COMP6248 Reproducibility Challenge
Online Learning Rate Adaptation With Hypergradient Descent

Hieu Tran 28311132 kht1g16@soton.ac.uk

Shabeer Rauf 28576888 smr3g16@soton.ac.uk

Abstract

This is a report for the Deep Learning Reproducibility Challenge using the paper “Online Learning Rate Adaptation With Hypergradient Descent” published in the ICLR 2018 conference. All the experiments in this paper were reproduced to assess the results produced in the paper. The three experiments include comparing the result produced by three algorithms: Stochastic Gradient Descent, Stochastic Gradient Descent with Nesterov momentum and Adam with their hypergradient descent versions over three applications: logistic regression, multilayer perceptron and fine-tuning.

1. Overview:

1.1. Hypergradient Descent:

Hypergradient descent method is defined by applying gradient descent on the learning rate of a gradient descent algorithm. The update is gradient-based and applied every iteration in an online manner.

For a gradient descent model with a scalar learning rate *α*, at time *t*, and objective function *f* and previous parameter θ_{t-1}, the update to the parameters can be written as:

$$\theta_t = \theta_{t-1} - \alpha * \nabla f(\theta_{t-1})$$

The paper then calculates the partial derivative of the update value $\nabla f(\theta_{t-1})$ with respect to the learning rate *α*. By applying the chain rule to the equation

$$\frac{df(\theta_{t-1})}{d\alpha} = \frac{df(\theta_{t-1})}{d\theta_{t-1}} \frac{d\theta_{t-1}}{d\alpha} = \nabla f(\theta_{t-1}) \frac{d(\theta_{t-2} - \alpha \nabla(\theta_{t-2})}{d\alpha} = \nabla f(\theta_{t-1})(-\nabla f(\theta_{t-2}))$$

This allows the update to the learning rate with a simple dot product and small memory cost. An additional parameter β is introduced as the hypergradient learning rate, which is used as the learning rate for the gradient descent of the algorithm’s learning rate:

$$\alpha_t = \alpha_{t-1} - \beta * \nabla f(\theta_{t-1})(-\nabla f(\theta_{t-2}))$$

2. Experiments:

The hypergradient descent algorithms were run along with their original versions on three different tasks in order to prove that the algorithms are able to achieve loss trajectories closer to the optimal one compared to the results achieved by a tuned initial learning rate. The three tasks chosen were a classification problem performed on the MNIST dataset with logistic regression, multilayer perceptron and VGG net.

2.1. Online tuning of learning rate:

The first experiment listed was to perform online tuning of the learning rate for Stochastic Gradient Descent and Adam algorithms with their HD alternates on the logistic regression and multilayer perceptron models. The learning rate of the algorithms are chosen from a set of values: {10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}} and for the HD optimisers, the hypergradient learning rate is set at a fixed 10^{-4}. The experiment was set up with the MNIST dataset, with all 60000 instances in the training set all went toward training the model and the 10000 test data points used for validation. For the MLP model, there are two hidden layers of 1000 units and use ReLU activation. The results obtained in this experiment shows that in all cases, the hypergradient descent models bring the loss curve nearer to optimal than the non-HD ones.

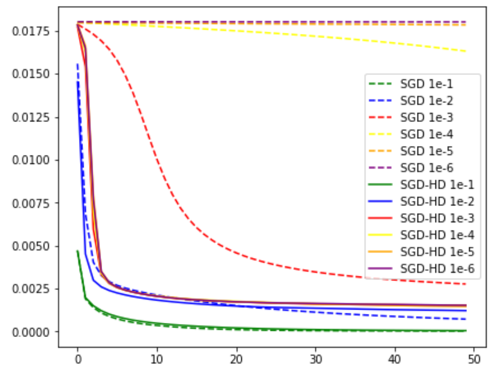
2.2. Tuning:

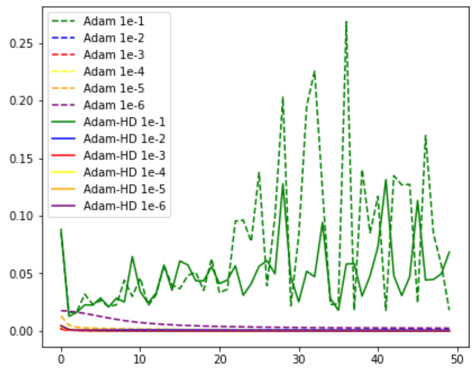
For the tasks of the experiment, the chosen learning rate $\alpha = 0.001$ for all optimisers. With SGD with Nesterov and the HD version, $\mu = 0.9$. The hypergradient learning rate chosen for Adam-HD is 10^{-7} and for the two SGD algorithms are 10^{-3}.
This report reproduced all three of the experiments listed in the paper. The experiments were run using Pytorch and the Python library hypergrad provided by the authors of the paper.

3. Reproduced results:

3.1. Online tuning:

- Multilayer perceptron:





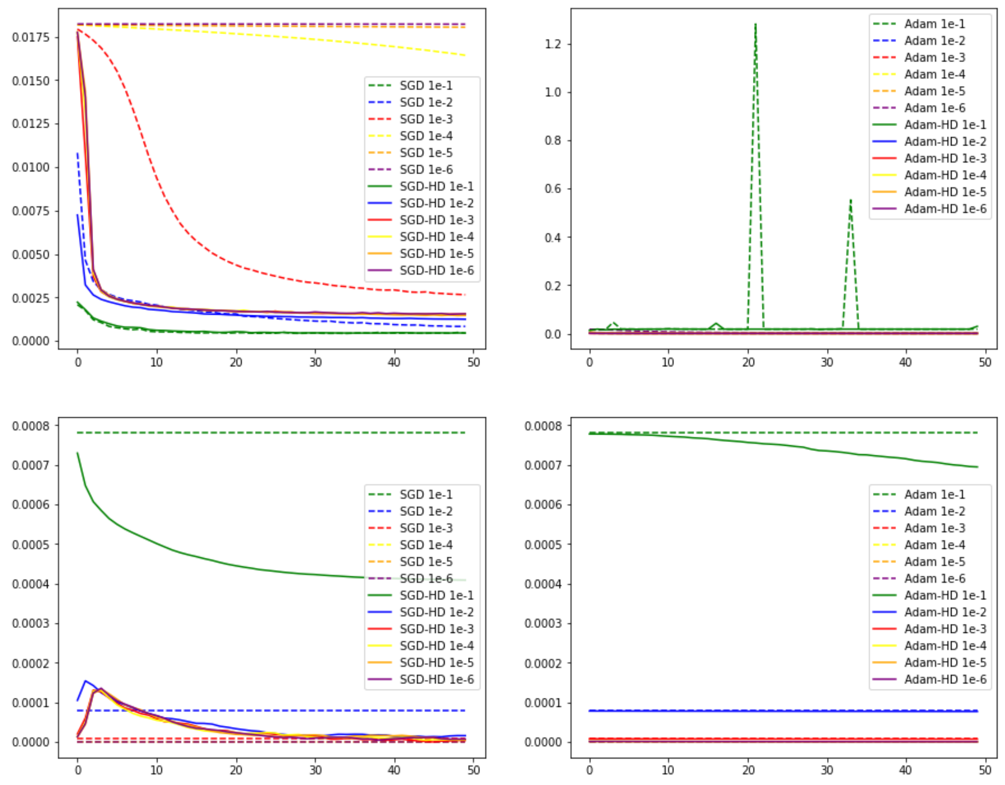
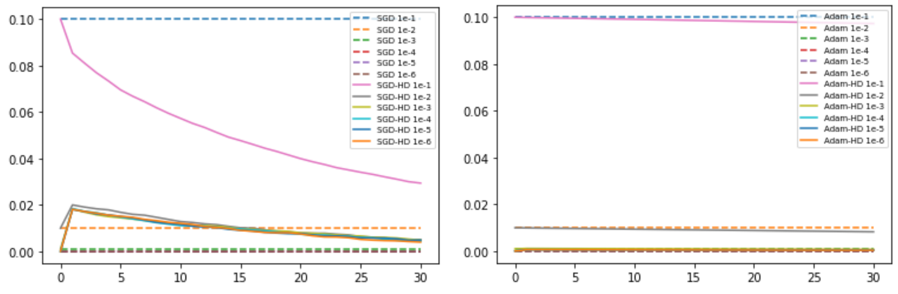


Figure 1. Online tuning result for MLP. Top: training loss. Middle: validation loss. Bottom: learning rate

The images above are the plot of the training loss, validation loss and learning rate in each epoch when training an MLP with 2 1000-unit hidden layers using SGD and Adam, together with SGD and Adam with hypergradient descent. The learning rate is picked from the set of values: {10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}} and the hypergradient learning rate is set to be 1e-4 for SGD-HD. For Adam with hypergradient descent, when the hypergradient learning rate set in the paper of 1e-4 was given to the optimiser, all models experienced exploding gradients as the learning rates of the models all reached NaN. Therefore, for the reproduced experiment, the hyperlearning rate for Adam-HD is chosen as the hyperlearning rate set in the following experiment in the paper: 1e-7.
Except from the difference in the hypergradient learning rate parameter for Adam-HD, the reproduced experiment finds the same conclusion of the online tuning experiment in the paper: For all value of α, the MLP model running with the SGD-HD algorithm has it loss curve closer to the optimal value than the model running with normal SGD.

- Logistic regression:



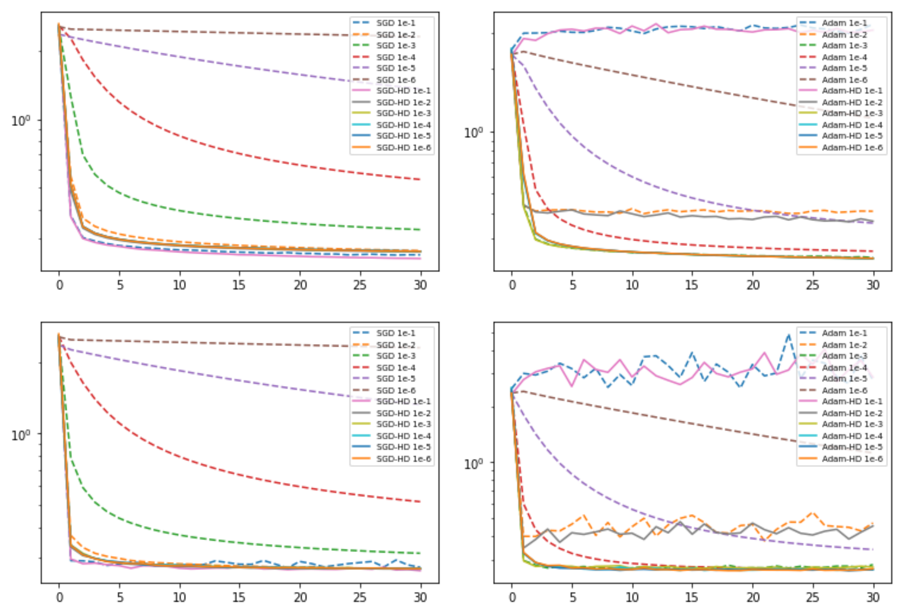


Figure 2: Online tuning result for logistic regression. Top: training loss. Middle: validation loss. Bottom: learning rate

Figure 2 shows a comparison of online tuning for logistic regression using stochastic gradient descent and Adam, along with their hypergradient descent methods. The figure shows that in almost all cases the hypergradient methods vastly outperform the regular methods. The loss curves of hypergradient methods, irrespective of their initial learning rates approach the loss curve of the most optimised regular method. As shown in figure 4, as long as the hypergradient learning rate is not too high, the hypergradient methods always outperform the regular methods.

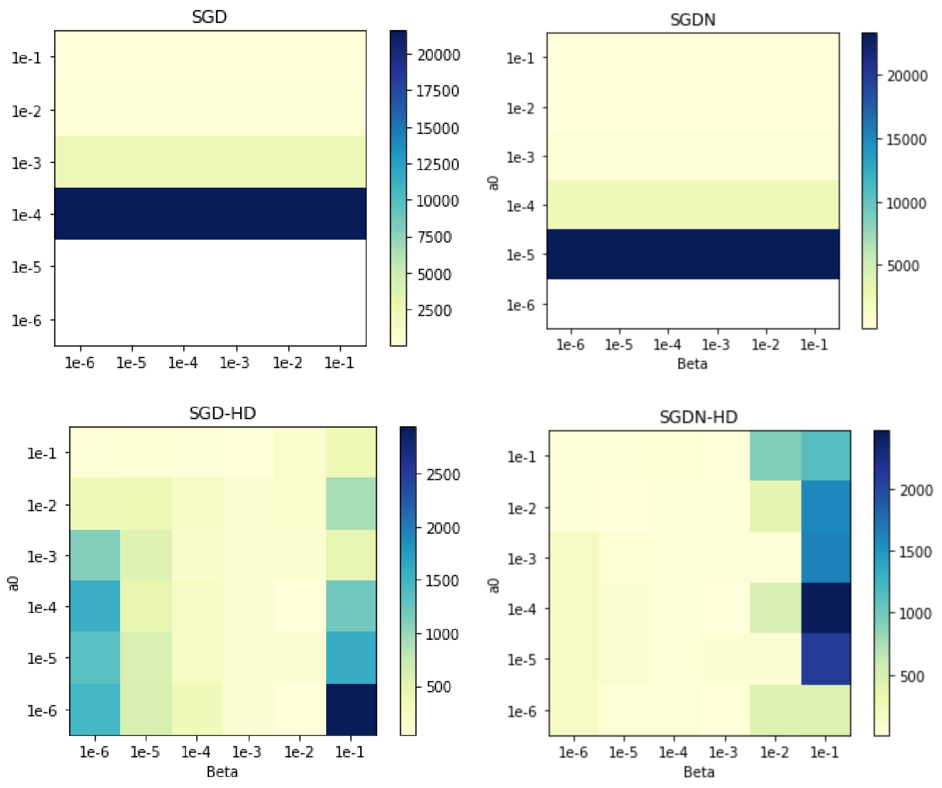
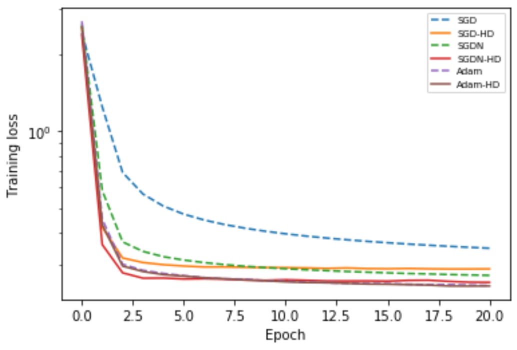
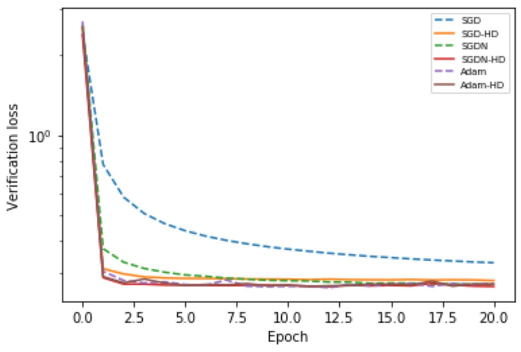


Figure 4: Grid search of hyperparameters for logistic regression

3.2. Tuning:

- Logistic regression:





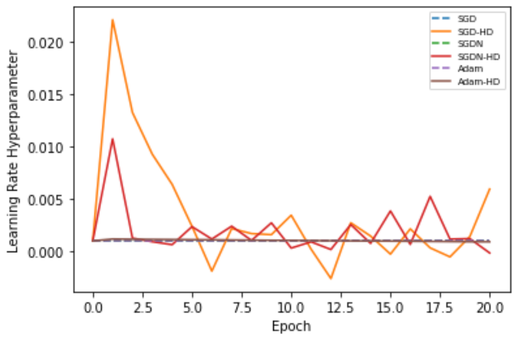
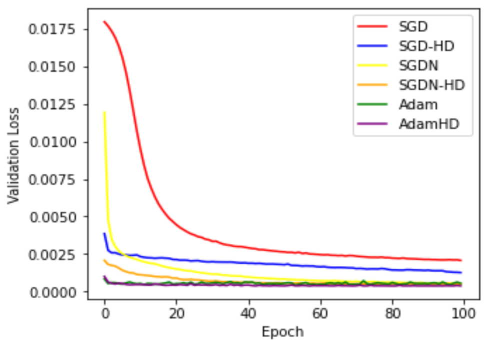
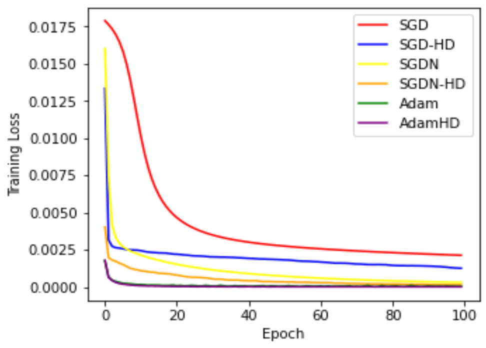


Figure 3: Logistic regression tuning with $\alpha = 0.001$

To tune the optimisers, a logistic regression classifier was used to classify the MNIST database. The training loss, validation loss and evolution of learning rates was plotted for Adam, stochastic gradient descent, nesterov stochastic gradient descent and their hypergradient descent counterparts. All of the optimisers used a learning rate of 0.001 and L2 regularisation of 10^{-4}. For the Nesterov SGD variants, we take $\mu = 0.9$. For Adam, we use $\beta 1 = 0.9$, $\beta 2 = 0.999$, $\varepsilon = 10^{-8}$ and apply a $1/\sqrt{t}$ decay to the learning rate. It can be seen that for each of the optimizers, the hypergradient descent method outperformed the regular versions.

- Multilayer perceptron:





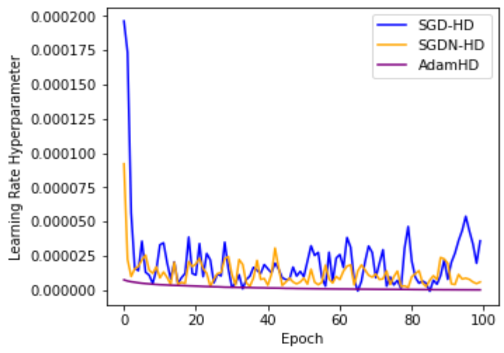


Figure 3. Multilayer Perceptron tuning with $\alpha = 0.001$

The images above are the plot of the validation loss, training loss and learning rate hyperparameter for the six optimisers. For all optimisers, the learning rate is set at 0.001. With the hypergradient descent SGD algorithms(SGD and SGD with Nesterov), the hypergradient learning rate is set at 1e^{-4}. While for the Adam-HD, the parameter is set to 1e^{-7}. Similar to the online tuning experiment, the MLP is set up with 2 hidden layers, each with 1000 hidden units and uses ReLU activation. The training dataset is the 60000 images in the training data of MNIST while the 10000 test images are used for validation.
The result achieved from the experiment once again confirms the finding of the paper, the hypergradient descent algorithms achieved results close to optimality quicker than the non-hypergradient descent ones.

- VGG net:

The VGG net experiment used a machine with Intel Core i7-6850K CPU, 64 GB RAM, and NVIDIA Titan Xp GPU and the training time with 200 epochs took over two hours. Due to these requirements on computing resources, the experiment on tuning the VGG net cannot be reproduced in this report.

4. Conclusion

Hypergradient descent is a method which calculates the partial derivative of the update with respect to the learning rate to tune the learning rate for better convergence speed. The reproduced experiments have confirmed the finding in the study, which is the algorithm with hypergradient descent results in an error trajectory much closer to optimality than the non-hypergradient-descent algorithm given the same initial learning rate.

5. References

1. Baydin A.G., Rubio D.M., Wood F., Cornish R. and Schmidt M.,2018.”Online learning rate adaptation with hypergradient descent” ICLR 2018
2. T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. Proceedings of the 30th International Conference on Machine Learning, 28:343–351, 2013.
3. J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research,