CS269 Project 2 Report: A Discussion Over Gradient-based Optimization Algorithms

1 Introduction

This is the report for my codes which serves as a project to investigate the effects and impacts of different approaches to gradient based optimization, using the `PyTorch` deep learning library. This library, by providing these approaches with optimized matrix operations, allows them to efficiently converge to the good points on the surfaces leading to a high accuracy. In this report, first I will go through the theoretical background of using these modules, then explain my codes as well.

My submission in this project is written in an **object-oriented** fashion which allows the user to choose the dataset and customizations of their own, and run the experiments with a variety of optimization schemes (many more than requested in project description).

2 Gradient-based Optimization

In this part the feasible gradient based optimization schemes for tuning the parameters in the neural networks are investigated briefly, so as to know their advantages and disadvantages. In this we will use the material in [2] and [3] as well. First, we will start with the batch gradient descent.

Note that the gradient descent, in general, aims to minimize the value of an objective function (a loss function, since I mentioned minimization) which is defined over the domain of values for model's parameters ($\Theta \in \mathbb{R}^d$). It does so by taking steps in the opposite of gradient's direction, which we know from calculus and Newton's method, etc. that will tend to find a local minima for the surface.

2.1 Batch Gradient Descent

This approach is otherwise known as the basic gradient descent or vanilla gradient descent, and in this approach the whole dataset should fit in the memory and will be used to compute the gradient. A learning rate η is used to gauge the extent to which the step to be taken should be long. And the parameters of our model will be updated with respect to the gradient of the loss function:

$$\Theta = \Theta - \eta \dot{\nabla}_{\Theta} J(\Theta)$$

Since neither MNIST nor CIFAR is small enough for this method, it is not investigated here. The point worth mentioning about the batch gradient descent is that it is first of all very slow and requires the dataset to be fit in the memory, and as an advantage it is guaranteed to converge to global minimum for convex surfaces and to a local minima for the surfaces that are non-convex.

2.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) can be considered a more flexible version of the Batch Gradient Descent. For each item, the parameter update process takes place. The advantage of this model is this *jumping* mechanism which allows it to jump from a local minima to a path leading to a better minima, however, the level flucutations in general is higher since the updates are of high variance.

2.3 Mini-batch SGD

In this part, the main power of SGD is used in a bit more controlled fashion, allowing the model to work with mini-batches and iterate over them. This imposes some problems and challenges that have to be addressed. One of them is that choosing the proper learning rate to achieve a convenient pace of convergence might be a cumbersome task to carry out. Also, learning rate scheduling cannot account for that and counteract these negative effects efficiently, since it is a deterministic process independent of the dataset. Also, the same update happens for both frequent and infrequent features. Also, saddle points and their attributes might cause the algorithm to get stuck.

2.4 Momentum-based SGD

Imagin we throw a small ball in a bowl-shaped valley. The idea is that when the direction is relevant, the ocillations will be dampened by considering the consistency of the direction and being accelerated in that direction towards a local minima. Therefore, first a momentum is computed as follows:

$$\upsilon_t = \upsilon_{t-1}\gamma + \eta \nabla_{\Theta} J(\Theta)$$

Then the parameter update is performed:

$$\Theta = \Theta - \upsilon_t$$

2.5 Nesterov-Momentum

Now the ball will be even smarter with being augmented with a prescience, which manifests itself as follows:

$$\upsilon_t = \upsilon_{t-1}\gamma + \eta\nabla_\Theta J(\Theta - \gamma\upsilon_{t-1})$$

Then the parameter update is performed:

$$\Theta = \Theta - \upsilon_t$$

2.6 Adagrad

This algorithm adapts the learning rate to the parameters so as to perform larger updates for the infrequent features.

Say the G_t is a matrix which contains the sum of the squares of the past gradients with respect to all the parameters, the parameter update will be changed as follows:

$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

This, mainly eliminates the need for tuning the learning rate since based on the past gradients it will now be tuned. The problem, however, is that it is a bit too aggressive and damps the update eventually.

2.7 Adadelta

To reduce the Adagrad's aggressiveness, this algorithm is proposed to use the expected values instead of the previous approach. It works as follows:

$$\Delta\Theta_t = -\frac{\eta}{\text{RMS}(g)_t} g_t$$

The RMS is the square root of a value of $E[g^2]_t$ which is computed with exponentially weighted moving average.

2.8 RMSProp

This unpublished algorithm is mainly AdaDelta with 0.9 as the value of the contribution of the previous step in computing the expected value.

2.9 RProp

This algorithm, otherwise known as *resilient backpropagation*, mainly works with the sign change of the gradient to tune the step by decreasing it with a decay factor η if the sign change appears (therefore meaning that the changes are less certain), and also increasing it with another factor if the sign change did not happen [1].

2.10 Adam

This method, Adaptive Moment Estimation, is another method that requires no advanced learning rate setting in the beginning. In addition to what AdaDelta does, it keeps an average of past gradients as well. If we consider the ball example, this adds friction to it too so it will like flat surfaces more than slopes.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\upsilon_t = \beta_2 \upsilon_{t-1} + (1 - \beta_2) g_t^2$$

Then the bias will be removed:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1 - \beta_2^t}$$

Then the update takes place:

$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon} \hat{m}_t$$

This method performs really well in practice and is experts' favorite in many applications.

2.11 Adamax

This method is the result of using infinite norm (max) for the gradient, and it is useful as well.

3 Transform

The normalization that has been discussed in the project description is there to assist us with making it simpler for the model to learn the distribution. The experiments are performed with and without this transformation.

4 Implementation

For the implementation we have used a combination of PyTorch (a famous deep learning library by google which provides us with many optimized matrix functionalities used by the optimization schemes). For plotting the results, I used the `TensorboardX` package which serves as the binding for PyTorch and Tensorflow's powerful Tensorboard functionality, allowing the user to dynamically investigate the curves and paths. The results for the tensorboard are also shared in my codes and can be verified.

5 Results

The graphs and results for the running loss computed at every 100 minibatch are depicted in the figures. Note that the figures are generated using the Tensorboard with the smoothing factor of 0.6, therefore the fluctuations are somehow a bit less than the actual curves for better demonstration of path. Also for the loss function the MSELoss is used as well. For any further details please check out my code which is written as concisely as I could possibly write.

As for the observation, the `accuracies.txt` file the results are printed. The bottomline is that having the normalization in the data feeding part makes it generally easier for the algorithm to converge, and the accuracy is usually slightly higher than the without the transform case.

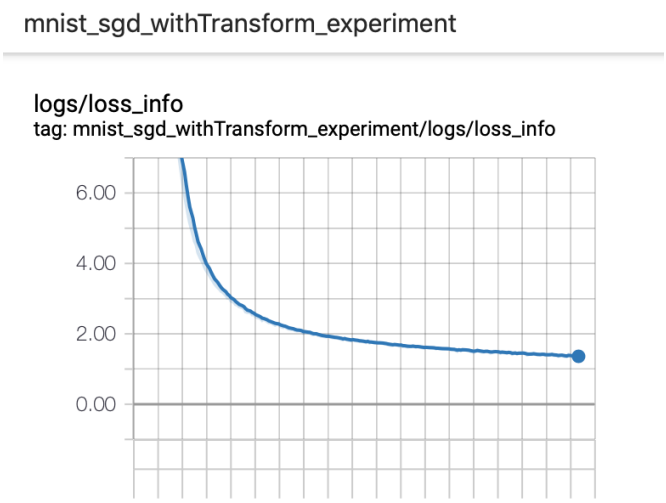

Figure 1: MNIST SGD - With Transform

References

[1] Rprop. Wikipedia, Wikimedia Foundation, 26 Mar. 2019, en.wikipedia.org/wiki/Rprop. Accessed 25 Apr. 2019.

[2] Sebastian Ruder. An Overview of Gradient Descent Optimization Algorithms. Sebastian Ruder, Sebastian Ruder, 29 Nov. 2018, ruder.io/optimizing-gradient-descent/.

[3] CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/neural-networks-3/.

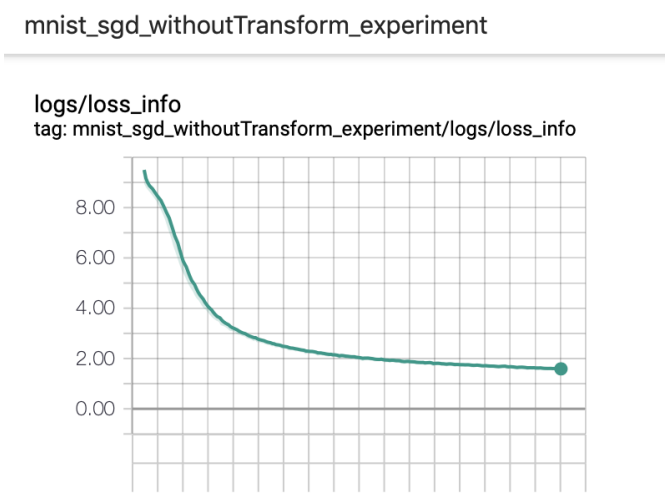

Figure 2: MNIST SGD - Without Transform

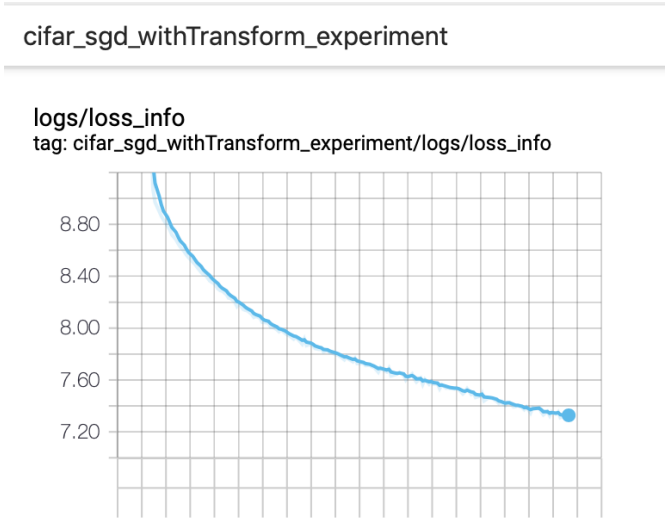

Figure 3: CIFAR SGD - With Transform

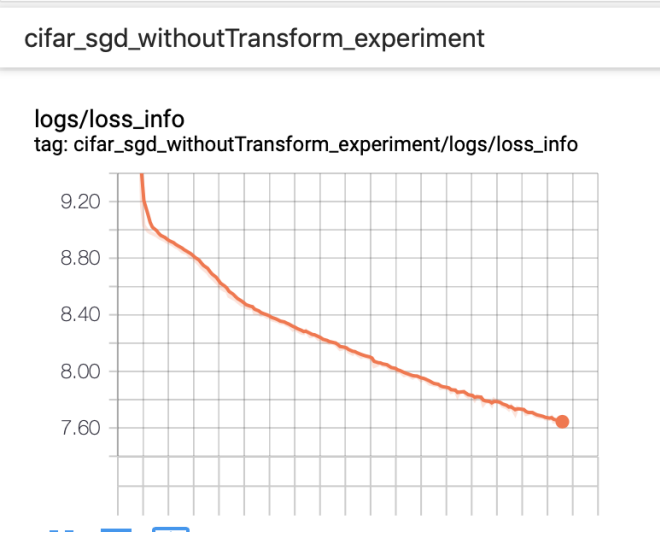

Figure 4: CIFAR SGD - Without Transform

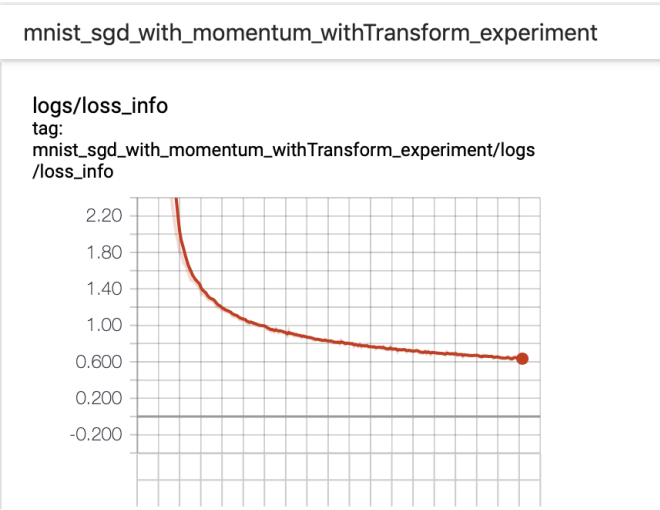

Figure 5: MNIST SGD with Momentum - With Transform

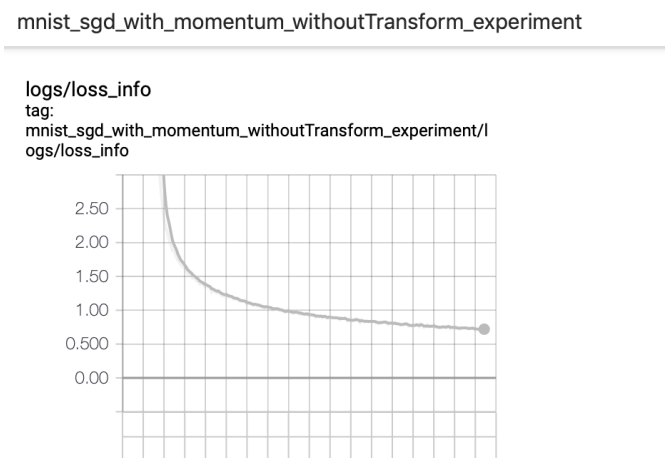

Figure 6: MNIST SGD with Momentum - Without Transform

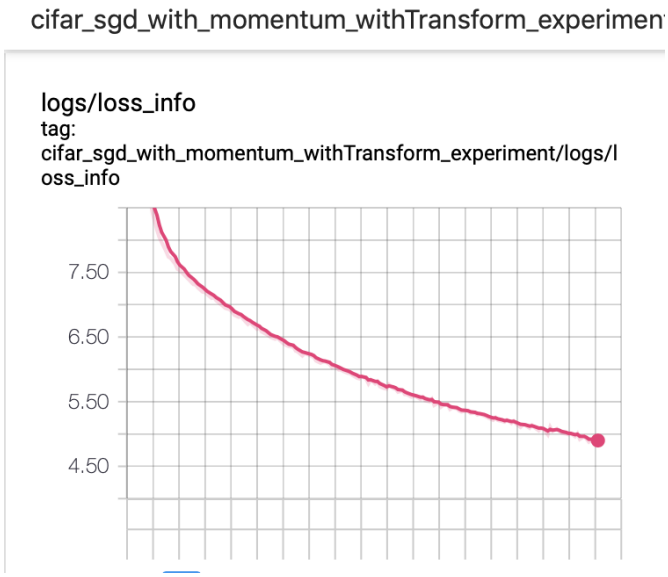

Figure 7: CIFAR SGD with Momentum - With Transform

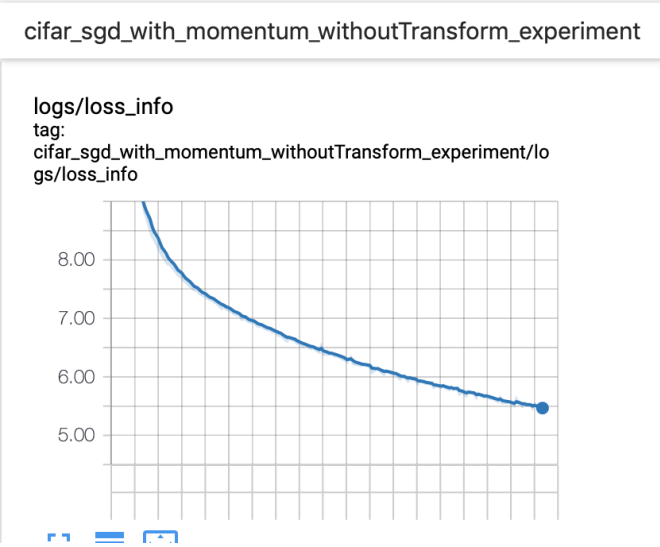

Figure 8: CIFAR SGD with Momentum - Without Transform

Figure 9: MNIST Adagrad - With Transform

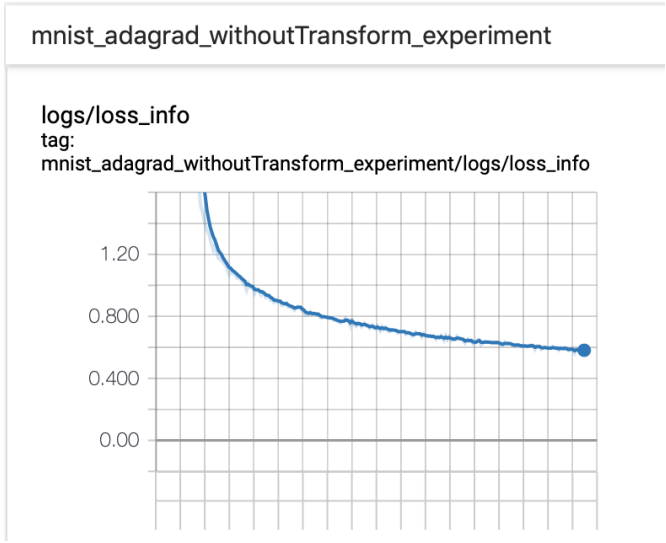

Figure 10: MNIST Adagrad - Without Transform

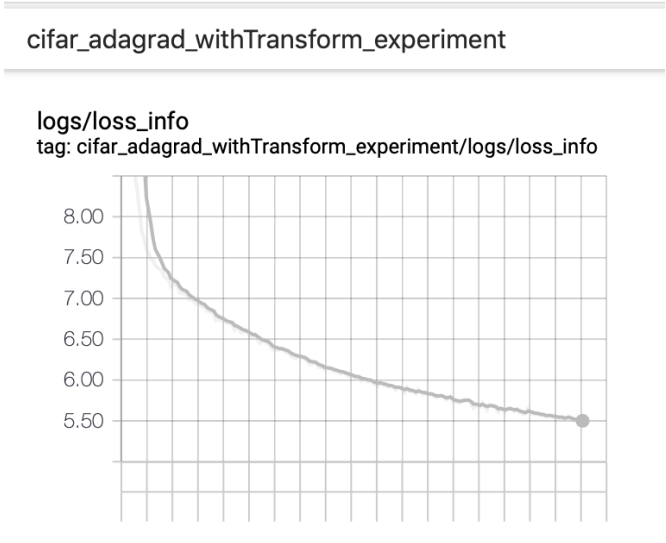

Figure 11: CIFAR Adagrad - With Transform

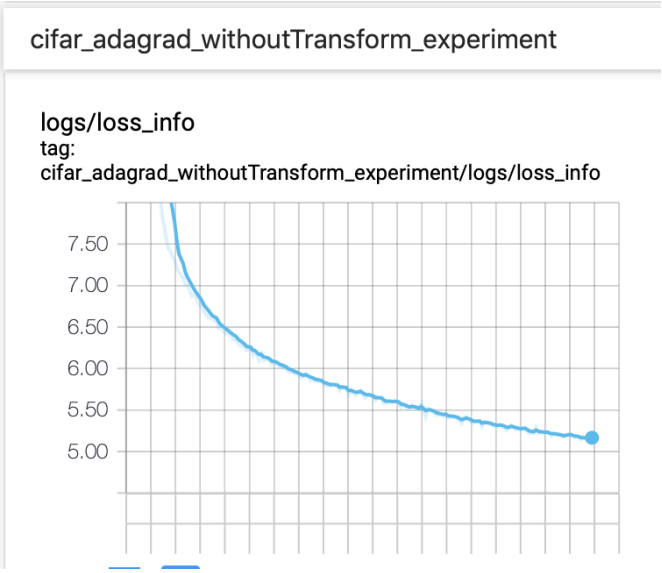

Figure 12: CIFAR Adagrad - Without Transform

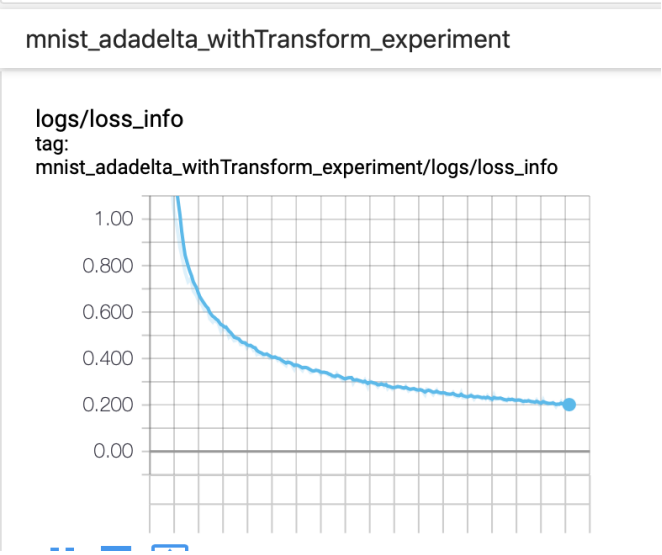

Figure 13: MNIST AdaDelta - With Transform

Figure 14: MNIST AdaDelta - Without Transform

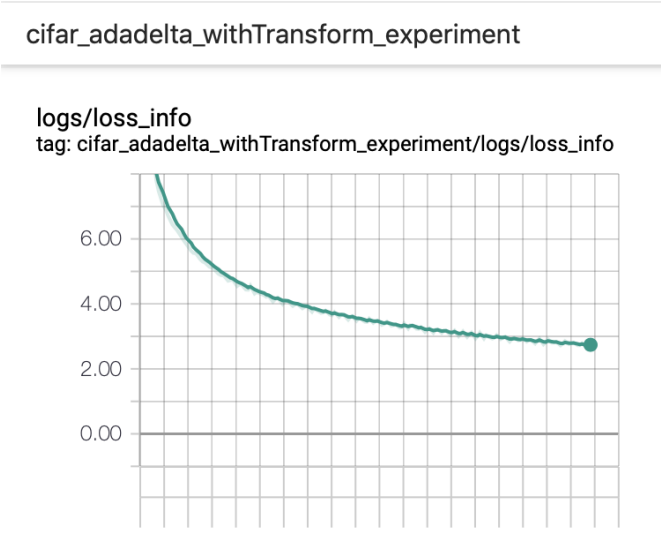

Figure 15: CIFAR AdaDelta - With Transform

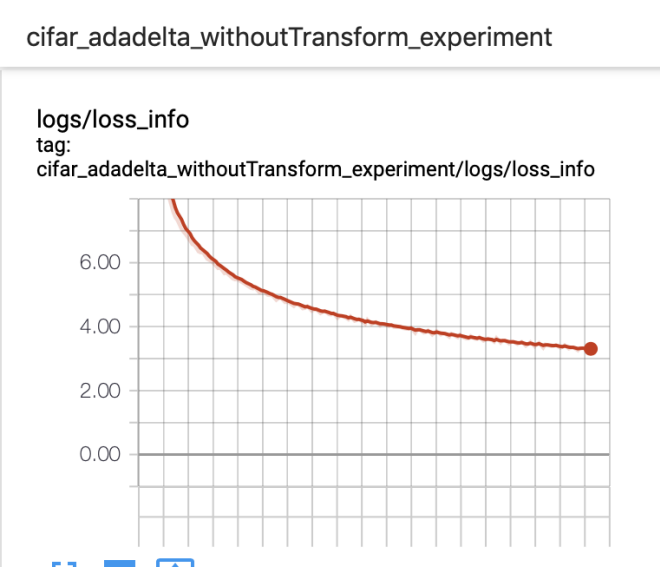

Figure 16: CIFAR AdaDelta - Without Transform

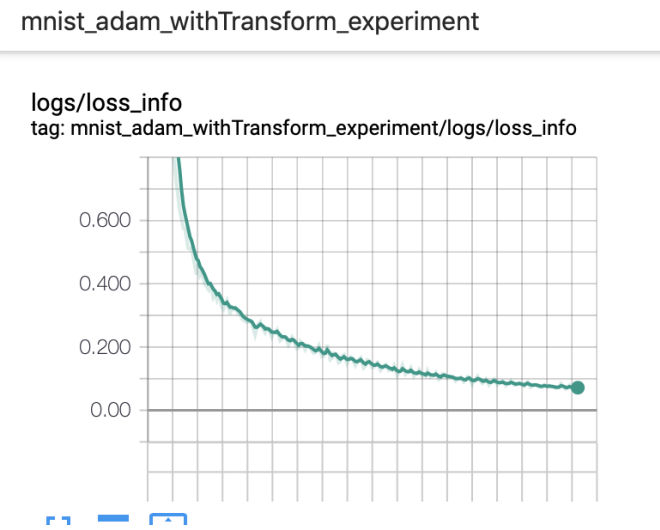

Figure 17: MNIST Adam - With Transform

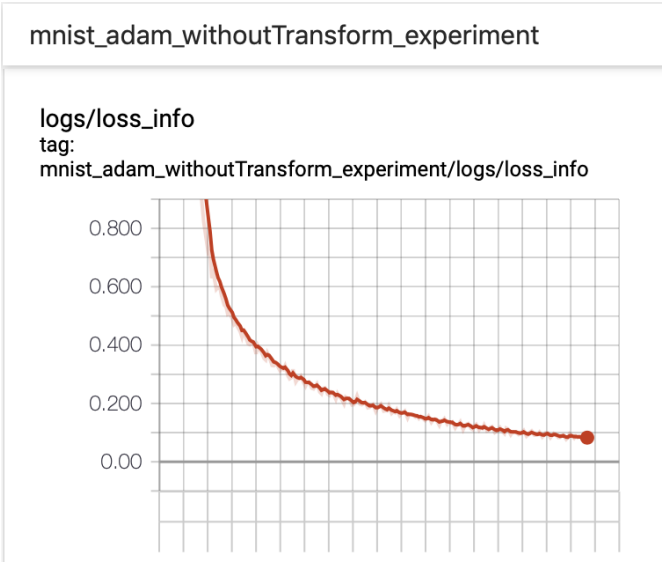

Figure 18: MNIST Adam - Without Transform

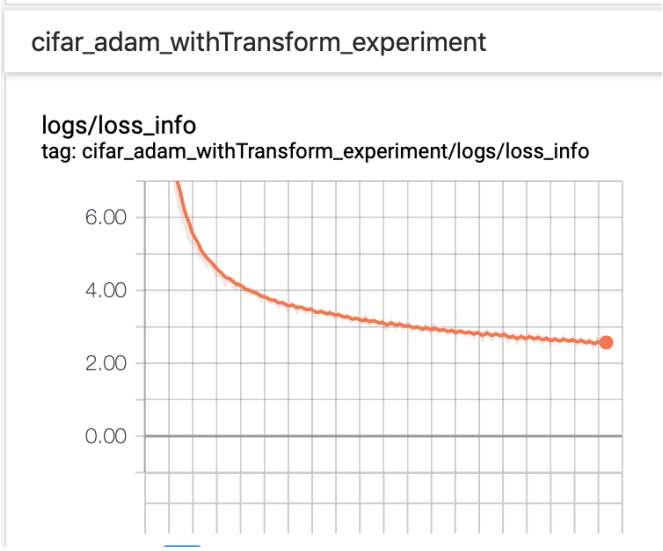

Figure 19: CIFAR Adam - With Transform

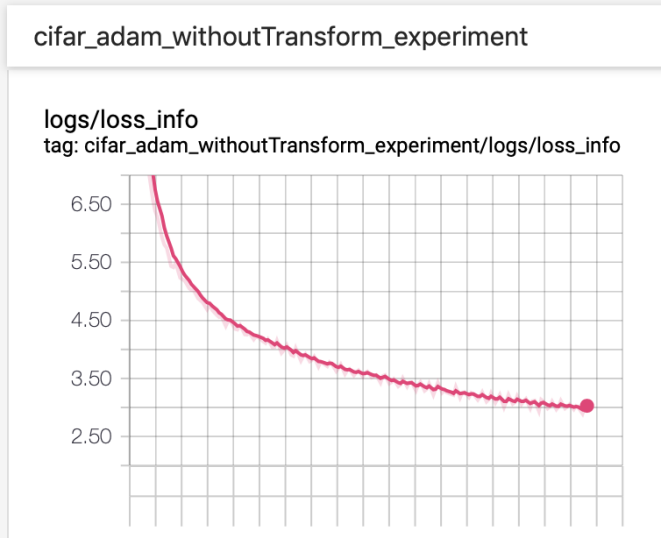

Figure 20: CIFAR Adam - Without Transform

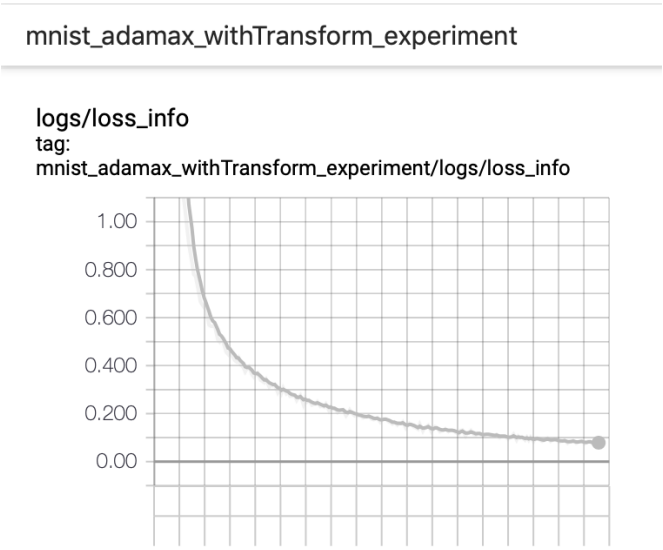

Figure 21: MNIST Adamax - With Transform

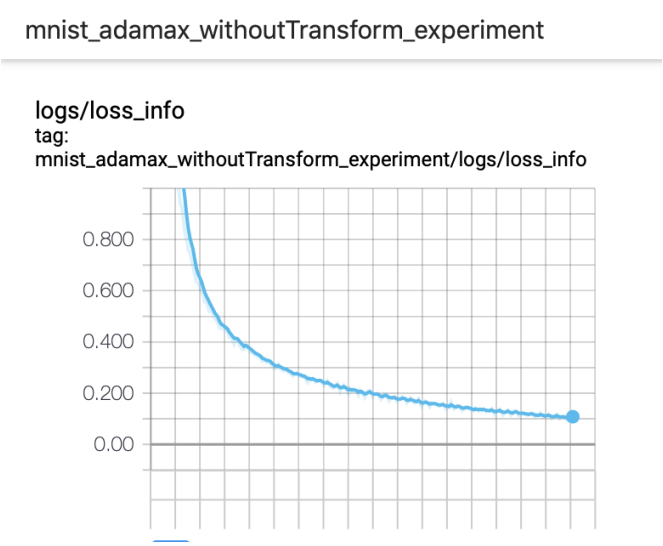

Figure 22: MNIST Adamax - Without Transform

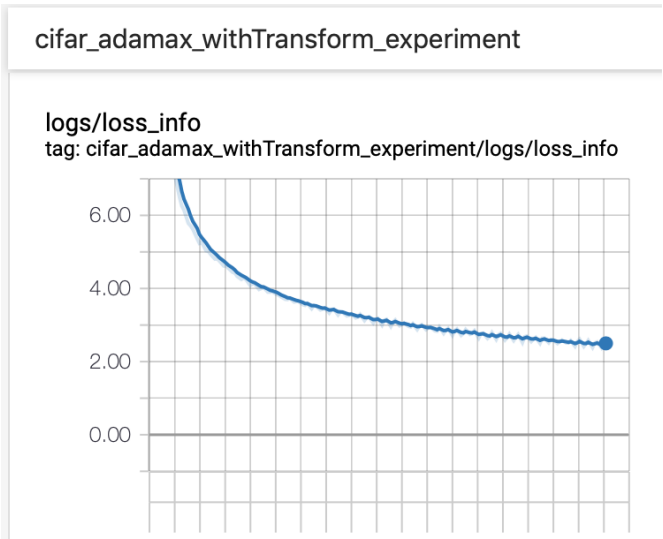

Figure 23: CIFAR Adamax - With Transform

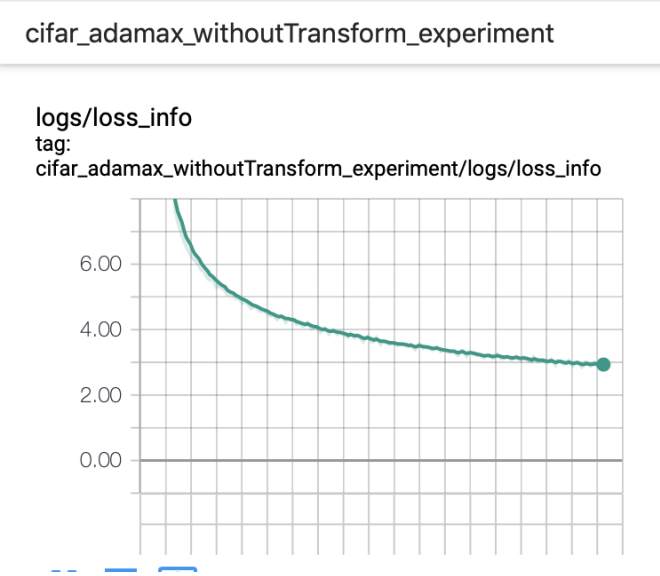

Figure 24: CIFAR Adamax - Without Transform

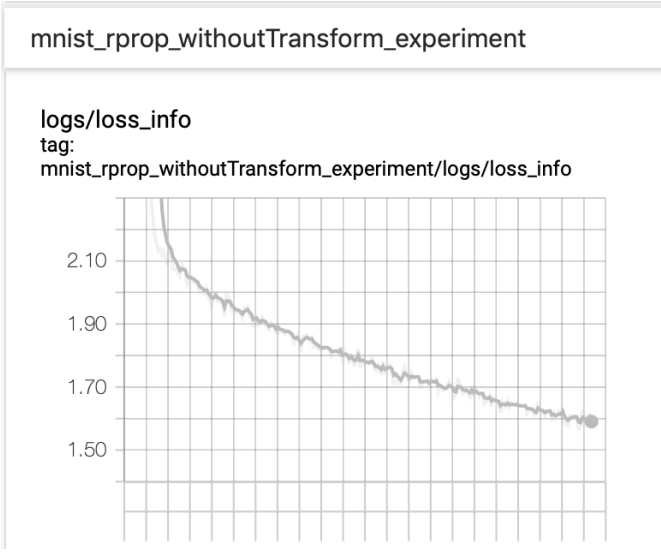

Figure 26: MNIST RProp - Without Transform

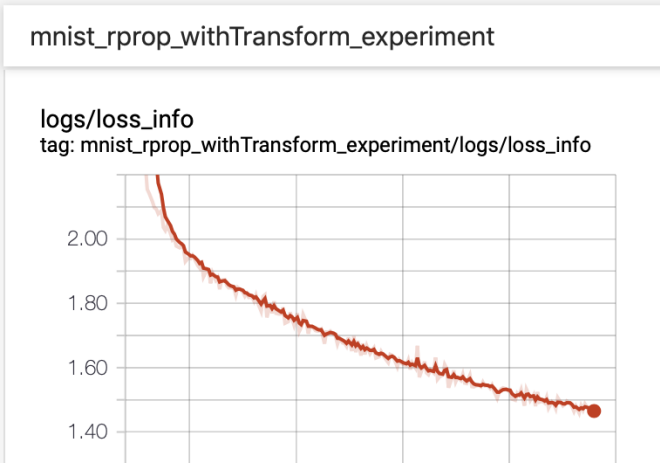

Figure 25: MNIST RProp - With Transform

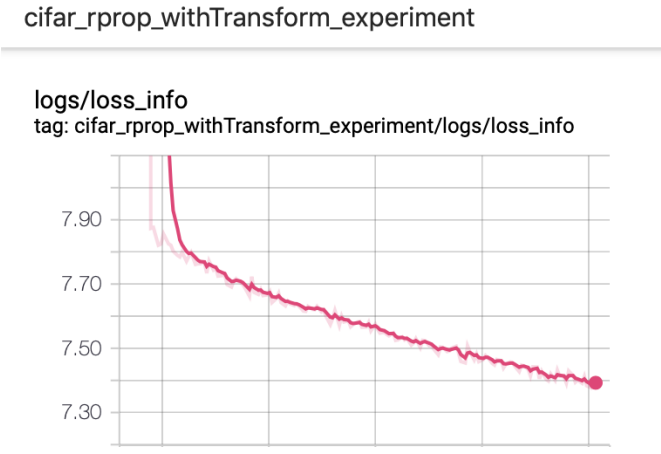

Figure 27: CIFAR RProp - With Transform

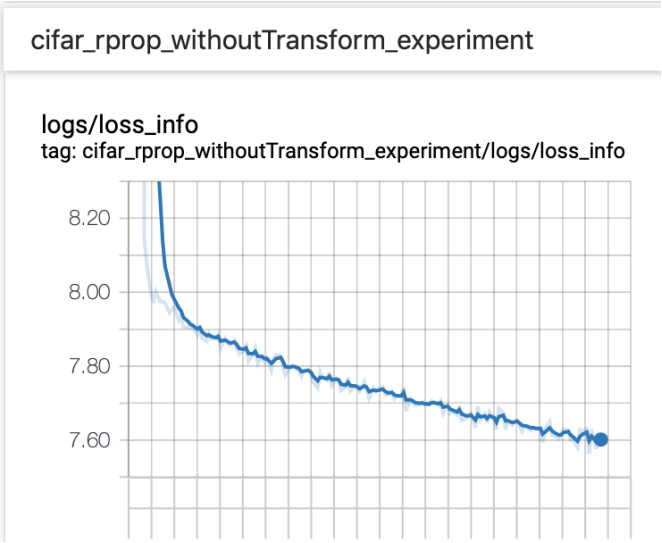

Figure 28: CIFAR RProp - Without Transform

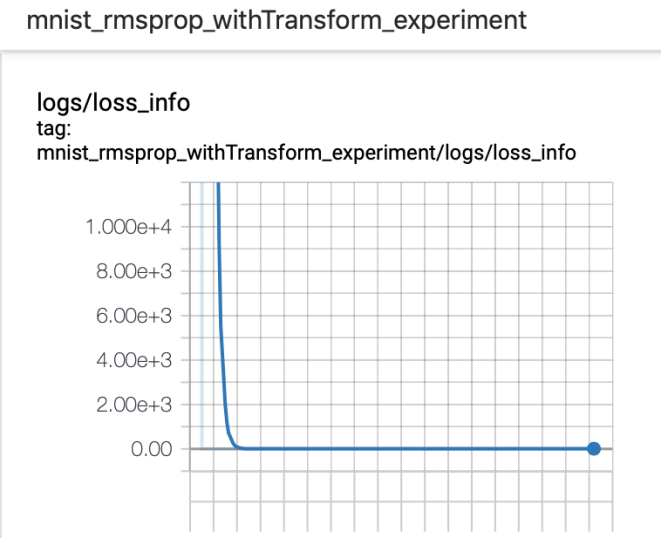

Figure 29: MNIST RMSProp - With Transform

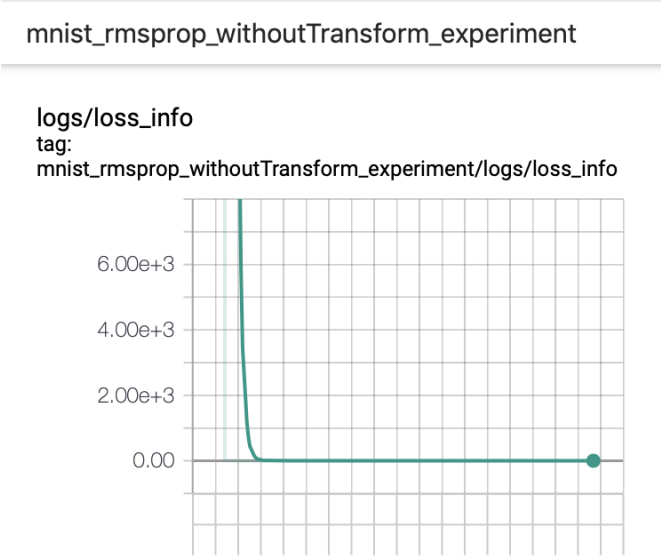

Figure 30: MNIST RMSProp - Without Transform

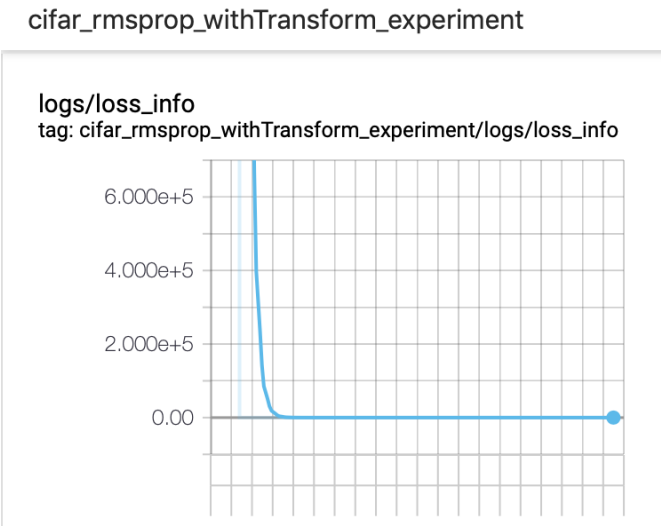

Figure 31: CIFAR RMSProp - With Transform

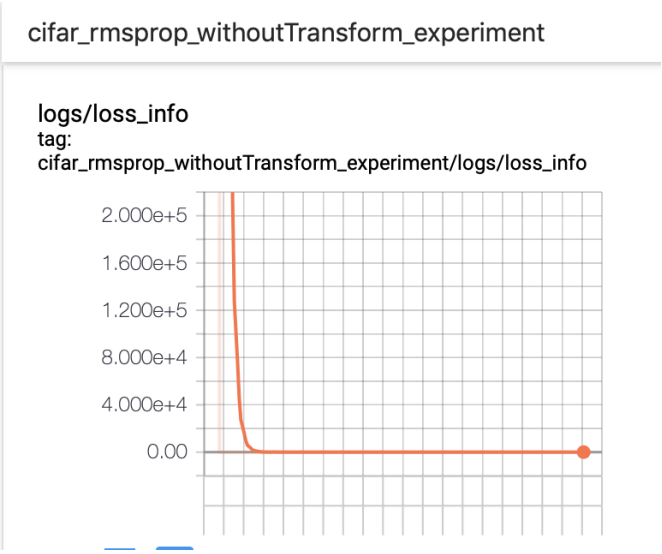

Figure 32: CIFAR RMSProp - Without Transform

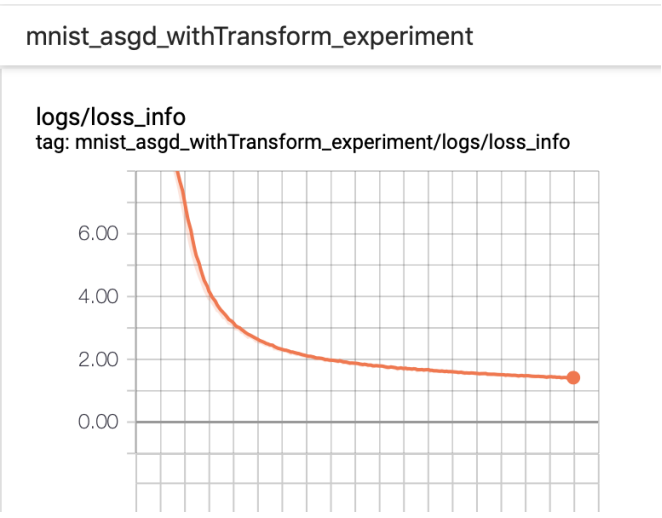

Figure 33: MNIST ASGD - With Transform

Figure 34: MNIST ASGD - Without Transform

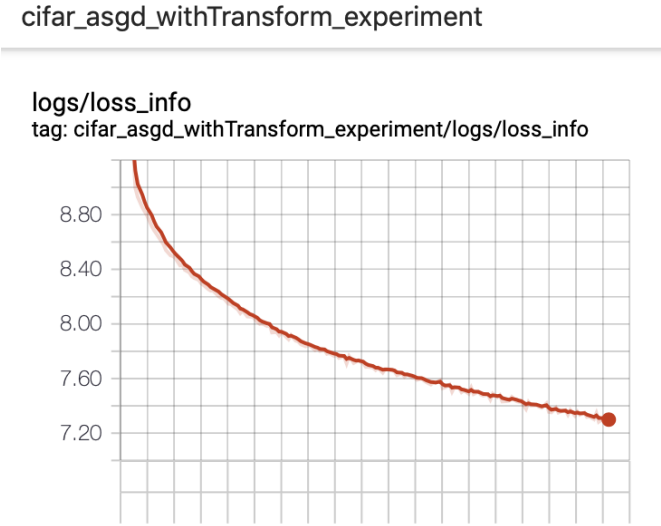

Figure 35: CIFAR ASGD - With Transform

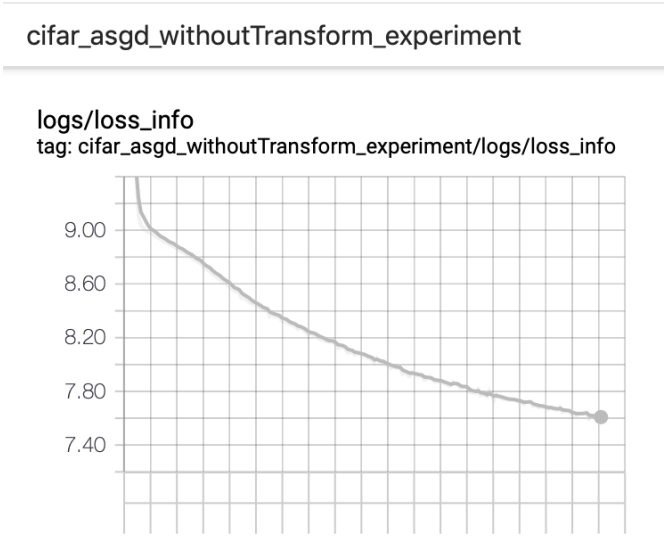

Figure 36: CIFAR ASGD - Without Transform