COMP6248 REPRODUCIBILITY CHALLENGE: ACCELERATING SGD WITH MOMENTUM FOR OVER-PARAMETERIZED LEARNING

Ethan Young, Conor Chalcroft, Ahmad Mustafa Kida
Electronics and Computer Science, University of Southampton
{ey1g18,cc6u21,ak8g18}@soton.ac.uk

ABSTRACT

This report aims to replicate a new optimizer called Momentum Added Stochastic Solver (MaSS) for overparmeterized learning. The authors of the original paper claim that MaSS addresses the non-convergence issue of the widely used Nesterov SGD optimizer by adding a compensation term. They postulate that MaSS achieves faster convergence rates compared against a range of optimizers. We test MaSS across a range of neural network architectures, and the non-convex Rastrigin function.

1 INTRODUCTION

The standard momentum accelerated SGD optimizer and its Nesterov variant are popular choices in training modern neural networks. In momentum accelerated SGD, the optimizer accumulates a velocity vector in the optimal convergence direction. It then adds this velocity vector, ν_t, to the gradient before updating the parameters, θ, that try to minimize the objective function. The Nesterov variant not only adds the velocity vector to the gradient, but computes the gradient of the objective function with the velocity vector added onto it (i.e $\nabla f(\theta + \nu_t)$). The intuition behind this is to more abruptly force the updates in the direction of the optimal parameters that minimise the objective function. To ensure convergence, the optimal step size must be much smaller than that of SGD, where the effect of the momentum vector is negated - the non-acceleration effect. MaSS attempts to solve the non-convergence issue by adding a compensation term, allowing for convergence with the same step sizes as SGD.

In this paper, we completely re-implement the MaSS optimizer using PyTorch, as well as the neural network architectures that were experimented on in the original paper. We test MaSS against relevant optimizers: SGD+momentum; SGD+Nesterov; and Adam; and extend our analysis on MaSS by visualising its performance on the Rastrigin loss surface. The code for this project can be found at `https://github.com/COMP6248-Reproducability-Challenge/MaSS-optimiser`.

2 RASTRIGIN FUNCTION

The Rastrigin function, formulated in eq. (1), is a non-convex function with a global minima at the origin. The n parameter is the number of dimensions and A is a constant that dictates how "bumpy" the loss surface of the function is. We set this value to 1 to allow the optimizers to better navigate this surface. The non-convexity of this function, coupled with its abundance of local minima, make it an informative candidate for visualising optimizer performance.

$$f(\mathbf{x}) = A_n + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right] \tag{1}$$

We test the MaSS optimizer on the 2D Rastrigin function which is symmetric about the X and Y axis, therefore reducing our testing domain. We limit the range to $-5 \leq f(x, y) \leq 5$. We initialize

the starting points of the optimizers to be $x, y = [-3, -4]$, and conduct tests with 2 different step sizes: η = 0.04 and η = 0.01. We analyse the paths taken by the different optimizers and note their respective convergence points.

From Figure 1, we observe that the MaSS optimizer is better able to navigate the loss surface of the Rastrigin function at larger step sizes, more so than the optimizers. Though the MaSS optimizer is able to converge at the global minima, it perpetually deviates around it. This could be the subject of further studies. At smaller step sizes, the MaSS optimizer exhibits the worst performance out of all the optimizers, getting stuck at a very early local minima. From further analysis, we notice that on the larger step size, MaSS exhibits the fastest convergence rate out of all the optimizers. The performance of the Nesterov optimizer is seen to decrease. This is a key result postulated in the original MaSS paper.

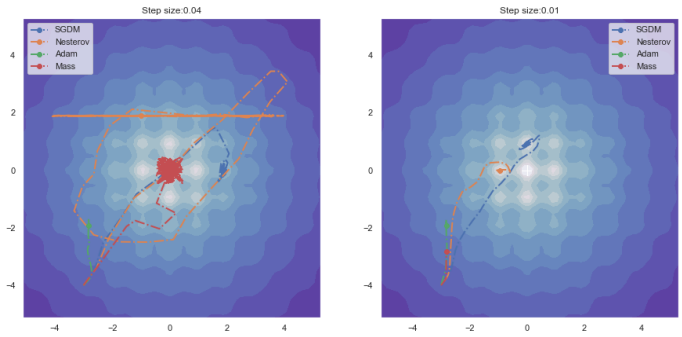

Figure 1: Rastrigin function contour plot with different optimizers starting at $x, y = [-3, -4]$

3 Neural Network Architectures

3.1 Testing Procedures

As mentioned above, we train different NN architectures in this paper: a fully connected NN (FCNN), a CNN, and a ResNet (ResNet-32). We pay close attention to the nuances in training procedures that other papers fail to include. As per the requirements of original paper, we train the ResNet and CNN on the CIFAR-10 dataset for 300 epochs and 150 epochs respectively, and the FCNN architecture on the MNIST data set for 100 epochs. The initialised learning rates used to train the models are shown in Table 1, all initialised with a batch size of 64. All recorded results shows the models respective performances of the test sets of the data sets they were trained on.

For ResNet & CNN, reduction of learning rate and data augmentation were implemented, techniques typically used to achieve state of the art results. We reduce the learning rate by a factor of 10 at epochs: 150 & 225 for the ResNet, and 60 & 120 for the CNN. We augment the data by enabling random horizontal flips, and random vertical and horizontal shifts with a range of 10% of the image width or height. Lastly, for the MaSS hyper-parameters κ & α, we adhere to the model-specific values listed in the paper.

3.2 Fully Connected Neural Network

A Fully Connected Neural Network (FCNN) was created for this paper, the structure of this was a model with three hidden layers and an output later, with ReLU and dropout with a keep probability of 50%, with an input of 784 features and output of 10 features. This initial model was expanded to include batch normalisation (using the PyTorch batch normalisation function with one dimension and 100 features), resulting in a faster and more stable learning process.

Each optimizer was tested both with and without batch normalisation, with results shown in Table 1. From this data, it can be seen that without batch normalisation the highest accuracy came from the MaSS optimizer at 87.78%. With batch normalization added the accuracy of each model increased drastically, each averaging roughly 96%. In this test the Nesterov model achieved 96.59%, slightly higher than MaSS with 96.45%.

The training accuracy and loss for each of these tests has been plotted in Figure 2. These graphs show only the best performing test for each optimizer, allowing direct comparison between them over the training life cycle. For the optimizers without normalisation, MaSS was by far the quickest to converge, as shown by the loss graph. With batch normalisation implemented, both accuracy and convergence rates were consistently improved, with very similar results for each optimizer, with SGD with Nesterov performing slightly above the other optimizers.

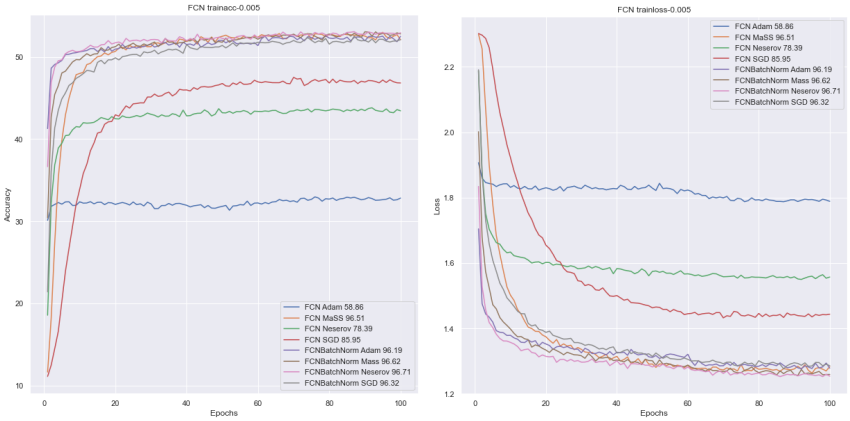

Figure 2: FCNN Training Accuracy, plotting Accuracy against epochs for four optimizers (Left), with and without batch normalisation. FCNN Training Loss, plotting Accuracy against epochs for four optimizers, with and without batch normalisation (Right)

3.3 ResNet

The ResNet architecture used in this project was a ResNet-32, containing 32 convolutional layers, built following the model description given in the original paper. This consisted of 15 residual blocks (3 different stacks of 5 blocks) followed by a 2x2 average pooling layer with a stride of 2, and a final softmax non-linearity output layer of size 10. The specifics of layer output sizes can be found in the original paper. This model has $\approx$467,000 trainable parameters in total and was used to classify the CIFAR-10 dataset.

The results for the classification accuracy of each optimizer are shown in Table 1, and training loss graphs for both learning rates shown in Figure 3. These plots represent the training loss for the first 150 epochs, only showing up to a loss value of 0.5. This is because the full plots contain 4 loss curves with very similar paths, all converging to virtually zero after the learning rate is dropped at 150 epochs, making it difficult to draw conclusions. It should be noted that the original paper does not include any plots for ResNet results, possibly for similar reasons.

Looking more acutely at the losses in Figure 3, it appears that though MaSS is fast to converge, it is not the fastest for either learning rate. This does not concur with the key theoretical conclusion drawn in the original paper, that MaSS has accelerated convergence rate over SGD. The classification results in Table 1 indicate that MaSS performs optimally for a smaller learning rate, but is still outperformed by other optimizers. Again, this is not in accordance with the original paper, where MaSS achieved the highest accuracy for both learning rates.

3.4 CNN

The CNN architecture analysed in this section mirrors the CNN architecture in the original paper. It consists of 3 ReLU activated convolutional layers, the first 2 of which contain 64 channels, and the last containing 128 channels. Each convolutional layer is accompanied by a 2x2 max pooling layer with stride of 2. This is followed by a ReLU activated fully connected layer of size 128. A dropout layer with keep probability of 0.5 succeeds this and an output layer of size 10 with softmax non-linearity is added after the dropout layer. We remove the last max pooling layer as this causes the input to the fully connected layer to be of incorrect size. They introduce a second CNN architecture in which they test the MaSS optimizer. They add a batch normalisation layer after each convolution

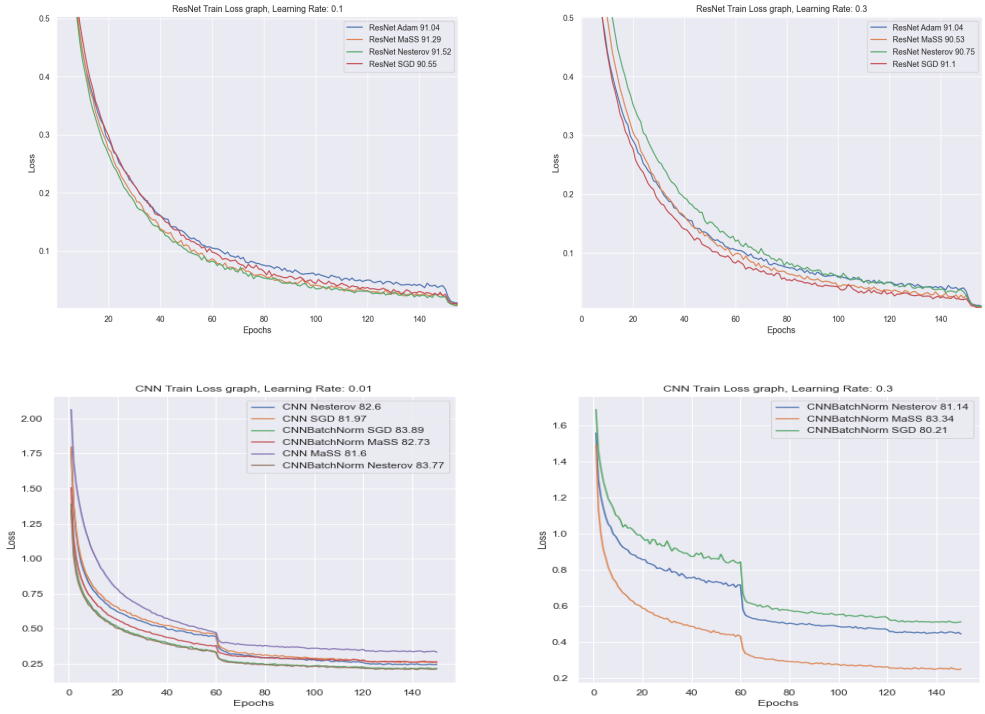

Figure 3: Training loss curves for all optimizers for the ResNet (top) and CNN (bottom), with learning rates: 0.1 (left) and 0.3 (right) for the ResNet, and 0.01 (left) and 0.3 (right) for the CNN.

and remove the dropout layer after the fully connected layer. We repeat the experiments conducted with both version of the neural network architectures.

Using the non batch-normalised variant of the CNN, all of the optimizers initialised on the larger step size $\eta = 0.3$ diverge, as evidenced in Table 1. This is in direct contrast to the batch normalised variant wherein all the optimizers initialised on both learning are able to converge. This contradicts their paper as they state that MaSS is able to converge in situations that Nesterov does not.

From fig. 3, at $\eta = 0.01$, the MaSS optimizer exhibits the slowest convergence speed on both model variants.The Nesterov optimizer exhibits the best performance on the non batch normalised variant, achieving an accuracy of 82.60%. At $\eta = 0.3$ the MaSS optimizer not only achieves the highest classification accuracy at 83.34%, but also the fastest convergence rate out of all the optimizers trained on the same learning rate, evidenced in the loss plot in Figure 3.

Model	η	SGD	SGD+Nesterov	MaSS	Adam
CNN	0.01	81.97%	82.60%	81.60%	80.19%‡
	0.3	Diverges	Diverges	Diverges	
CNN BatchNorm	0.01	83.89%	83.77%	82.73%	83.40%‡
	0.3	80.21%	81.14%	83.34%	
ResNet-32	0.1	90.55%	91.52%	91.29%	91.04%‡
	0.3	91.10%	90.75%	90.53%	
FCNN∗∗	0.005	82.26%	65.11%	87.78%	52.14%
FCNN BatchNorm∗∗	0.005	96.12%	96.59%	96.45%	96.04%

Table 1: Classification accuracy of all models.
‡Initialised with $\eta = 0.001$; ∗∗ Average of three runs

4 Conclusion

This paper indicates that MaSS achieves fast convergence rates on a range of architectures, but is still sometimes outperformed by current optimizers. This could be due to the inherent stochasticity in the initialisation of weights in neural networks. Lastly, analysis on the Rastrigin function and

CNN indicate that MaSS converges fastest with an increased step size. More clarity needs to be achieved on MaSS performance in neural networks.