ANALYSIS OF THE GENERALIZATION GAP DUE TO LEARNING RATE AND BATCH SIZE

CS7180 RESEARCH PROJECT REPORT
NORTHEASTERN UNIVERSITY

Kaushik Nishtala
nishtala.k@northeastern.edu

December 17, 2020

ABSTRACT

The performance of deep neural networks strongly depend on the choice of initial learning rate and batch size. Recent literature showed that the large learning rate achieves better generalization soon after the learning rate is annealed than the model trained with smaller learning rate throughout. On the other hand, models trained using only small learning rates or large batches have been found to generalize poorly despite enjoying faster optimization of the training loss. Our understanding of deep learning methods still remains limited in areas of generalization and network properties, so this project explores published methods in different settings to address these questions. In addition, this project also discusses an extension of the learning rate phenomenon to strongly convex objectives.

***K*eywords** Learning rate · Batch size · Generalization

1 Introduction

When training deep neural networks, many people still hold an opinion that training with optimal hyperparameters is like black magic because there are just plenty of hyperparameters to tune. For instance, what kind of learning rate policy to follow, what kernel size to pick for the architecture, what weight decay, and dropout value will be optimal for the regularization? among others. In the interest of this project, I focus on the learning rate and batch size, the most important hyperparameters to tune because this choice of what initial learning rate or batch size we pick can have a significant effect on the performance of deep neural networks. More specifically, It has been observed in practice that when using a small learning rate or larger batch, there is a degradation in the quality of the model, as measured by its ability to generalize.

The importance of this problem is partly attributed to a potential improvement in generalization as a model's ability to generalize is central to the success of a model. On the other hand, the sequential nature of the iteration and small batch sizes does not make Stochastic Gradient Descent really good for parallelization. Although there have been many efforts made to parallelize it for deep learning, they're still limited or bottlenecked by the small batch sizes.

So one obvious solution to resolve this is to use large batch sizes as this increases the amount of computation per iteration, which then can be effectively distributed or parallelized. So there has been a considerable amount of research in this direction to reduce this generalization gap in order to improve model scalability which could potentially reduce the training time by orders-of-magnitude. Additionally, numerous papers have proposed explanations for these phenomenon, such as sharpness of the local minima [1, 2] the time it takes to move from initialization [3, 4] and the scale of SGD noise [5]. However, we still have a limited understanding of a major part of the large learning rate as well as the large batch phenomenon.

My Contribution. In an effort to view and understand this problem in different perspectives/settings, this project aims to replicate and survey the results of the two closed studies and experiment with methods to identify the issue of the generalization gap. Next, I implement them in a practical Deep learning setting like WideResNet on different datasets to identify the regularization effects of large learning rates and batch sizes and compare the network test loss with those networks trained with adaptive gradient methods. Finally, I extend them to a convex setting with a theoretical and empirical analysis to show that this behaviour can appear even in strongly-convex learning problems such as Linear regression.

Organization. In Section 2, I discuss the results of some closed related work. In Section 3, I mention the results of my experiments extending previous work. In Section 4, I provide a theoretical and empirical demonstration of this phenomenon with a simple convex example and discuss findings and future directions in Section 5.

2 Literature Survey

There have been many recent studies to understand the effect of learning rate and large batch and in turn the generalization gap using properties of the loss landscape. This work was partly inspired by Li et al. [6], which explains the large learning rate phenomenon using the learning order of the model, i.e., the rates at which the model learns different types of examples. They have theoretically and empirically demonstrated the effects of large learning rate using a simple neural network setting.

Moreover, there lies another body of work [7, 8, 3, 9, 6] that suggests that the stochasticity of SGD is pivotal to the learning-rate effect. That is, different learning rates lead to different generalization behavior due to variation in stochasticity. They hypothesize that this is due to the regularization effects caused by the stochasticity in the gradient estimates of SGD at high learning rates.

Finally, the task of training with larger batch sizes is closely related to learning rate since the optimal choice of these two parameters is taken together in practice, and there have been tremendous effort to extensively study these phenomena. They particularly focus on the computer vision tasks using SGD optimizer. Particularly, Keskar et al. [1] proposed that training with a large batch size or small learning rate leads to sharp local minima and explains with empirical results on how this results in the generalization drop.

This project is heavily influenced by the works done by Li et al. [6] and Keskar et al. [1]. The following subsections summarize and discuss their methodologies and main results in detail:

2.1 Study of the large learning rate phenomenon

2.1.1 Premise

Figure 1 shows the train and validation accuracies of WideResNet16 network trained on CIFAR-10 data. It is plotted against number of epochs for both small and large learning rate. The grey dotted vertical line represents the point where the learning rate is annealed. From looking at the section of accuracy curve before annealing, it seems to us that the small learning rate model outperforms the large learning rate model in both training and test error. More concretely, the model trained with small learning rate which is shown in red outperforms the large learning rate shown in blue until epoch 60. Only after annealing the learning rate, a sudden performance peak is observed in the accuracy curve corresponding to the large learning rate and it outperforms the small learning rate in terms of generalization.

2.1.2 Proposed Approach

Li et al. [6] explained this phenomenon via a concept of learning order of the model, i.e., the rates at which the model learns different types of examples. And to do so, they constructed a simple distribution with which when we train a two-layer neural network under large and small initial learning rates, the network's learning order helps us determine its generalization.

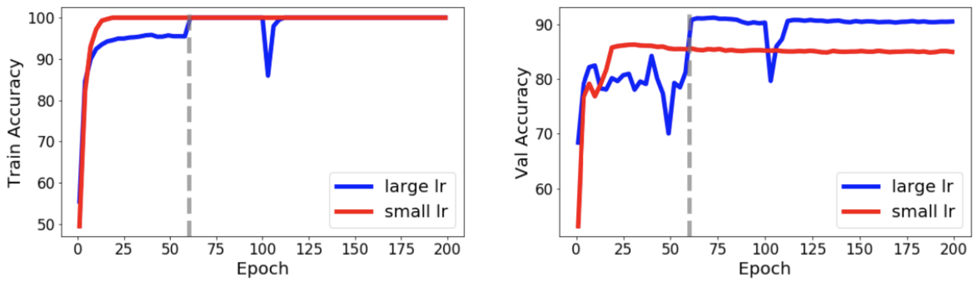


Figure 1: Generalization disparity between large learning rate + anneal and small learning rate model

2.1.3 Methodology

INFORMAL DEFINITION: They considered a distribution over training examples consisting of two types of patterns, where pattern here refers to a grouping of features. The first type consists of a set of easy-to-generalize and hard-to-fit patterns. The second type consists of a set of hard-to-generalize and easy-to-fit patterns

UNDERSTANDING THE TERMS. A feature with very low noise (discrete pattern) are relatively easier to generalize than those that have very high noise which makes them hard to generalize and we would then require much more samples or sample complexity to learn them. In a similar sense, easy-to-fit patterns are learnable by low-complexity classifiers as the patterns form a linearly separable data. So, a simple classifier is sufficient enough to learn them , and the opposite is true for hard-to-fit patterns or non-linear data.

FORMAL DEFINITION: Mathematically, the distribution contains examples with two types of components: a P component, which models hard to generalize, easier to fit patterns, and a Q component which models easy to generalize, hard to fit patterns. They assumed that the label y, the dependant variable has a uniform distribution over $\{-1, 1\}$, and the data x is generated as

$$\begin{aligned}
&\text{Conditioned on the label } y \\
&\qquad \text{with probability } p_0, \qquad x_1 \sim \mathcal{P}_y, \text{ and } x_2 = 0 \\
&\qquad \text{with probability } q_0, \qquad x_1 = 0, \text{ and } x_2 \sim \mathcal{Q}_y \\
&\text{with probability } 1 - p_0 - q_0, \qquad x_1 \sim \mathcal{P}_y, \text{ and } x_2 \sim \mathcal{Q}_y
\end{aligned}$$

So, conditioned on the value of y, they generate p_0 fraction of the data to be component P, q_0 fraction of the data to be only component Q and the rest of the data distribution contains both pattern types. Since the component P represents hard-to-generalize and easier-to-fit patterns, for each label of y, P_{-1} and P_1 are assumed to be two half Gaussian distributions with a margin γ_0 between them.

$$x_1 \sim \mathcal{P}_1 \Leftrightarrow x_1 = \gamma_0 w^\star + z | \langle w^\star, z \rangle \geq 0, \text{ where } z \sim \mathcal{N}(0, I_{d\times d}/d)$$
$$x_1 \sim \mathcal{P}_{-1} \Leftrightarrow x_1 = -\gamma_0 w^\star + z | \langle w^\star, z \rangle \leq 0, \text{ where } z \sim \mathcal{N}(0, I_{d\times d}/d)$$

Therefore, we observe that when x is present, the linear classifier sign ${w^*}^T x$ can classify the example correctly with a margin of γ_0. Intuitively, this means that P is defined in such a way that it is linearly separable, thus learnable by low complexity classifiers. However, due to the dimensionality d, P has high noise and requires a relatively large sample complexity to learn. Therefore P satisfies the requirement of modeling hard-to-generalize, easier-to-fit patterns.

Similarly, the Q component is defined in such a way that the distribution Q_{-1} and Q_1 are supported only on three distinct directions $z-\zeta$, z and $z+\zeta$, and because of this low span, they are low-noise and memorizable.

$$x_2 \sim \mathcal{Q}_1 \Leftrightarrow x_2 = \alpha z \text{ with } \alpha \sim [0,1] \text{ uniformly}$$
$$x_2 \sim \mathcal{Q}_{-1} \Leftrightarrow x_2 = \alpha(z + b\zeta) \text{ with } \alpha \sim [0,1], b \sim \{-1,1\} \text{ uniformly}$$

They chose $z-\zeta$ and $z+\zeta$ (shown in blue) to have negative labels and z to have positive labels. Also note that z here is a d dimensional vector that defines our distribution and the norm of ζ is $\ll 1$ so this means that $z-\zeta$, z and $z+\zeta$ are fairly close to each other as shown in Figure 2.

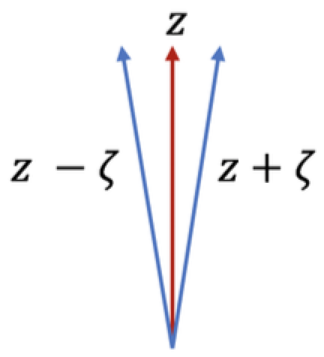


Figure 2: Distribution of z

They choose this type of Q to be the easy-to-generalize, hard-to-fit pattern. Since z here is not linearly separable from $z-\zeta$ and $z+\zeta$, non-linearity is necessary to learn Q.

So now that they formally defined the distribution , they then used a two-layer neural network with RELU activation to learn this data distribution.

We consider this expression $N_A(u, U; x) = u^T(1(Ax) \odot Ux)$ which denotes the output of the neural network for an input x. The first layer weights are denoted by $U \in R^{m \times 2d}$ and the second layer weight is denoted by $u \in R^m$ and 1 is the indicator vector that characterizes the RELU activation function.

The training algorithm that they considered is stochastic gradient descent with spherical Gaussian noise. The weights U are initialized to have i.i.d. entries from a Gaussian distribution with some variance, and at each iteration of gradient descent they added spherical Gaussian noise to the gradient updates where γ_t here denotes the learning rate at time t.

$$U_0 \sim \mathcal{N}(0, \tau_0^2 I_{m\times m} \otimes I_{d\times d})$$
$$U_{t+1} = U_t - \gamma_t \nabla_U(\widehat{L}_\lambda(u, U_t) + \xi_t) = (1 - \gamma_t\lambda)U_t - \gamma_t(\nabla_U \widehat{L}(u, U_t) + \xi_t)$$
$$\text{where } \xi_t \sim \mathcal{N}(0, \tau_\xi^2 I_{m\times m} \otimes I_{d\times d})$$

Having defined this setting, the authors introduced two algorithms corresponding to large and small initial learning rate. Algorithm 1 or LS algorithm states that we use learning rate η_1 for t_0 iterations until the training loss drop below the particular threshold of $\epsilon_1 + qlog2$. After which, we anneal the learning rate to η_2

(which is assumed to be much smaller than η_1) and proceed until the training loss drops to ϵ_2. Similarly, the S algorithm which is for small learning rate model, states that we use a fixed learning rate of η_2 and stop training when the loss is less than ϵ_2.

Algorithm 1 (L-S): The learning rate is η_1 for t_0 iterations until the training loss drops below the threshold $\varepsilon_1 + q \log 2$. Then we anneal the learning rate to $\gamma_t = \eta_2$ (which is assumed to be much smaller than η_1) and run until the training loss drops to ε_2.

Algorithm 2 (S): We used a fixed learning rate of η_2 and stop at training loss $\varepsilon_2' \leq \varepsilon_2$.

2.1.4 Main results

Using these algorithms, they formulated their final results as follows:

Theorem 3.4 (Analysis of Algorithm L-S)**.** *Under Assumption 3.1, 3.2, and 3.3, there exists a universal constant $0 < c < 1/16$ such that Algorithm 1 (L-S) with annealing at loss $\varepsilon_1 + q \log 2$ for $\varepsilon_1 \in \left(d^{-c}, \kappa^2 p^2 q^3\right)$ and stopping criterion $\varepsilon_2 = \sqrt{\varepsilon_1 / q}$ satisfies the following:*

1. *It anneals the learning rate within $\widetilde{O}\left(\frac{d}{\eta_1 \varepsilon_1}\right)$ iterations.*
2. *It stops at at most $t = \widetilde{O}\left(\frac{d}{\eta_1 \varepsilon_1} + \frac{1}{\eta_2 r \varepsilon_1^3}\right)$. With probability at least 0.99, the solution U_t has test (classification) error and test loss **at most** $O\left(p\kappa \log \frac{1}{\varepsilon_1}\right)$.*

The learning order and generalization of the L-S model is such that, before annealing the learning rate, the model only learns an effective classifier for P which amounts to $(1 - q)N$ samples. They also observed that the network does not learn Q before annealing because the large learning rate creates too much noise to effectively learn Q . This is shown in the following lemma:

Lemma 4.2. *In the setting of Theorem 3.4, w.h.p., for every $t \leq \frac{1}{\eta_1 \lambda}$,*

$$|g_t(z + \zeta) + g_t(z - \zeta) - 2g_t(z)| \leq \widetilde{O}\left(\frac{r^2}{\lambda}\right) = \widetilde{O}(d^{-1/4})$$

By taking a function g that simulates the output of the neural network whose input is the distribution Q with $z - \zeta$, z and $z + \zeta$ components, they bounded this function for every timestep before annealing. Doing so, they obtained a poor margin on Q that essentially states that the network does not learn anything meaningful for the Q component before we anneal. Only after the learning rate is annealed, the model memorizes Q and correctly classifies examples with only a Q component during test time.

Similarly, the analysis of the algorithm S shown below is such that the network will quickly memorize the Q component which is low noise and ignore the P component for the examples containing both P and Q components . Thus, it only learns P on this fraction of data which is equivalent to pN examples.

Theorem 3.5 (Lower bound for Algorithm S)**.** *Let ε_2 be chosen in Theorem 3.4. Under Assumption 3.1, 3.2 and 3.3, there exists a universal constant $c > 0$ such that w.h.p, Algorithm 2 with any $\eta_2 \leq \eta_1 d^{-c}$ and any stopping criterion $\varepsilon_2' \in (d^{-c}, \varepsilon_2]$, achieves training loss ε_2' in at most $\widetilde{O}\left(\frac{d}{\eta_2 \varepsilon_2'}\right)$ iterations, and both the test error and the test loss of the obtained solution are **at least** $\Omega(p)$.*

From the lemma given below, $\bar{W}$ is the gradient of the weights restricted to examples containing Q, and it does not learn much about P from those examples and therefore it must overfit to the pN examples of our distribution.

Lemma 5.2. *In the setting of theorem 3.5, let*

$$\overline{W}_t^{(2)} = \frac{1}{N}\eta_2 \sum_{s\leq t}(1-\eta_2\lambda)^{t-s} \sum_{i\in\mathcal{M}_2} \nabla_W \widehat{L}_{\{i\}}(U_s) \tag{5.1}$$

be the (accumulated) gradient of the weight W, restricted to the subset $\mathcal{M}_2$. Then, for every $t = O\left(d/\eta_2\varepsilon_2'\right)$, we have: $\left\|\overline{W}_t^{(2)}\right\|_F \leq \tilde{O}\left(d^{15/32}/\varepsilon_2'^2\right)$. For notation simplicity, we will define $\varepsilon_3 = d^{-1/32}\frac{1}{\varepsilon_2'^2}$. Then, $\left\|\overline{W}_t^{(2)}\right\|_F \leq \tilde{O}\left(\sqrt{d}\varepsilon_3\right)$.

And since this value of pN is less than $\frac{d}{2}$ by the choice of their parameters, we will not have enough samples to learn the d-dimensional distribution P and therefore it is bound to perform relatively worse on these patterns at test time. By this reasoning, it is understandable that the network misclassifies a constant fraction of P-only examples at test time. Because of the poor margin here, the predictions will be heavily influenced by noise. This intuition is followed to prove the classification lower bound of $\Omega(p)$.

To summarize, In a real-world network, the large learning rate model first learns hard-to-generalize, easier-to-fit patterns and is unable to memorize easy-to-generalize, hard-to-fit patterns, leading to a plateau in accuracy. When the learning rate is annealed, it is able to fit these patterns now explaining the sudden improvement in both train and test accuracy. On the other hand, the small learning rate model quickly overfits to the easy-to-generalize, hard-to-fit patterns because of the low amount of SGD noise present in them before fully learning the hard-to-generalize patterns. Therefore, this results in poor validation error on these type of patterns.

Finally, it was proved that the final generalization error of the model trained with the L-S algorithm will end up a factor $O(x) = O(p^{1/2})$ smaller than the generalization error of the model trained with S algorithm.

2.1.5 Mitigation Strategy

To close the generalization gap, the authors have come up with a mitigation strategy where instead of using large learning rate and annealing to a small learning rate, they showed the regularization effect also exists if we use a small learning rate (η_2) and large pre-activation noise and then decay the noise.

$$U_{t+1} = U_t - \eta_2 \nabla_U (\widehat{L}_\lambda(u, U_t) + \xi_t)$$

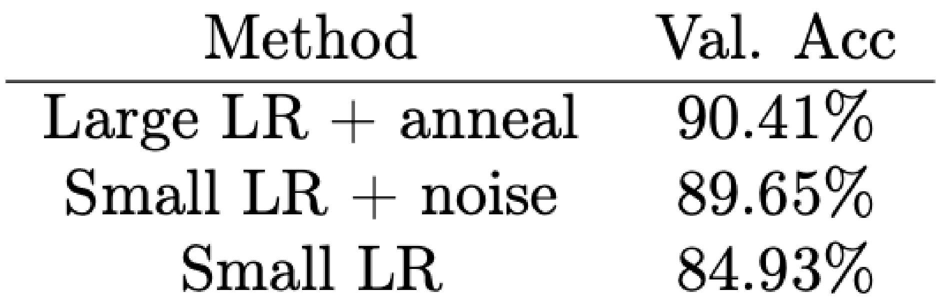
$$\text{where } \xi_t \sim N(0, \tau_\xi^2 I_{m\times m} \otimes I_{d\times d})$$

$$f_t(x) = u^\top \left(\mathbb{1}(U_t x + \Xi_t) \odot (U_t x + \Xi_t)\right)$$

They injected random gaussian variable, Ξ_t which refers to pre-activation noise in order to induce regularization effects. They proved that using this strategy, brings us to the same conclusion as the theorem for large learning rate model (L-S algorithm).

2.1.6 Experiments

Initially, they demonstrated on CIFAR-10 images without data augmentation that this regularization gives a 4.72% increase in validation accuracy when adding noise to a small learning rate as shown in Figure 3.



Method	Val. Acc
Large LR + anneal	90.41%
Small LR + noise	89.65%
Small LR	84.93%

Figure 3: Val. accuracies for WideResNet16 trained and tested on CIFAR-10 data w/o data augmentation

Next, they used the patch-modified CIFAR-10 dataset to show the effects of learning order. Initially, they split the 50000 CIFAR-10 training images in such a way that 20% of the updates are on clean images, 16% of updates are on patches only, and 64% of updates are on mixed images (i.e., those containing both the patch as well as the CIFAR-10 data).

The patches they generated are memorizable 7 x 7 pixel data that are added to a subset of images. The patches are generated in such a way that they are not easily separable, but they are low in variation and therefore easy to memorize. These are equivalent to component Q which are easy-to-generalize and hard-to-fit patterns. The patch will be located in the center of the image as shown in Figure 4.

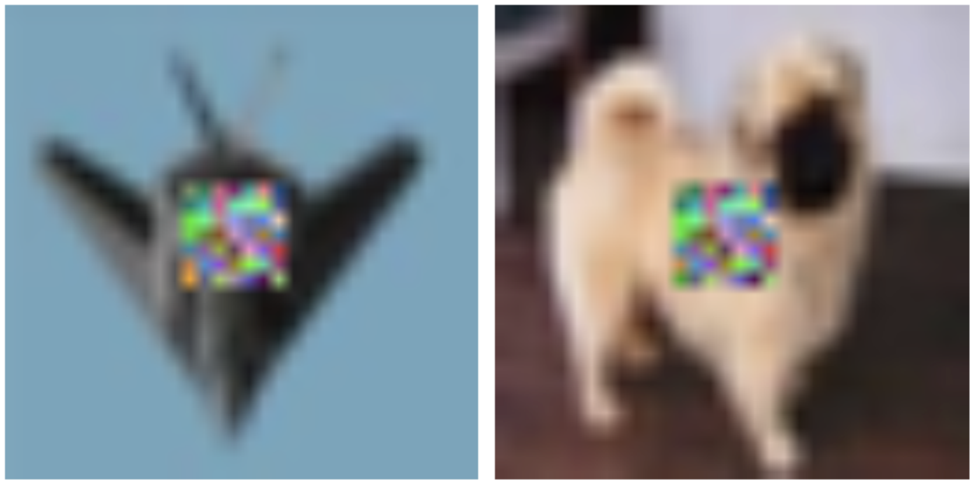


Figure 4: CIFAR-10 images with patches added.

They train on this patch modified dataset using 3 methods - large initial learning rate with annealing at the 30^{th} epoch, small initial learning rate, and small learning rate with noise annealed at the 30^{th} epoch.

Figure 5 plots the validation accuracy vs. epoch on clean (no patch) and patch-only images. From the plot on the right which is for patch-only images, it is obvious that the small learning rate picks up the signal in the patch very quickly as shown in red, whereas the other two methods only memorize the patch after annealing.

From the figure on the left, which is for clean images, we see that the small learning rate method, shown in red is learning the CIFAR images using only a small fraction of all the available data. This is because , from Figure the validation accuracy of a small LR model when training on the full dataset is around 83%, but the validation on clean data after training with the patch is just 70%. Therefore, it explains that the small learning rate model learns only a fraction of clean data.

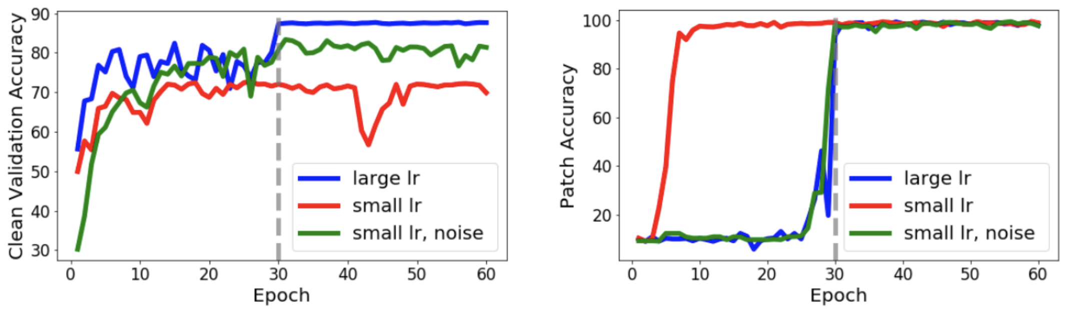


Figure 5: Accuracy vs. epoch on patch-augmented CIFAR-10

2.2 Study of the large batch size phenomenon

2.2.1 Premise

Similar to learning rate, It has been observed in practice that when using a larger batch there is a degradation in the generalization ability of the model - similar to the effect of using small learning rate throughout. As the paper by Keskar et al.[1] quotes "The lack of generalization ability is due to the fact that large-batch methods tend to converge to sharp minimizers of the training function".

Hochreiter & Schmidhuber [10] informally defined that a significant number of large positive eigenvalues in hessian indicates that the minimizer is sharp. On the contrary, small-batch methods converge to flat minimizers characterized by having small eigenvalues of Hessian. They observed that unlike small-batch methods, the large-batch methods are attracted to regions with sharp minimizers as shown in Figure 6 and they are unable to escape basins of these minimizers thereby causing poor generalization.

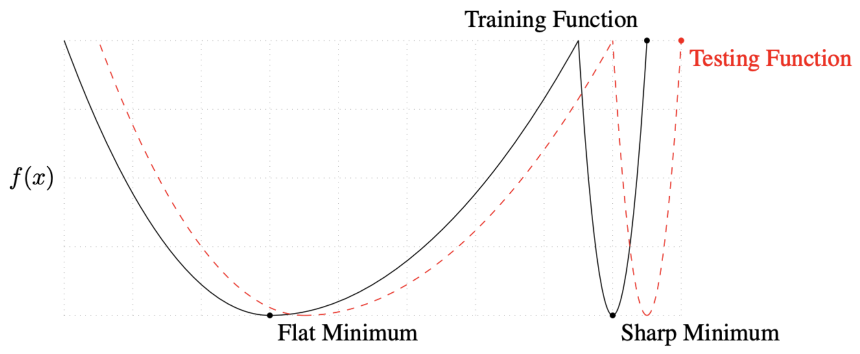


Figure 6: A Conceptual Sketch of Flat and Sharp Minima

2.2.2 Sharpness of Minima

Sharpness of a minimizer can be characterized by the magnitude of the eigenvalues of Hessian, but due to computational constraints, an approximate sensitivity measure is considered. It is explained by a small neighborhood of a solution and computing the largest value that the function can attain in that neighborhood. Formally, they defined the measure of sharpness (or sensitivity) as follows:

Metric 2.1. *Given $x \in \mathbb{R}^n$, $\epsilon > 0$ and $A \in \mathbb{R}^{n \times p}$, we define the $(\mathcal{C}_\epsilon, A)$-sharpness of f at x as:*

$$\phi_{x,f}(\epsilon, A) := \frac{(\max_{y \in \mathcal{C}_\epsilon} f(x + Ay)) - f(x)}{1 + f(x)} \times 100.$$

2.2.3 Experiments

In order to verify this phenomenon, they considered 6 multi-class classification network configurations for their experiments as shown in Figure 7 and use mean cross entropy loss as the objective function.

Name	Network Type	Architecture	Data set
F_1	Fully Connected	Section B.1	MNIST (LeCun et al., 1998a)
F_2	Fully Connected	Section B.2	TIMIT (Garofolo et al., 1993)
C_1	(Shallow) Convolutional	Section B.3	CIFAR-10 (Krizhevsky & Hinton, 2009)
C_2	(Deep) Convolutional	Section B.4	CIFAR-10
C_3	(Shallow) Convolutional	Section B.3	CIFAR-100 (Krizhevsky & Hinton, 2009)
C_4	(Deep) Convolutional	Section B.4	CIFAR-100

Network configurations. NETWORK F1: 784-dimensional input layer followed by 5 batch-normalized layers of 512 neurons each with ReLU activations. The output layer consists of 10 neurons with the softmax activation.

NETWORK F2: Similar to F1. 360-dimensional input layer followed by 7 batch-normalized layers of 512 neurons with ReLU activation. The output layer consists of 1973 neurons with the softmax activation.

NETWORKS C1 AND C3: The C1 network is a modified version of the popular AlexNet configuration. The configuration C3 is identical to C1 except it uses 100 softmax outputs instead of 10.

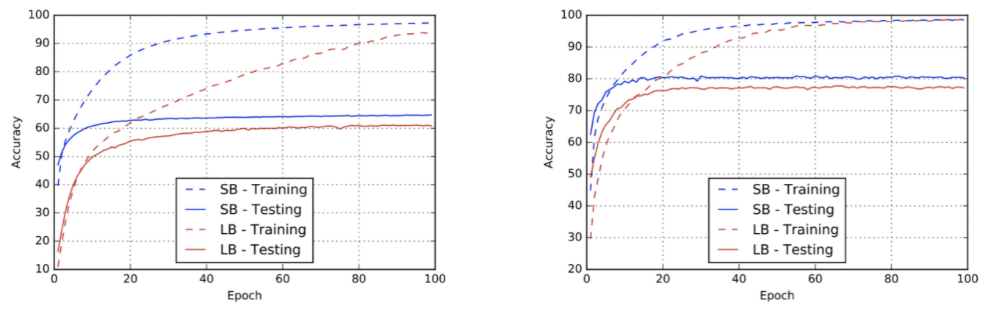
NETWORKS C2 AND C4: The C2 network is a modified version of the popular VGG configuration and the configuration C4 is identical to C2 except that it uses 100 softmax outputs instead of 10.

Setting of the experiments. 10% of the data is considered as Large batch size. 256 data points are considered to be small-batch size. ADAM optimizer is selected for both regimes. All experiments were conducted 5 times from different (uniformly distributed random) starting points and results are shown in Figure 7.

	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

Figure 7: Performance of small-batch and large-batch variants of ADAM on the 6 networks

Note that the generalization gap is not due to over-fitting as no decay in performance on test data is observed due to memorizing noise of the training data. This can be seen in Figure 8 for networks F2 and C1 where there is no dip in the test accuracies (shown by solid lines) due to overfitting.



(a) Network F_2 (b) Network C_1

Figure 8: Training and testing accuracy for SB and LB methods as a function of epochs.

They also produced the sharpness values for each network trained with small batch and large batch and you could notice from Figure 9 that large-batch method defines points of larger sensitivity values of the training function leading to more sharpness.

	$\epsilon = 10^{-3}$		$\epsilon = 5 \cdot 10^{-4}$	
	SB	LB	SB	LB
F_1	1.23 ± 0.83	205.14 ± 69.52	0.61 ± 0.27	42.90 ± 17.14
F_2	1.39 ± 0.02	310.64 ± 38.46	0.90 ± 0.05	93.15 ± 6.81
C_1	28.58 ± 3.13	707.23 ± 43.04	7.08 ± 0.88	227.31 ± 23.23
C_2	8.68 ± 1.32	925.32 ± 38.29	2.07 ± 0.86	175.31 ± 18.28
C_3	29.85 ± 5.98	258.75 ± 8.96	8.56 ± 0.99	105.11 ± 13.22
C_4	12.83 ± 3.84	421.84 ± 36.97	4.07 ± 0.87	109.35 ± 16.57

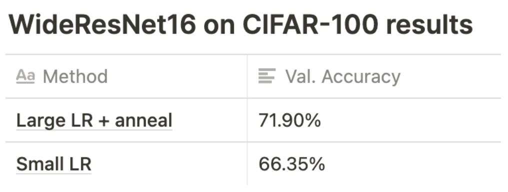
Figure 9: Sharpness of Minima defined by sensitivity

3 Experiments

As for my experiments, I have initially tried to implement and reproduce the papers' computationally feasible results.

3.1 Experiments related to study by Li et al. [6]

Initially, I trained WideResNet16 on CIFAR-100 images without data augmentation. I've observed a similar trend which is consistent with the results produced by the author. We can clearly notice this generalization disparity between large learning rate and small learning rate networks from the figure below.

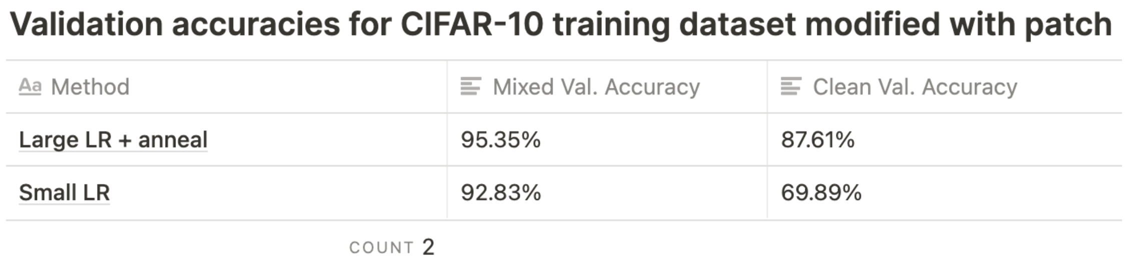


WideResNet16 on CIFAR-100 results

Method	Val. Accuracy
Large LR + anneal	71.90%
Small LR	66.35%

All models are trained for 200 epochs, annealing the learning rates by a factor of 0.2 at the 60th, 120th, and 150th epoch for all models. The large learning rate model uses an initial learning rate of 0.1, whereas the small learning rate model uses initial learning rate of 0.01.

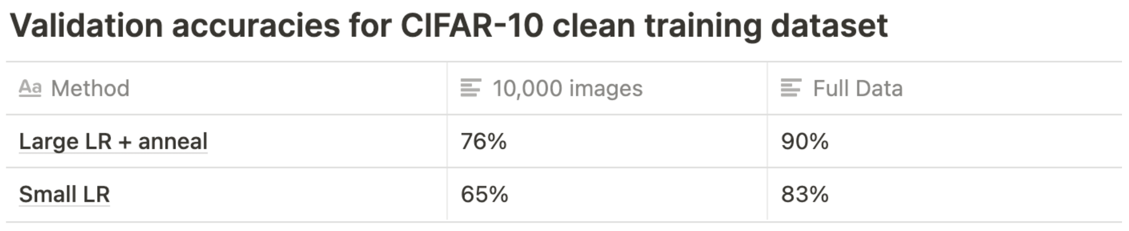
A problem that I discovered from Figure is that the generalization disparity that was observed could be simply because the large learning rate model can already generalize better on clean CIFAR-10 images and may not be due to the learning order effect. The paper does not mention the exact validation accuracies on clean and patch only images, so there is no way to verify this from the plots alone. So, I've trained the model with the modified CIFAR-10 dataset to find the validation accuracies corresponding to the mixed validation set which contain patches and the clean validation set which does not contain any patches. And the results are shown in this table below.



Validation accuracies for CIFAR-10 training dataset modified with patch

Method	Mixed Val. Accuracy	Clean Val. Accuracy
Large LR + anneal	95.35%	87.61%
Small LR	92.83%	69.89%

COUNT 2



Validation accuracies for CIFAR-10 clean training dataset

Method	10,000 images	Full Data
Large LR + anneal	76%	90%
Small LR	65%	83%

As evident from the results, when trained on 10000 clean CIFAR images, the small LR model achieves 65% and the large LR model achieves 76% validation accuracy. For comparison, on the full clean dataset they both achieve 83% and 90% validation accuracies .

Now, If we compare the small learning rate model that is trained with patch modified data vs clean data here, we observe that 69.89% clean image accuracy for the small LR model trained on the patch dataset is much closer to 65% than 83%. This suggests that because of learning order, it is using only a fraction of the available CIFAR-10 samples that is close to 10,000 samples and not close to the full data.

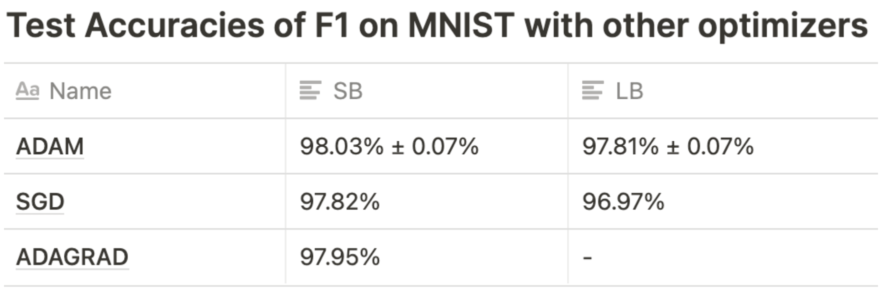
On the other hand, the large LR model achieves final clean validation accuracy of 87.61% when trained on the patch dataset, which is very close to the 90% rather than 76% on the clean dataset. This says that the large LR model is still using the majority of the images to learn CIFAR-10 examples before annealing, as it has not yet memorized the patches.

This is an interesting observation that verifies that the generalization disparity observed in the plot is only due to the learning order of the network. I tried experimenting with the patch parameters and other tunable hyper-parameters and found that changing them too much would make it so that large and small learning rates either ignore the patch or overfit to the patch and output really bad results. Therefore, these experiments are conducted using the optimal parameters that the authors have used.

Configuration. The hyperparameters for the patch generation are chosen to be $\sigma_z = 1.25$, $\beta = 0.1$, $\alpha = 1.75$. My large initial learning rate model trains with learning rate 0.1, annealing to 0.004 at the 30^{th} epoch and the small LR model trains with fixed learning rate 0.004. The small LR with noise model trains with fixed learning rate 0.004, initial noise 0.4, and decays the noise to 4e-6 after the 30^{th} epoch. All models are trained for 60 epochs total, starting from the same choice of patches.

3.2 Experiments related to study by Keskar et al. [1]

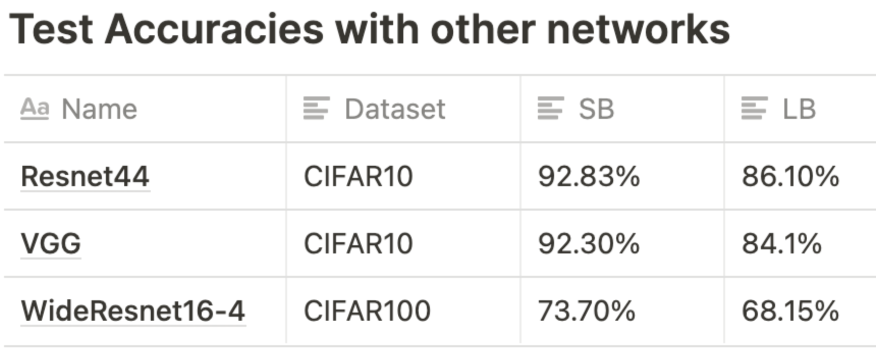
To verify the generalization disparity between small and large batches in a different setting, I implemented the F1 network on MNIST data with Stochastic Gradient Descent and AdaGrad as optimizers. I ran into some memory issues while running other networks but from the result shown below, a similar trend is observed as before. (except the computational infeasible Large batch result from AdaGrad)



Test Accuracies of F1 on MNIST with other optimizers

Name	SB	LB
ADAM	98.03% ± 0.07%	97.81% ± 0.07%
SGD	97.82%	96.97%
ADAGRAD	97.95%	-

And as expected, I've obtained similar results when I used Resnet44, standard VGG configuration and WideResnet16-4 to train on CIFAR-10 and CIFAR-100 datasets to compare the trend. It can be verified from the figure below that large batch generalizes poorly here as well. The focus here is not to achieve superior performance and compare accuracies across the networks but to observe and analyze the generalization gap.



Test Accuracies with other networks

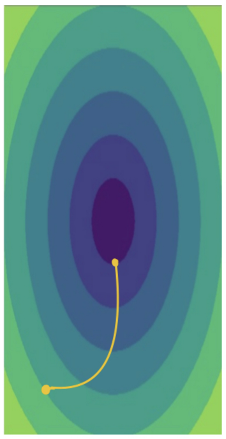
Name	Dataset	SB	LB
Resnet44	CIFAR10	92.83%	86.10%
VGG	CIFAR10	92.30%	84.1%
WideResnet16-4	CIFAR100	73.70%	68.15%

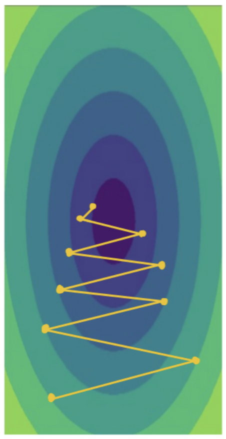
4 Large-learning rate phenomenon in strongly-convex objectives

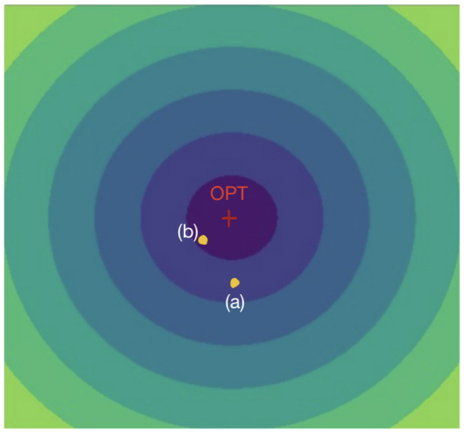
4.1 Premise

We observed that training with a large initial learning-rate followed by a smaller annealed learning rate can vastly outperform training with the smaller learning rate throughout – even when allowing both to reach the same train loss. Now, I verify this claim in strongly convex objectives, particularly a simple linear regression in 2 dimensions.

However a problem with strongly-convex objectives is that they have a unique minima. However, it has been proved by related works that early-stopping breaks this uniqueness: For a set of minima with the same train loss ϵ, they can have different generalization properties when the train loss surface differs from the test loss surface. This could arise due to overparameterization or undersampling. For example, the concept is visualized below. We observe that the left two figures show gradient descent on the train loss, with different learning rate schedules, stopping at the same small value of train-loss. However, they both differ in test loss where the large learning model performs better than small learning rate model in terms of generalization.


(a) Small learning rate


(b) Large, then small learning rate



(c) Test loss landscape

4.2 Theoretical Analysis

We illustrate the main idea with a simple linear regression problem. Let's assume that the data D is distributed over $(x, y) \in \mathbb{R}^2 \times \mathbb{R}$. We consider the independent variable $x \in \{(0,1), (1,0)\}$ uniformly at random. The dependent variable $y = \langle \beta^*, x \rangle$ for some ground truth $\beta^* = (\beta_1^*, \beta_2^*) \in \mathbb{R}^2$.

Our objective is to learn a linear regression such that $\hat{y}(x) := \langle \beta, x \rangle$ with small mean squared population loss. Specifically, for model parameters $\beta = (\beta_1, \beta_2) \in \mathbb{R}^2$, the population loss is given by

$$L_D(\beta) := \mathbb{E}_{x,y \sim D}[(\langle \beta, x \rangle - y)^2]$$

Therefore, our goal is to find $\beta^* = argmin_\beta L_D(\beta)$. Suppose for the experiment, we have a limited set of samples $n \in \{x_i, y_i\}$. Therefore, our empirical loss would be

$$\hat{L}_n(\beta) = \frac{1}{n}\Sigma_{i \in [n]}(\langle \beta, x_i \rangle - y_i)^2$$

We then perform gradient descent on the empirical loss starting at $\beta = 0$ and stopping when the loss $\hat{L}_n(\beta) \leq \epsilon$ for some train loss threshold ϵ while taking the number of samples $n = 3$.

We know that with high enough probability (around 3/4), two of the samples will have the same value of x. Let's say this value is $x = (0,1)$. So, our samples (x_i, y_i) are $\{((0,1), \beta_1^*), ((0,1), \beta_1^*), (1,0), \beta_2^*)\}$. Then after substituting these values in our empirical loss, we have

$$\hat{L}_n(\beta) = \frac{2}{3}(\beta_1 - \beta_1^*)^2 + \frac{1}{3}(\beta_2 - \beta_2^*)^2$$

Since the population loss is the expectation of empirical loss, it is just the average of each sample and is given by

$$L_D(\beta) = \frac{1}{2}(\beta_1 - \beta_1^*)^2 + \frac{1}{2}(\beta_2 - \beta_2^*)^2$$

Therefore, we can observe that since we have an undersampled data ($n = 3$), the train loss is distorted as compared to test/population loss landscape.

The key observation here is that the level sets L_D and $\hat{L}_n$ are not identical although their global minima is. That is, not all points of small train loss $\hat{L}_n(\beta)$ have identical test loss $\hat{L}_n(\beta)$.

To see this, we run a simple analysis - since the model is run until a train loss of ϵ is achieved, we consider two different ways that this train loss could be achieved. One way is to make first term to be 0 and second term to be ϵ from the equation of empirical loss. Another way is to make the first term to be ϵ and second term to be 0. These two cases have test losses which differ by a factor of two, as shown in the figure below.

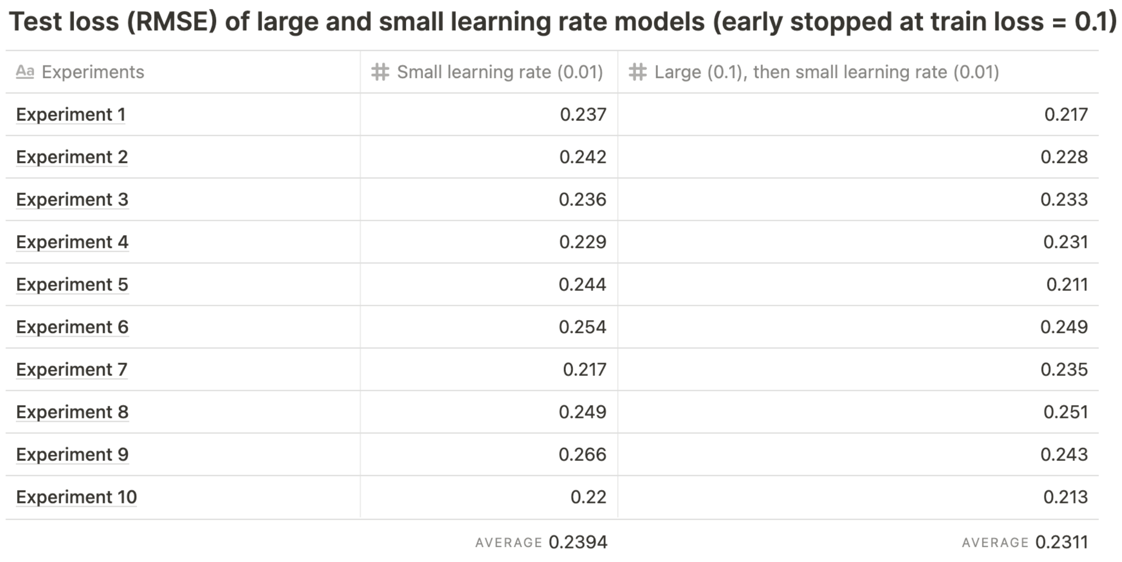
Train Loss $\hat{L}_n(\beta)$	Test Loss $L_{\mathcal{D}}(\beta)$
ε	0.75ε
ε	1.5ε

Therefore, we proved the claim that for models trained until a particular train loss ϵ, they could have different test losses due to undersampling/overparameterization. Further analysis could be built on this result to prove the behavior of generalization gap but it is beyond the scope and time requirements of this course.

4.3 Empirical Analysis

To prove this empirically, I take a simple example of linear regression in two dimensions. For a dataset of 10 examples randomly generated using Gaussian distribution of mean 0 and variance 1, I run the experiment 10 times to find the average difference between test losses obtained when trained with large learning rate of 0.1 and small learning rate of 0.01.

With a high enough probability of around 75%, the results given in the figure below show a better performance of the model in the case of large initial learning rate + anneal as compared to small learning rate only.



Test loss (RMSE) of large and small learning rate models (early stopped at train loss = 0.1)

Experiments	Small learning rate (0.01)	Large (0.1), then small learning rate (0.01)
Experiment 1	0.237	0.217
Experiment 2	0.242	0.228
Experiment 3	0.236	0.233
Experiment 4	0.229	0.231
Experiment 5	0.244	0.211
Experiment 6	0.254	0.249
Experiment 7	0.217	0.235
Experiment 8	0.249	0.251
Experiment 9	0.266	0.243
Experiment 10	0.22	0.213
	AVERAGE 0.2394	AVERAGE 0.2311

5 Discussion

This work surveyed the results of two interesting studies and verified their claims and results in different settings. Specifically, Li et al. have shown that the order in which a neural net learns to fit different types of patterns plays a crucial role in generalization. Moreover, Keskar et al. present several numerical experiments that support the claim that convergence to sharp minimizers gives rise to the poor generalization of large-batch methods for deep learning. Then, I drafted some theoretical analysis and verified this phenomenon in strongly convex problems, more specifically linear regression in two dimensions.

The results from these papers poses some questions: can one design neural network architectures for various tasks that are suitable to the properties of Large learning rate and Large batch methods? Can the networks be initialized in a way that enables these methods to succeed?

5.1 Future directions

To study this interesting phenomenon, the next steps would be to compare generalization performance of large initial learning rate model with adaptive learning rate methods, use batch normalization, experiment with other learning rate schedules as well as other data augmentation techniques in order to observe if there is any improvement in the generalization of large batch regimes and how this affects the sharpness of the training function using parametric plots of the training and test loss landscapes. In conclusion, this is an interesting direction to explore - to analyze how the selection of the most important hyper parameters affect the performance and scalability of deep neural networks.

References

[1] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[2] Robert Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? *CoRR*, abs/1802.06175, 2018.

[3] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks, 2018.

[4] Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A walk with sgd, 2018.

[5] Yeming Wen, Kevin Luk, Maxime Gazeau, Guodong Zhang, Harris Chan, and Jimmy Ba. Interplay between optimization and generalization of stochastic gradient descent with covariance noise. *CoRR*, abs/1902.08234, 2019.

[6] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *CoRR*, abs/1907.04595, 2019.

[7] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. *CoRR*, abs/1710.06451, 2017.

[8] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic gradient descent as approximate bayesian inference, 2018.

[9] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *CoRR*, abs/1812.06162, 2018.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.