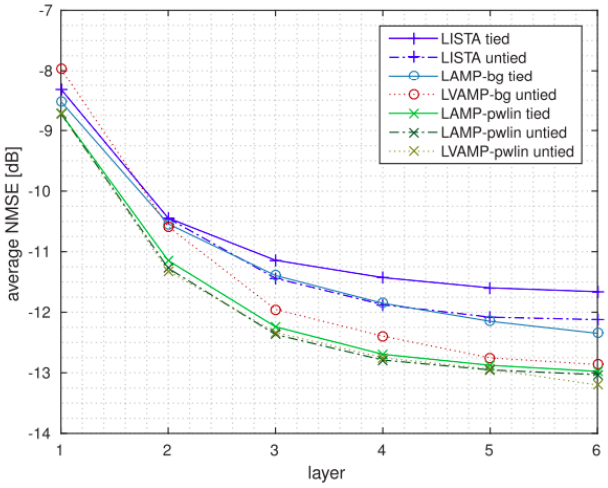


Figure 2. Test NMSE versus layer for compressive random access. bg stands for bernoulli-Gaussian, pwlin stands for piece-wise linear shrinkage

The addition of Onsager correction in LAMP-l_1 provides significant performance improvement over LISTA. To converge to the same error, LISTA takes 15 layers while LAMP-l_1 only needs 7 layers. The performance of LAMP can further be improved by replacing the l_1 shrinkage with more sophisticated exponential shrinkage, spline shrinkage, or bernoulli-Gaussian priors.

LVAMP converges at similar rate as LAMP given the same shrinkage function when A is i.i.d. Gaussian matrix. However, as condition number of A increases, LAMP starts to slow down while LVAMP maintains its fast convergence rate.

2.3. Learned Denoising-AMP

The Onsager correction term in AMP coerces many image inverse problems to be generalized into a denoising problem with white Gaussian noise. Instead of exploring different priors, Metzler et al. (2016) exploits the denoising step of AMP by replacing the shrinkage function $\mathcal{P}$ with denoisers. The many sophisticated imaging denoising algorithms available, such as BM3D, boost the reconstruction quality of the proposed denoising-AMP (D-AMP) and is robust to measurement noise.

With the caveat that the denoiser *D* easily propagates gradient. the D-AMP algorithm can be unrolled into a learned

D-AMP network (LDAMP) (Metzler et al., 2017):

$$b_t = \frac{1}{M} v_{t-1} \text{div} D_{t,\mathrm{w}_{t-1}}(x_{t-1} + A^H v_{t-1}; \lambda_{t-1}) \tag{12a}$$

$$v_t = y - Ax_t + b_t \tag{12b}$$

$$\lambda_t = \frac{1}{\sqrt{M}} \|v_t\|_2 \tag{12c}$$

$$x_{t+1} = D_{t,\mathrm{w}_t}(x_t + A^H v_t; \lambda_t) \tag{12d}$$

where $\text{div} D_{t,\mathrm{w}_{t-1}}$ stands for the divergence of denoiser D with weights w_{t-1}. div$D$ is usually calculated through Monte Carlo approximation.

While the formulation has striking resemblence to LAMP, the major difference between LDAMP and LAMP is that LDAMP only trains on the weights of the denoiser w_t layer-by-layer instead of the A, B matrices in LAMP.

LDAMP achieves state-of-the-art image compressive sensing result that outperform DAMP both in peak signal-to-noise ratio (PSNR) and speed. And because the denoisers can produce more photo-realistic images than the common shrinkage functions, the resulting reconstruction are perceptually more natural.

2.4. ISTA + GAN

LISTA, LAMP, and LVAMP improves the convergence rate of traditional iteration-based algorithm by learning shrinkage parameters and sensing matrix A. However, the quality of reconstructed $\hat{x}$ is still limited by the prior's ability to remove artifact. LDAMP improves on that by employing more sophisticated denoisers. But traditional image prior and hand-designed denoisers cannot fully capture the low-dimensional manifold of natural image $\mathcal{X}$ as well as deep learning methods that directly draws sample pairs from $\mathcal{X}$.

Mardani et al. (2018) proposes GANCS to learn the manifold $\mathcal{X}$ through generative adversarial network (GAN). The goal is to find the intersection between the feasible set of $y = Ax$ and $\mathcal{X}$.

We can rewrite equation (6) from LISTA to the following:

$$\check{x} = \mathcal{G}(By) \tag{13a}$$

$$\hat{x}_{t+1} = S\check{x} + By \tag{13b}$$

The soft-thresholding operator in ISTA and LISTA is replaced by a generator network $\mathcal{G}$, which projects the initial guess By onto the manifold $\mathcal{X}$. Eq(13b) remains the same as in LISTA, ensuring that the guess $\check{x}$ is consistent with the input y.

Another change from LISTA to GANCS is that the loss function is now replaced by a discriminator network $\mathcal{D}$. The $\mathcal{D}$ network tries to assign 1 to real image drawn from $\mathcal{X}$ and 0 otherwise. The better $\mathcal{G}$ is at mapping the input By to $\mathcal{X}$, the more likely it is to fool $\mathcal{D}$.

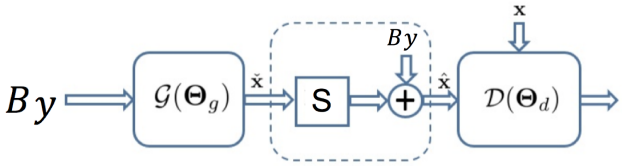


Figure 3. GANCS structure for manifold learning

GAN often suffer from vanishing gradient problem, and depending on the loss function used for training GAN, $\mathcal{D}$ may severely over power $\mathcal{G}$ such that the model collapses. To mitigate the problem, Mardani et al. chose to adapt a mixture of l_1 costs and Least-squares GAN (LSGAN) (Mao et al., 2017) that jointly minimize the discriminator cost

$$\min_{\Theta_d} \mathbb{E}_x[(1 - \mathcal{D}(x; \Theta_d))^2] + \mathbb{E}_y[(\mathcal{D}(\mathcal{G}(By; \Theta_g); \Theta_d))^2] \tag{14}$$

and the generator cost

$$\begin{aligned} \min_{\Theta_g} \mathbb{E}_y \Big[\|y - A\mathcal{G}(By; \Theta_g)\|^2 \Big] \\ + \eta \mathbb{E}_{x,y} \Big[\|x - \mathcal{G}(By; \Theta_g)\|_1 \Big] \\ + \lambda \mathbb{E}_y \Big[(1 - \mathcal{D}(\mathcal{G}(By; \Theta_g); \Theta_d))^2 \Big] \end{aligned} \tag{15}$$

through stochastic alternating minimization.

While GANCS draws inspiration from LISTA and contains layers of residual blocks in the dashed box in 3, it is not strictly an unrolled network. The projection step $\mathcal{G}$ happens only once for each iteration.

Since GANCS is proposed in hope for better and faster MRI reconstruction, Mardani et al. tested the algorithm on contrast enhanced abdominal images from pediatric patients. On a NVIDIA Titan X Pascal GPU, image reconstruction takes only 10-20 milli-seconds, which is two orders of magnitude faster than the current state-of-the-art. The reconstruction also has higher signal-to-noise ratio (SNR) and structure similarity index (SSIM) than existing methods.

At this point, we see that the shrinkage functions in the traditional iterative algorithms can be replaced by deep learning to achieve joint optimization of efficiency and image reconstruction quality. However, theoretical guarantee of GAN-based algorithm still requires further investigation.

3. Learning Proximal Operators for Plug-and-play Frameworks

Recently, we've seen a trend of incorporating deep priors in image reconstruction tasks. Within a proximal framework, instead of finding the prior function itself, we can directly learn the proximal operator of the prior.

Among proximal methods, alternating direction method of multiplier (ADMM) (Boyd et al., 2011) has been used widely in image processing tasks due to its modularity. Here, we'll review the so-called Plug-and-play (PnP) ADMM frameworks and show how they can be combined with deep learning to offer more flexible solutions while being able to extract useful distributions from training images.

3.1. From ADMM to Plug-and-play ADMM frameworks

Recall that optimization problems of the following form

$$x' = \underset{x}{\operatorname{argmin}}\, f(x) + \lambda\phi(x) \tag{16}$$

is commonly used for image processing or reconstruction where we have a data fidelity term and a prior ϕ. An example of such optimization problems is maximum a posteriori estimation (MAP). ADMM solves the above problem by introducing an additional variable for the ϕ term and converting the unconstrained problem to a constrained one, i.e.:

$$x', z' = \underset{x,z}{\operatorname{argmin}}\, f(x) + \lambda\phi(z)$$
$$\text{subject to } x = z$$

Then, ADMM finds the minimizers by iteratively solving a series of subproblems that take the form of proximals:

$$x_{t+1} = \underset{x\in\mathbf{R}^n}{\operatorname{argmin}}\, f(x) + \frac{\rho}{2}\|x - z_t + \bar{u}_t\|_2^2 \tag{17a}$$

$$z_{t+1} = \underset{z\in\mathbf{R}^n}{\operatorname{argmin}}\, \lambda\phi(z) + \frac{\rho}{2}\|z - x_{t+1} + \bar{u}_t\|_2^2 \tag{17b}$$

$$\bar{u}_{t+1} = \bar{u}_t + (x_{t+1} - z_{t+1}) \tag{17c}$$

Where $\bar{u}_t = \frac{1}{\rho}u_t$ and ρ is an augmented Lagragian parameter and can be seen as a step size. In this case, we can see that f and ϕ are seperated into two independent update steps, hence the modular structure. In the context of image processing, this allows us to decouple the sensing and prior. Commonly used ϕ can be total variation regularization, wavelet sparsity, etc.

(Venkatakrishnan et al., 2013) proposed that we do not need the actual ϕ for solving the inverse problem and instead, directly finding the corresponding proximal would suffice. In other words, the priors take the form of proximal operators in the ADMM formulation. Their approach is to replace (17b) with a denoiser $\mathcal{P}$ such that

$$z_{t+1} = \mathcal{P}(x_{t+1} - u_t) = \mathcal{P}(\tilde{z}_t) \tag{18}$$

This $\mathcal{P}$ therefore serves as a proximal operator for the update step. The resulting algorithm is called Plug-and-Play (PnP) ADMM and it has been shown to achieve good performance in a wide range of image processing tasks.

The denoiser $\mathcal{P}$ can be a traditional image filter or even a denoising network trained on natural images as in the case of (Rick Chang et al., 2017) and (Meinhardt et al., 2017). However, one issue with the PnP framework it converges under a number of conditions, which limits the denoisers we could use. Furthermore, how to train a neural network that properly satisfy these conditions is a challenging problem. We will review relevant works in the following sections.

3.2. One Network to Solve Them All

(Rick Chang et al., 2017) learns the proximal operator $\mathcal{P}$ using generative adversarial networks (GANs). The goal is to train a single network that suits multiple image restoration tasks at once. The paper considers only linear problems and f in equation (16) becomes $\frac{1}{2}\|y - Ax\|_2^2$, where A is the sensing matrix. This penalty form means they assume Gaussian noise in the data.

Figure 5 offers a geometric interpretation of the approach. Given a large natural image dataset, (Rick Chang et al., 2017) proposed to learn a classifier $\mathcal{D}$ that fits the underlying distribution of natural images. $\mathcal{P}$ is trained to fit the proximal of $\mathcal{D}$ and serves as a projector onto the natural image set. Another important takeaway from this figure is that the proximal $\mathcal{P}$ should be independent of the type of inverse problem or sensing matrix here, which is possible due to the modularity of ADMM.

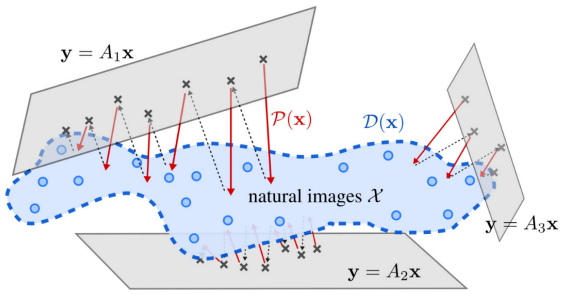


Figure 4. Geometric interpretation of proximal $\mathcal{P}$ and classifier $\mathcal{D}$. An ideal $\mathcal{P}$ should be able to project image-like signals to the set of natural images $\mathcal{X}$. A_1, A_2, A_3 correspond to the sensing matrices for different problems.

The projector $\mathcal{P}$ and classifier $\mathcal{D}$ can be trained jointly via adversarial learning. The training framework is shown in Figure 5 below.

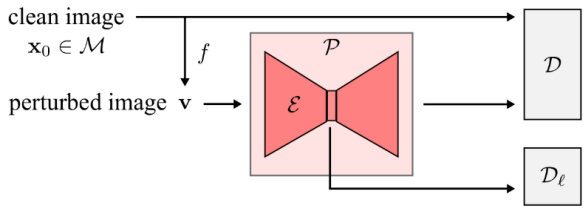


Figure 5. Adversarial Learning framework for finding $\mathcal{P}$ and $\mathcal{D}$. $\mathcal{M}$ denotes the training set and ϵ is a parameter of the denoiser.

The projector and classifier can be seen as players with adversarial objectives. On one hand, the projector is trained to confuse the classifier by projecting the perturbed images onto decision boundary of $\mathcal{D}$. As the projector improves, $\mathcal{D}$ (seen as a discriminator) will also tighten its decision boundary. If the training is successful, then $\mathcal{P}$ should be able to project perturbed images to somewhere close to or within the set of natural images. $\mathcal{D}_l$ is an additional classifier in the latent space to help improve performance of the training (in contrast to $\mathcal{D}$ which is a classifier in the image space).

One critical issue when using a neural network is that the problem becomes non-convex if the decision function of $\mathcal{D}$ is non-convex. For non-convex ADMM, the paper quotes the established theory that the algorithm converges to a stationary point if (1) the gradient of the decision function ϕ is Lipschitz continuous and (2) ρ is sufficiently large (still, global optimum is not guaranteed). The convergence under non-convex settings is still an active field which we will discuss in Section 3.3 in detail. To achieve these conditions, cross entropy loss is used as the discriminative loss and $\phi(x) = \log(S(\mathcal{D}(x)))$, where S is the sigmoid function. The smooth exponential linear unit (ELU) (Clevert et al., 2015) is used as its activation function to guarantee differentiability and network weights are truncated to bound the gradients. The ELU unit is defined as

$$ELU(x) = \begin{cases} x & \text{if x} > 0 \\ \alpha(\exp(x) - 1) & \text{otherwise (here } \alpha > 0) \end{cases}$$

The learned proximal can be used for different image restoration problems by only replacing sensing matrix A. The authors tested the algorithm on compressive sensing, pixelwise inpainting and denoising, scattered impainting, blockwise inpainting and super-resolution and has achieved reasonably good results (Figure 6). However, we still noticed quite a few artifacts in the restored images.

task	ℓ_1 prior	proposed	specially-trained
compressive sensing (10×)	13.01 (±2.75)	25.43 (±3.74)	25.18 (±2.82)
pixelwise inpaint, denoise	20.68 (±1.65)	26.29 (±1.98)	30.13 (±1.66)
2× super-resolution	27.30 (±2.50)	27.11 (±3.21)	22.59 (±2.89)
scattered inpaint	27.85 (±2.58)	25.69 (±3.45)	18.30 (±2.55)

(a) ImageNet

Figure 6. Results on 100k Images randomly drawn images from the ImageNet dataset, for various image processing tasks. The SNR performances are competitive to the state-of-the-art problem specific networks

In summary, the main contribution of this method is that it eliminates the need for problem-specific retraining and the flexibility makes it suitable for mobile platforms. There are two major limitations to this method. One is the global optimum is not guaranteed, especialy for more challenging problems such as block impainting and 4X super-resolution. Second is that the weights of the networks can still affect the projection operator. We also noticed that the authors trained and tested only within each dataset (MNIST, ImageNet, etc). Even though the learned proximal is not specific to the image processing task, it's unclear whether it is specific to the types of images being used in the training process.

There were multiple works published concurrently with this paper presenting similar ideas. Another really classic work is (Meinhardt et al., 2017), in which they used a deep convolutional denoising network as the proximal in a PnP primal-dual hybrid gradient (PDHG) framework. They have also achieved good performances in a large variety of image processing tasks. They did not providet theoretical backing on the convergence of their method but they stated that it converges well enough in their experiments.

3.3. Theories on Convergence

Plug-and-play ADMM has shown empirical success in a wide range of image processing tasks. However, this leads to the theoretical question: when does the algorithm converge and what denoisers or proximal operators are we allowed use? While the convergence of ADMM with an explicit ϕ is already very well-addressed, the convergence of PnP ADMM where we directly apply a denoiser $\mathcal{P}$ seems to still be an open area. Theoretical analysis of PnP frameworks has many challenges since the denoiser is often non-linear without proper close form expressions (Ryu et al., 2019). Furthermore, given the convergence conditions, how to enforce them on the deep denoisers or learned proximals is an important question.

Lipschitz Condition on P (Ryu et al., 2019) is one of the most recent works to establish convergence of serveral PnP frameworks under a Lipschitz condition on $\mathcal{P}$, which is more relaxed than the non-expansiveness condition. Furthermore, they proposed using real spectral normalization (realSN) to regularize training of the deep denoiser to satisfy such condition, which we will explain in the next section.

Assume $\mathcal{P}$ satisfies

$$\|(P-I)(x)-(P-I)(y)\|_2^2 \leq \epsilon^2 \|x-y\|_2^2 \qquad (19)$$

for all $x, y \in \mathbf{R}^N$ and for some $\epsilon > 0$, then the paper proves that all three PnP methods converge. In particular, assuming condition 19 and f is μ-strongly convex, for some $\epsilon \in [0, 1)$ then the PnP-ADMM has fixed-point convergence for

$$\frac{\epsilon}{(1+\epsilon-2\epsilon^2)\mu} < \frac{\lambda}{\rho} \qquad (20)$$

Even though we focus on ADMM here, the PnP framework can extend to other proximal methods such as forward-backward splitting (FBS) or Douglas-Rachford splitting (DRS). The paper also establishes convergence of these methods under condition (19). Some of the interesting points here they mentioned are (1) PnP DRS and PnP ADMM both requires f to be strongly convex, but PnP FBS additionally requires ∇f to be $L-$Lipschitz and (2) PnP ADMM and PnP FBS has the same set of fixed points. The tradeoff here is that ADMM has better convergence properties while FBS is easier to implement. As the paper pointed out, one issue here is that some applications don't satisfy strong convexity (e.g. compressed sensing, sparse interpolation, super-resolution).

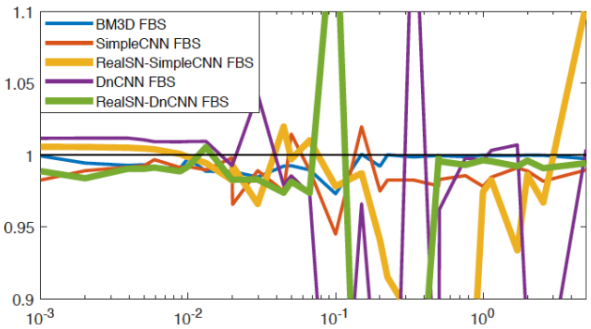
Regularizing Lipschitz Continuity during NN Training Spectral normalization was proposed in (Miyato et al., 2018) and normalize the spectral norms of layer-wise weights to 1 (ReLU layers). Such regularization has many advantages such as stablizing GAN training and improves their performance. RealSN is an extension of the spectral normalization (SN) method to achieve a tighter constraint on the network's Lipschitz constant. Given a convolutional linear operator K_l and its conjugate K_l^*, and U_l and V_l being estimates for the left and right single vectors, each forward pass of the NN consists of the following two steps:
1. Apply one step power method to estimate spectral norm of K_l

$$V_l \leftarrow K_l^*(U_l)/\|K_l^*(U_l)\|_2\,,$$
$$U_l \leftarrow K_l(V_l)/\|K_l(V_l)\|_2$$

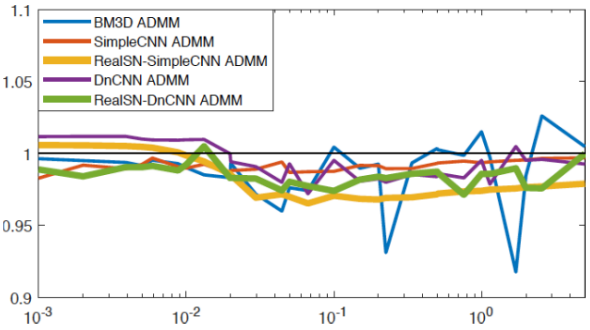
2. Normalize K_l with the spectral norm estimation

$$K_l \leftarrow K_l/\langle U_l, K_l(V_l)\rangle$$

To verify the theory, the authors implemented a simple CNN with RealSN, and DnCNN with RealSN and compared their convergence with a BM3D and the original CNNs for various tasks. They verified that the RealSN-Simple-CNN and RealSN-DnCNN both satisfy condition (19), whereas BM3D does not. This is done by measuring the distribution of the LHS of (19). They also computed the average contraction factor of these methods with PnP FBS and PnP ADMM (lower contraction factor means faster convergence), as shown in Figure 7 . Applying realSN did slightly lower and stabilize the contraction factor in the PnP ADMM case. Overall, PnP ADMM gives more stable results than PnP FBS since PnP FBS requires a certain program parameter α to fall into a specific interval.



(a) PnP-FBS



(b) PNP-ADMM

Figure 7. Average Contraction Factor over 500 iterations. α is the tunning parameter in the programs and corresponds to $\frac{\lambda}{\rho}$ based on our ADMM formulation

In terms of performance, RealSN-Simple-CNN and RealSN-DnCNN gives signficantly higher than the rest of the methods for CS-MRI tasks. It is unclear whether they have all converged, however, and the number of iterations each of them took. On the otherhand, all methods have close performance for single photon imaging within the same number of iterations.

Other Works There are a few other works on the theoretical analysis of PnP frameworks. (Sreehari et al., 2016) proposed a non-expansive condition for the convergence of PnP ADMM. However, this condition is too strict for many type of denoisers. (Chan et al., 2016) proposed a slightly weaker condition and showed that for any "bounded denoisers", PnP ADMM has fixed-point convergence. Currently establishing the convergence of PnP frameworks is still an active research area. It would be interesting to see whether convergence can be secured with more relaxed conditions and without the need for strong convexity. While researchers have found a few sufficient conditions, it is unclear what the actual sufficient and necessary condition is to reach convergence. Furthermore, even though the ADMM theories when ϕ is given are well established, we currently do not know how to find ϕ based on $\mathcal{P}$, that is - we do not know what priors any arbitrary denoiser correspond to. Another interesting question is given these convergence conditions, how do we create effective denoisers and whether we can modify existing state-of-the-art denoisers to fit these conditions. It would be interesting to see any works along these directions in the future.

4. Conclusion

In this report, we reviewed techniques that combine learning based methods with iterative proximal methods. On one hand, learning based methods can learn accurate distributions from training images and has achieved state-of-the-art performances for each specific image processing problem, while the challenge lies in the cost of problem-specific data collection and retraining. On the other hand, proximal methods offer modularity and generalization, making it suitable for a wide range of image processing tasks. However, better priors are needed to improve the reconstruction quality. We are interested in the intersection where their advantages can be combined.

The first class of techniques we reviewed is unrolling, in which we aim to speed up iterative proximal reconstructions by fixing the number of iterations to a small number T and unrolling it to a network. Due to the large number of parameters in the network, we can generate a reasonable approximation even with very few layers. From LISTA to GANCS, by adding the Onsager correction term and incremental relaxing of the proximal operator, both the convergence speed and the recovered signal quality see massive improvement.

When unrolled network is combined with other network structures, as in the case of GANCS, theoretical guarantee of convergence and the speed of convergence is still an open topic of research.

The second category is learning proximal operators for PnP frameworks. The main motivation is to avoid problem-specific training while maintaining the state-of-the-art performances from learning-based methods. Instead of learning the direct inverse mapping, we directly learn the priors in the form of their proximal function via training. The proximal can be learned in many different ways as we mentioned earlier, for example, via GAN or denoising networks, etc. Proximal learning has been demonstrated for a variety of proximal methods such as ADMM, PDHG, etc. These learning proximal frameworks have worked well empirically in many different image processing tasks. However, we are still lacking theory on convergence and the selection of denoisers. This is particularly challenging as we introduce neural networks due to their complexity. Currently, establishing convergence for learning proximal is an active research area.

Finally, when introducing learning, an inevitable problem is the large amounts of training data required, as seen in all the techniques we reviewed. Naturally, another open question here is: can we still learn proximals when we have limited or small amounts of data? This is important because in certain types of problems it is very hard to collect lots of training data. A potential solution could be using a technique similar to (Heckel & Hand, 2018), in which only a single image is needed to train the network. Futhermore, we have seen few works that discuss the potential of applying these techniques to 3D data such as 3D point clouds acquired by LiDARs, etc. 3D data introduces very high complexity in the problem and it would be impressive to see any of the above work applied to 3D reconstruction problems in the future.

References

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Borgerding, M., Schniter, P., and Rangan, S. Amp-inspired deep networks for sparse linear inverse problems. *IEEE Transactions on Signal Processing*, 65(16):4293–4308, 2017.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Chan, S. H., Wang, X., and Elgendy, O. A. Plug-and-play admm for image restoration: Fixed-point convergence and applications. *IEEE Transactions on Computational Imaging*, 3(1):84–98, 2016.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 399–406. Omnipress, 2010.

Heckel, R. and Hand, P. Deep decoder: Concise image representations from untrained non-convolutional networks. *arXiv preprint arXiv:1810.03982*, 2018.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2017.

Mardani, M., Gong, E., Cheng, J. Y., Vasanawala, S. S., Zaharchuk, G., Xing, L., and Pauly, J. M. Deep generative adversarial neural networks for compressive sensing mri. *IEEE transactions on medical imaging*, 38(1):167–179, 2018.

Meinhardt, T., Moller, M., Hazirbas, C., and Cremers, D. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1781–1790, 2017.

Metzler, C., Mousavi, A., and Baraniuk, R. Learned d-amp: Principled neural network based compressive image recovery. In *Advances in Neural Information Processing Systems*, pp. 1772–1783, 2017.

Metzler, C. A., Maleki, A., and Baraniuk, R. G. From denoising to compressed sensing. *IEEE Transactions on Information Theory*, 62(9):5117–5144, 2016.

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

Rick Chang, J., Li, C.-L., Poczos, B., Vijaya Kumar, B., and Sankaranarayanan, A. C. One network to solve them all–solving linear inverse problems using deep projection models. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5888–5897, 2017.

Ryu, E. K., Liu, J., Wang, S., Chen, X., Wang, Z., and Yin, W. Plug-and-play methods provably converge with properly trained denoisers. *arXiv preprint arXiv:1905.05406*, 2019.

Schniter, P., Rangan, S., and Fletcher, A. K. Vector approximate message passing for the generalized linear model. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pp. 1525–1529. IEEE, 2016.

Sreehari, S., Venkatakrishnan, S. V., Wohlberg, B., Buzzard, G. T., Drummy, L. F., Simmons, J. P., and Bouman, C. A. Plug-and-play priors for bright field electron tomography and sparse interpolation. *IEEE Transactions on Computational Imaging*, 2(4):408–423, 2016.

Venkatakrishnan, S. V., Bouman, C. A., and Wohlberg, B. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pp. 945–948. IEEE, 2013.