Effect of Batch Normalization on Loss Landscape

Department of Mathematics
Oxford University
Candidate Number: 1062194

Abstract

Batch Normalization (BatchNorm) proposed by Ioffe and Szegedy in 2015 [1] is one of the most widely used techniques to successfully stabilize and accelerate training of deep neural networks (DNNs) by standardizing the inputs of activation functions in intermediate layers of DNNs. While the effects of Batch Normalization on optimization have been extensively studied, there are few researches on how it specifically affects the loss landscape. This mini-project aims to investigate the effect of Batch Normalization on the loss landscape of a neural network. We mainly discuss the research of ([2] Santurkar et al., 2018) and analyze the result by the method proposed by ([3] Li et al., 2017) for visualizing the loss landscape in order to demonstrate how BatchNorm affects it. Codes are available at Github.

1 Introduction

1.1 Batch Normalization

In general, numerical data points in a data set always have multiple features, each of them often ranges in different scale and units. This may result in rugged loss function with imbalance gradients. For the parameters related with low gradient, due to the nature of back-propagation, they will update slowly in comparison to others and even stop to update them, which may cause the low training speed. To solve this problem, Batch normalization layers are introduced layer-wise in an neural network.

Batch Normalization proposed by Ioffe and Szegedy in 2015 [1] is an effective technique for training neural networks. The Intuitive idea of the Batch Normalization is a kind of feature scaling that standardized the inputs of the next layer. It is achieved by inserting a new layer that receive the original inputs of the activation and control the mean and variance of the distributions of them. In particular, for a single neuron, if $y_1, y_2, \ldots, y_N$ are the outputs of the layer before BatchNorm in a mini-batch, then operation of BatchNorm can be described as follow:

$$BN(y_i) = \gamma \frac{y_i - \mu}{\sigma} + \beta$$

where $u = \frac{1}{N}\sum_{i=1}^{N} y_i$ and $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - u)^2$ are the mean and variance in one mini-batch ($\frac{y_i-\mu}{\sigma}$ is the whitened output), and parameters γ, β are two parameters need to be learned. The reasons for batch normalization optimize neural network training will be discussed in detail in section 2.

1.2 Loss Landscape

Loss function ([4] Girosi et al., 1995) is a pre-defined measure to evaluate the performance of an algorithm. Mathematically, it is a mapping from weight space (including bias) onto a real number, intuitively representing some 'cost' associated with certain parameter setting of a model. The concept of loss landscape is a geometric envision of loss function, which represents a surface in a parameter space with dimension equal to the number of parameters used in the network. In general, neural network can be easily holding thousands of parameters, in these cases, the space holds far more than 3 dimensions as in physical reality, making any visualization of the loss landscape rather difficult.

Although it is impossible for us to directly visualize what high-dimensional loss landscape looks like, there are a few methods which can help us to visualize the loss landscape in low dimensions ([3] Li et al., 2017). Li(2017) [3] mentioned that there are two typical methods: $1D$ Linear Interpolation and surface plot with random directions. More details, $1D$ Linear Interpolation is used to create a one-dimensional subspace by evaluating the loss function on points along a line connecting two weight vectors θ_1 and θ'. Surface plot involves plotting a two dimensional surface of the loss function values on all the parameter vectors spanned by two randomly chosen vectors μ and η added to some initial starting point θ_0.

These techniques are able to shed the light on some effects of various optimization techniques and network architectures on the loss function. However, there are limitations in these two methods: $1D$ plot is hard to conveyed some information such as non-convexity; for direct $2D$ plot with random basis, it may also fail to capture the complex non-convexity of loss surfaces in low-resolution plots due to computational burden.

To address these issues, Li(2017) [3] proposed a new visualization method–filter normalization, which is an improvement to previous $2D$ plotting. Instead of taking random basis, it plots loss landscape using filter-wise normalized directions to remove the scaling effect of parameters and better capture the intrinsic geometry of loss surfaces. As discussed in ([3] Li et al,. 2017), loss landscape plotted by filter normalization captures the relationship between sharpness of minimizers and generalization ability. Mathematically, we also can compute 'principle curvatures' and the 'flatness' of the minima which can be considered by the eigenvalues of the Hessian of the loss function evaluated at the minima being small for flat minima and large and positive for sharp minima. ([5] Keskar et al., 2017). Yurii Nesterov(2014) [6] also points out that the non-smooth loss landscape which has a large number of "kinks" makes gradient descent–based training algorithms unstable. So it is necessary to design the algorithm with smooth landscape.

2 Batch Normalization and the Loss Landscape

Following the work by Santurkar(2018) [2], we also denote layer weights as $W_{i,j}$, our original loss function shall be denoted by $\mathcal{L}$, and the loss function with the BatchNorm layer inserted is denoted as $\hat{\mathcal{L}}$. We use x denote the input into a layer with y being the output ($y = Wx$) and $y_i \in \mathbb{R}^m$ is denoted the i-th (neuron) output of some batch, and $\hat{y}$ is the whitened output. As Santurkar's discussion, Lipschitzness and "effective" β-smoothness[1] may capture the smoothness of loss function. We will dissect the effect of BatchNorm on them.

2.1 Batch Normalization and Lipschitzness of the Loss

Santurkar(2018) [2] claimed that the gradient magnitude captured the Lipschitzness of the loss. They also demonstrated that when a BatchNorm layer is added after a single fully connected layer W, a favourable upper bound on the related loss function is attained:

$$\left|\left|\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right|\right|^2 \leq \frac{\gamma^2}{{\sigma_j}^2}\left(\left|\left|\nabla_{\boldsymbol{y}_j}\mathcal{L}\right|\right|^2 - \frac{1}{m}\langle 1, \nabla_{\boldsymbol{y}_j}\mathcal{L}\rangle^2 - \frac{1}{\sqrt{m}}\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\rangle^2\right) \tag{1}$$

For $\langle 1, \nabla_{\boldsymbol{y}_j}\mathcal{L}\rangle^2$, it grows quadratically in the dimension of $\nabla_{\boldsymbol{y}_j}\mathcal{L}$, and thus it is significant. For $\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\rangle^2$, it may also be significant since the gradient of the loss calculated with respect to some variables is highly likely to be correlated to the variables themselves. Furthermore, for the scaling constant γ/σ_j, σ_j tends to be large in practice. Therefore, we can see that batch normalization provides a better Lipschitz constant of loss function than without it.

However, our loss landscape defined on the weight space. In order to explain how BatchNorm affects the loss landscape, we need to generalize formula (1) to the weight space. We now consider that

$$\frac{\partial\hat{\mathcal{L}}}{\partial W_{ij}} = \sum_{b=1}^{m}\frac{\partial\hat{\mathcal{L}}}{\partial y_j^{(b)}}\frac{\partial y_j^{(b)}}{\partial W_{ij}} \tag{2}$$

[1]The absolute, global bound of -smoothness will not be expected, we will only consider the change of gradients as we move in the direction of the gradients.

where $y_j^{(b)}$ denotes the b^{th} component of $\boldsymbol{y}_j$. Since $y = Wx$, $\frac{\partial y_j^{(b)}}{\partial W_{ij}}$ is equal to $x_i^{(b)}$. Then this sum can be written in form of inner product $\langle \nabla_{\boldsymbol{y}_j} \mathcal{L}, x_i \rangle$, and thus

$$\nabla_w \hat{\mathcal{L}} = X^T \nabla_{\boldsymbol{y}_j} \hat{\mathcal{L}}, \tag{3}$$

where input matrix $X \in \mathbb{R}^{m \times d}$ holding $X_{b_i} = x_i^{(b)}$. Additionally we suppose that $||X||_2 \leq \lambda$, and then $\left|\left|XX^T\right|\right|_2 \leq \lambda^2$, we have that

$$v^\top XX^T v \leq \lambda^2 \parallel v \parallel^2 \tag{4}$$

where equality is established if $XX^T = \lambda^2 I$. For the norm of gradient of the loss function with respect to the weights, we could deduce that it is bounded above by formula (4):

$$\left|\left|\nabla_w \hat{\mathcal{L}}\right|\right|^2 = \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}^T XX^T \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j} \leq \lambda^2 \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}^T \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j} = \lambda^2 \left|\left|\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right|\right|^2, \tag{5}$$

where equation holds if $XX^T = \lambda^2 I$, that is

$$\max_{\|X\|_2 \leq \lambda} \left|\left|\nabla_W \hat{\mathcal{L}}\right|\right|^2 = \lambda^2 \left|\left|\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right|\right|^2. \tag{6}$$

Similarly, we consider the same process for the vanilla network and get that

$$\max_{\|X\| \leq \lambda} ||\nabla_W \mathcal{L}||^2 = \lambda^2 \left|\left|\nabla_{\boldsymbol{y}_j} \mathcal{L}\right|\right|^2. \tag{7}$$

Now we plugging formula (5) and (6) into (1), we have

$$\begin{aligned}\max_{\|X\| \leq \lambda} \left\|\nabla_{\mathbf{w}} \hat{\mathcal{L}}\right\|^2 &\leq \frac{\gamma^2}{\sigma_j^2} \lambda^2 \left(\left\|\nabla_{\boldsymbol{y}_j} \mathcal{L}\right\|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{\boldsymbol{y}_j} \mathcal{L}\right\rangle^2 - \frac{1}{\sqrt{m}}\left\langle \nabla_{\boldsymbol{y}_j} \mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right) \\ &= \frac{\gamma^2}{\sigma_j^2}\left(\max_{\|X\| \leq \lambda} \|\nabla_{\mathbf{W}} \mathcal{L}\|^2 - \lambda^2 \frac{1}{m}\left\langle \mathbf{1}, \nabla_{\boldsymbol{y}_j} \mathcal{L}\right\rangle^2 - \lambda^2 \frac{1}{\sqrt{m}}\left\langle \nabla_{\boldsymbol{y}_j} \mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right).\end{aligned} \tag{8}$$

From formula (8), we can find that under the restriction of the norm of inputs, the loss landscape over the weight-space for a network with a BatchNorm layer will have a smaller maximum gradient than that without a BatchNorm layer. That is, Batch Normalization improves the bound (in worst-case) of gradient for the weight-based optimization problem, which makes the training process more stable([2] Santurkar et al., 2018).

2.2 Batch Normalization and β-smoothness

Now we turn our attention to the second-order gradient of loss. Recall the Taylor series expansion of the loss with respect to activation y:

$$\mathcal{L}(y + \Delta y) = \mathcal{L}(y) + \nabla \mathcal{L}(y)^\top \Delta y + \frac{1}{2}(\Delta y)^\top H(\Delta y) + o\left(\|\Delta y\|^2\right) \tag{9}$$

where H is the Hessian matrix of the loss, and in our case δy is a step in the gradient direction (we only consider the "effective" β-smoothness). we could see that the quadratic form of the loss Hessian captures the second order term of the Taylor expansion surrounding current point. Thus reducing this term, implies the improved β-smoothness and more predictive gradient. Then the gradient increases or decreases at a slower rate and thus allows the training process to use larger range of learning rates and accelerates network convergence.

According to Santurkar(2018) [2], the quadratic form of the loss Hessian inserted (a single) BatchNorm layer with respect to the activations $\boldsymbol{y}_j$ in the gradient direction is bounded above:

$$\left(\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}\left(\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2}\left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right)^\top \boldsymbol{H}_{jj}\left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right) - \frac{\gamma}{m\sigma^2}\left\langle \nabla_{\boldsymbol{y}_j} \mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle \left\|\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right\|^2. \tag{10}$$

Furthermore, if $\boldsymbol{H}_{jj}$ preserves the relative norms of $\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}$ and $\nabla_{\boldsymbol{y}_j}\mathcal{L}$, $\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}$ can be replaced by $\nabla_{\boldsymbol{y}_j}\mathcal{L}$ in the first term on the right hand side. That is

$$\left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)^\top \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}\left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j}\right)^\top \boldsymbol{H}_{jj}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j}\right) - \frac{\gamma}{m\sigma^2}\left\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle \left\|\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right\|^2. \quad (11)$$

Similar with Section 2.1, this result can also be generalized to the weight space and we can get an upper bound in case of the worst inputs scenario under some mild assumptions. By the fact that

$$\nabla_W^2 \hat{\mathcal{L}} = \frac{\partial \hat{\mathcal{L}}}{\partial W_{\cdot,j}\partial W_{\cdot,j}} = X^T \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j} X = X^T\left(\nabla^2_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)X \quad (12)$$

and by formula (3), we have

$$\left(\nabla_W \hat{\mathcal{L}}\right)^T \left(\nabla_W^2 \hat{\mathcal{L}}\right)\left(\nabla_W \hat{\mathcal{L}}\right) = \left(\nabla_{\boldsymbol{y}_\mathbf{j}}\hat{\mathcal{L}}\right)^T XX^T \left(\nabla^2_{\boldsymbol{y}_\mathbf{j}}\hat{\mathcal{L}}\right) XX^T \left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right). \quad (13)$$

Now, we also set constraint that $\| \ X \ \|_2 \leq \lambda$, and then $\| \ XX^T \ \|_2 \leq \lambda^2$. The formula (13) turns that $\left(\nabla_W \hat{\mathcal{L}}\right)^T \left(\nabla_W^2 \hat{\mathcal{L}}\right)\left(\nabla_W \hat{\mathcal{L}}\right) \leq \lambda^4 \left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)^T \left(\nabla^2_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)\left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)$, where the equation holds if $XX^T = \lambda^2 I$, that is

$$\max_{\|X\|\leq\lambda}\left(\nabla_W \hat{\mathcal{L}}\right)^T \left(\nabla_W^2 \hat{\mathcal{L}}\right)\left(\nabla_W \hat{\mathcal{L}}\right) = \lambda^4 \left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)^T \left(\nabla^2_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right)\left(\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right). \quad (14)$$

Plugging formula (14) into formula (10), that we assume H_{jj} preserves the relative norms of $\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}$ and $\nabla_{\boldsymbol{y}_j}\mathcal{L}$, we get that

$$\max_{\|X\|\leq\lambda}\left(\nabla_W \hat{\mathcal{L}}\right)^T \left(\nabla_W^2 \hat{\mathcal{L}}\right)\left(\nabla_W \hat{\mathcal{L}}\right) \quad (15)$$

$$\leq \frac{\gamma^2}{\sigma^2}\left[\max_{\|X\|\leq\lambda}\left\{(\nabla_\mathbf{w}\mathcal{L})^T\left(\nabla^2_\mathbf{w}\mathcal{L}\right)(\nabla_\mathbf{w}\mathcal{L})\right\} - \frac{\lambda^4}{m\gamma}\left\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle \left\|\nabla_{\boldsymbol{y}_j}\hat{\mathcal{L}}\right\|^2\right] \quad (16)$$

Santurkar(2018) [2] takes fairly mild assumptions that $\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\rangle$ and the quadratic forms involving the Hessian are non-negative, where $\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\rangle$ is positive if negative $\nabla_{\boldsymbol{y}_j}\mathcal{L}$ points towards the minimum of the loss function. Then formula (16) implies that Batch Normalization improves the bound (in worst-case) of quadratic form of the loss Hessian for the weight-based optimization problem, which improves "effective" β-smoothness with respect to weights and enables more predictive gradient ([2] Santurkar et al., 2018).

3 Numerical Simulation

In this section, we illustrate the experimental setting and analyze the main result of some Numerical Simulation. Inspired by the work of Santurkar(2018)[2], we build LeNet and VGG-16 Network with and without Batch Normalization layers separately by PyTotch. Cafir-10 dataset ([7] Krizhevsky, 2009) has been chosen in our experiments as the training the testing data set and all simulations use the Adam optimizer ([8] Kingma and Ba, 2017) and cross-entropy loss to train networks for 20 epochs. During training, batch size is set as 128 and learning rate is 0.01. Then we measure the variation of loss in the gradient direction and "effective" β-smoothness in each training step, and plot them in Figure 1.

Moreover, we use filter normalization method ([3] Li et al., 2017) to visualize the loss landscape of LeNet in Figure 2. We also find that there are extremely large values at the boundary points, which leads to large scale in figure and makes us hard to figure out the gradient in the central area of loss. To address this issue, we need to log scale the surface. The result also visualized in Figure 2.

Focusing on the different scaling in Figure 2, we could get that the variance of loss landscape with BatchNorm is smaller than that without BatchBorm, which also implies Batch Normalization flattens the loss landscape and improve the Lipschitzness. Moreover, we also see that Batch Normalization improves the fluctuation of loss landscape and make it more concave. This also means that Batch Normalization improve the β-smoothness. Therefore, Batch Normalization smoothen the loss landscape and make the neural network easy to train.

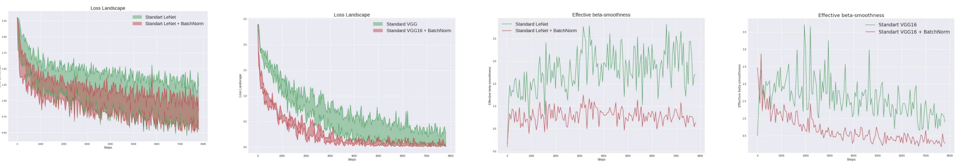

Figure 1: the first two plots represent the variation of loss of LeNet and VGG16 with and without Batch Normalization layers during training and remain plots visualize the "effective" β-smoothness of them. We could see that the gradient of LeNet with BatchNorm is more stable (smaller variation) and predictive (smaller "effective" β-smoothness) compared with that without BatchNorm. Images of VGG16 also reveal similar phenomenons.

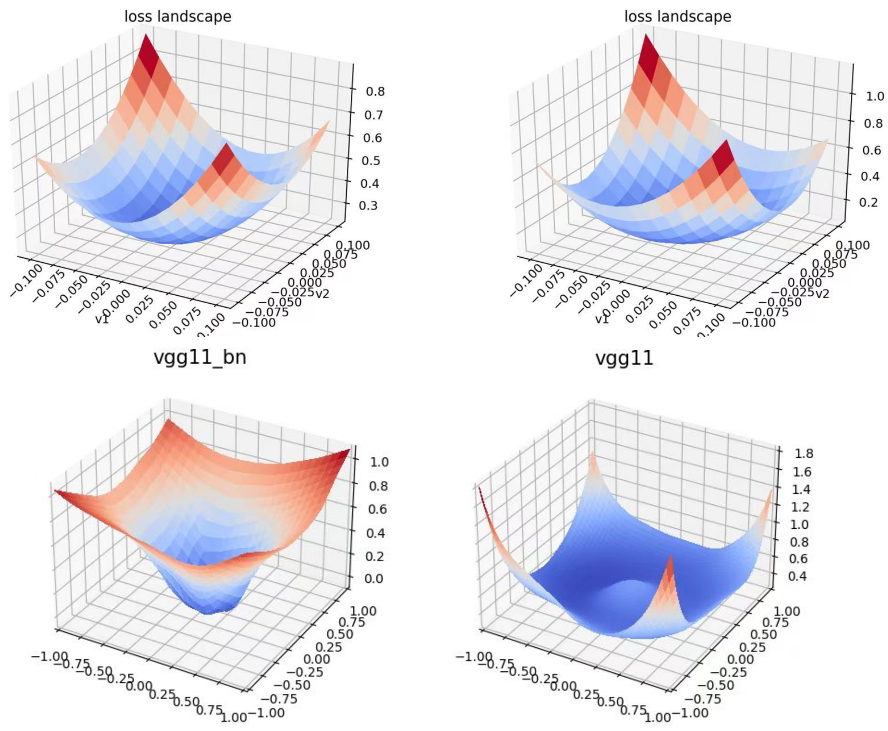

Figure 2: the first two images visualize the variation of loss landscape of LeNet after inserting Batch Normalization layers. we notice there is different scaling on the training loss axis, which implies the loss landscape subspace with BatchNorm is much flatter. Such flat surface also suggests Batch Normalization improve the Lipschitzness. The last two images refer to the surface of loss landscape of VGG 16 in log scale.

4 Conclusion

In this report, we dissect the effect of Batch Normalization on Loss landscape and find that Batch Normalization improve the Lipschitzness and 'effective' β-smoothness (in worst input case). Besides measuring the variance and 'effective' β-smoothness of loss landscape, we also implement new numerical simulation, filter normalization, to observe this phenomenon. Nevertheless, Batch Normalization may also affect other aspects, such as initialization, to optimize the training process. We could discuss and dissect them in the future.

References

[1] Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

[2] Santurkar, S., Tsipras, D., Ilyas, A. and Mądry, A., 2018, December. How does batch normalization help optimization?. In Proceedings of the 32nd international conference on neural information processing systems (pp. 2488-2498).

[3] Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., 2017. Visualizing the loss landscape of neural nets. arXiv preprint arXiv:1712.09913.

[4] Girosi, F., Jones, M. and Poggio, T., 1995. Regularization theory and neural networks architectures. Neural computation, 7(2), pp.219-269.

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

[6] Nesterov, Y., 2003. Introductory lectures on convex optimization: A basic course (Vol. 87). Springer Science Business Media.

[7] Krizhevsky, A. and Hinton, G., 2009. Learning multiple layers of features from tiny images.

[8] Kingma, D.P. and Ba, J., 2017. Adam: a method for stochastic Opoimization.