HYPER-REGULARIZATION: AN ADAPTIVE CHOICE FOR THE LEARNING RATE IN GRADIENT DESCENT, ICLR REPRODUCIBILITY CHALLENGE

Saeed Saadatnejad, Mohammadhossein Bahari, Bassel Belhadj

1 INTRODUCTION

Gradient descent is for sure, the most popular optimization algorithm among the proposed methods so far. The method simply considers the gradient and moves in the inverse direction relative to a parameter, called learning rate. Learning rate is an important parameter which has a significant effect on the performance of the algorithm since a small learning rate leads to slow convergence and a big one may cause divergence from the local optimum. As a result, choosing the suitable value for it is crucial and challenging due to the vital impact of it.

There is a lot of research on this topic addressing the issue. However, still, there is a lack of an automatic method with convergence guarantees. This paper proposes an automatic method for learning the learning rate which is called Hyper-Regularization. descriptions of the method! [hyperparameter and cast its adaptive choice with the parameter training into a joint process. We formulate this process as a maximum framework by imposing a regularizer on the hyperparameter.]

Our aim in this report is to reproduce the results shown in the paper and to analyze the method as well. In the next section, we will briefly explain the algorithm. Then, the experiments will be described and the results are compared with the reported ones in the paper. After that, the shortcomings of the method based on the experiments are discussed and finally, we will evaluate the paper and its achievements.

The link of this reputability challenge in GitHub is as follows: https://github.com/reproducibility-challenge/iclr$_2$019/$issues$/104

2 METHOD

In this section then we will briefly explain the proposed method. For the exact details and proofs, refer to the paper.

Consider an online learning problem where the data is from domain $X \subseteq R^d$ and each time we have a (sub)gradient g_t and the observed point is x_t and we want to predict x_{t+1}. The generalized form of the optimization will be like the following:

$$x_{t+1} = \sqcap_X(x_t - \zeta g_t) = argmin_{x \in X} ||x - (x_t - \zeta g_t)||_2^2 \tag{1}$$

Now we can convert it to a saddle point problem by adding a hyper-regularizer which is the difference between the new learning rate and an auxiliary vector η. Thus, it will be like the following equation:

$$max_\beta min_{x \in X} \phi_t(x, \beta) =< g_t, x - x_t > +1/2(||x - x_t||^2 - D(\beta, \eta_t)) \tag{2}$$

We have a different choice of ϕ function and η vector. Also, by solving the min-max optimization in an exact form or in an approximate form, we will have different algorithms for calculation of β. Four functions of KL-divergence, Reverse KL-divergence, Hellinger distance and X^2 distance is used for ϕ which made four different categories of KL, RKL, Hellinger and X^2 respectively. Also, we have three different methods for solving the equations so there are three different sub-categories in each of the mentioned algorithms (the algorithms 1,2,3 in the paper), e.g. we have KL1, KL2, and KL3.

Through the experiments, the performance of the algorithms are compared with each other and the parameters will be tuned.

3 EXPERIMENTS

There are two major experiments in the paper. The first one is a two-layer neural network with 500 hidden states which classifies MNIST database. The second one is a VGG-net with the CIFAR as the input. We have implemented both of the methods and derived the different figures for each of them. In the next subsections, we will briefly explain the structures and the results related to the paper. Moreover, there were issues which will be elaborated and the solutions we found out for them. In order to better assess the algorithm, we evaluated the method on a linear regression as well. We will use all of the results in the last section to draw conclusions.

All the experiments are run on a machine with Intel Core i7 (12 cores) CPUs and 2 NVIDIA GeForce GTX 1080 Ti GPUs.

It is worth mentioning that there are some details in the implementation which are noted in the paper. So our implemented code is according to these points which are:

- Bounded learning rate: The author used growth clipping in all methods to maintain stable performance. So the value of β_{t+1} in our experiments falls in $[\beta_t, 2\beta_t]$.
- Starting point: Although the paper claims that the algorithm is invariant to the initial value of the learning rate, our experiments show that it is way sensitive to that. As a result, since for initial learning rates 1e0 and 1e1, we have numerically big numbers that lead to nans, we fix learning rate to 1e-2 for first 50 loops, then the formulas are utilized. This is due to the distance between the weights and gradients from the global minimum point that cause big numbers without a small learning rate. So we let the values to get close to the local minimum and then the specific method for calculation of learning rate is used. We will discuss it later.
- Regularization: Based on the paper, l2 regularization with a coefficient of 10^{-4} is applied to both experiments.

3.1 LINEAR REGRESSION

Although the paper includes only the next two experiments, we have added a baseline to better compare the results. To do so, a linear regression is trained using the proposed method.

3.2 FULLY CONNECTED NEURAL NETWORK

The first experiment reported in the paper is a two-layer neural network which classifies MNIST dataset. It is composed of a hidden layer with 500 nodes with sigmoid function and an output layer which has 10 nodes with softmax activation function. Although we could write the code with Pytorch, we used python since we wanted to have control over the variables and monitor the impacts of the new method. Thus, the forward and the backward path are implemented. Based on the paper, each of the weights needs a separate learning rate which has to be calculated based on the gradient of that parameter. So, we have provided a function, *giveNewBetta*, which calculates the learning rate based on the gradient.

MNIST is a database of handwritten digits with 10 classes, composed of 60,000 examples as a training set, and a test set of 10,000 examples. it is an online dataset and is widely used for classification problems.

3.3 VGG NET CIFAR CLASSIFICATION

CIFAR-10 is a dataset which includes 60,000 images from 10 different classes (including car, airplane, ship, truck, dog, cat, frog, horse, deer, bird) and the challenge is to classify these images. There are a lot of different architectures which are used in order to better classify and achieve high accuracy. Among them, VGG net is one the most popular ones in image classification. In the paper, the second experiment is done using VGG on CIFA-10. Unfortunately, the exact architecture was not mentioned by authors so we used VGG-16.

VGG 16 has several convolution layers [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, M'] which M denotes the max pooling. The structure is depicted in

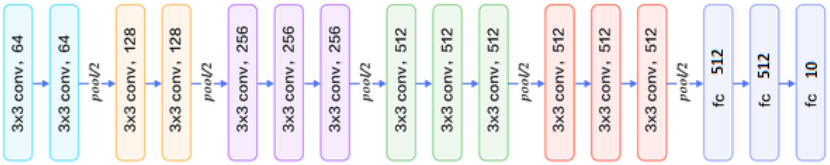


Figure 1: VGG net Structure

figure 1 As proposed in the paper, we implemented this by adding batch normalization after each convolution layer. This is a common method which avoids the shift in feature scale. The output of these convolution layers is passed through three fully connected networks and finally, the class label of the image is achieved.

The PyTorch library was used for implementing the codes.

3.3.1 Optimization implementation

As there are multiple layers in a VGG-16, we have a lot of weights and biases and its hard to compute the gradients of all parameters by hand (like what we did in the previous experiment). So we calculate these gradients by a function, *loss.backward*, provided by Pytorch library. After that, we cannot use regular optimizer of Pytorch since it is not allowed to use different learning rates for each element of a W in each step but we need that for implementing the method. Thus, we rewrite an optimization function using gradients calculated by a backprop function. In that way, we are able to calculate β with respect to papers formulas and then apply gradient descent.

the normalization part of the code is the regular normalization formula which has been extracted from ImageNet competition.

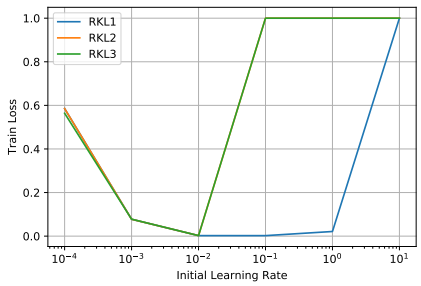
4 Results

The paper reports two main results based on the experiments. The first one is the choice of the initial learning rate where the training loss is reported for different initial values. The second one is training loss over epochs which shows the pace of convergence. The two next sections will explain the reproduced results and our assessments.

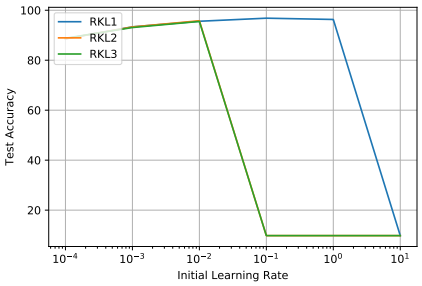
4.1 Issue with the start point

The author claims that the algorithm is robust to the initial value of β as it is mentioned in the paper's abstract. Figure 2(a), 2(b) which are related to figure 1 in the paper, show the training loss for different values of initial learning rate (ILR) after 40 epochs over the neural network. As clear, generally, the results are similar to the paper which shows divergence over large values, high error over small values and best results in the middle. However, we have divergence in RKL2, RKL3 in lower values compared to the paper. As opposed to the author's claim, it is obvious that the algorithm is highly dependant on ILR and specifically, for large ILR values the algorithm diverges. This fact can be seen even in the paper's figure and in our figure as well.

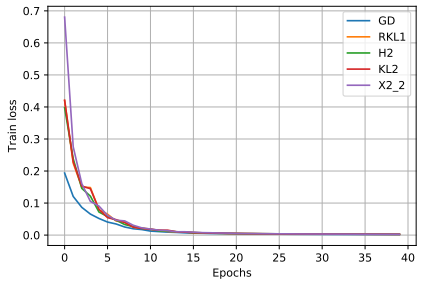
The fact is that when we are far from the global minimum, the value of the gradient is high and having a small value for the learning rate is crucial. Because of that, the algorithm diverges for high values of ILR since the method doesn't provide a small value for learning rate which is the same for all the formulations. We tried different things to solve the issue. One of them was fixing the learning rate for first rounds and whenever the gradient approaches the global minimum, use the proposed method. Although it worked for some cases, still we have problems with some of the others. Also, we tried to bound values, but it didn't work. Our mathematical investigations didn't lead to an answer to the problem of divergence. We think that probably the error with the proofs reported in the paper are in the conditions which are assumed and are not valid in large gradients.



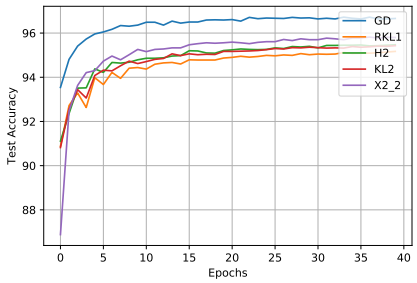
(a) Train loss with different Initial learning rates



(b) Test accuracy with different epochs



(c) Train loss with different Initial learning rates



(d) Train loss with different Initial learning rates

Figure 2: Training loss and test accuracy for different settings

4.2 CONVERGENCE TIME

The second important point about the algorithm is the convergence time. Figure 2(c), 2(d) which are related to figure 1 and figure 2 in the paper show no improvements over the traditional methods. It's worth mentioning that the algorithm didn't diverge in the full-batch mode with the paper suggested initial learning rates, so our figures are based on batch mode. As we can see, the loss is lower and the accuracy is higher in the gradient descent algorithm.

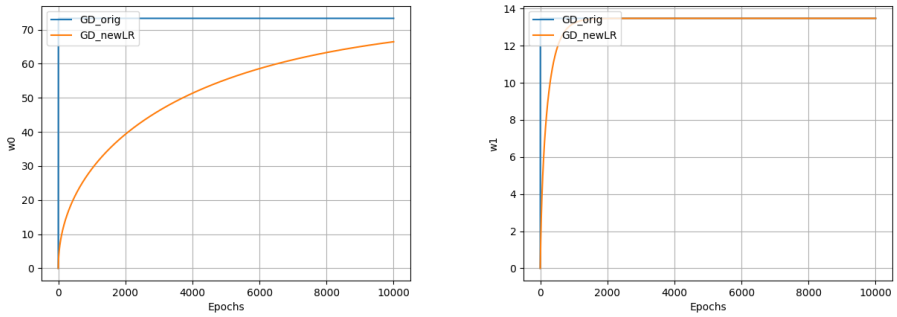
For the linear model, We observed no improvement compared to original GD. In GD when the gradient decreases (we are close to minimum) the weights decrease a bit but here since the learning rate is dependent to the gradient, it will decrease a lot which causes to receive results after 20,000 iterations compare to GD which needs 20 iterations. This fact is shown in figure 3 by plotting the parameters of the algorithm after some epochs.

Also, each iteration will take more time to be calculated. Here is the time for calculation of each iteration for the linear algorithm:

- Gradient Descent new LR: execution time per iteration=338.80 micro seconds
- Gradient Descent: execution time per iteration=64.90 micro seconds
- SGD new LR: execution time per iteration=512.00 micro seconds
- SGD: execution time per iteration=382.80 micro seconds

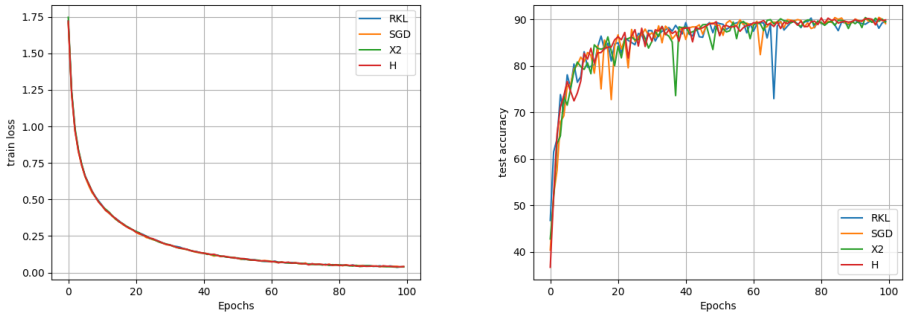
Moreover, the VGG net results are plotted in figure 4. This is the test accuracy and train loss over epochs which are related to figure 3 of the paper. There is not any meaningful difference between different types of algorithms.

All together, We can conclude that there are no improvements in the timing.



(a) w0 values over iterations (b) w1 values over iteration

Figure 3: Weight (parameter) values for linear regression using the original gradient descent and the proposed one



(a) training loss over iterations (b) test accuracy over iteration

Figure 4: Training loss and test accuracy for VGG net classification using diffrent algorithms including SGD, RKL, X2, H, KL

5 CONCLUSION

Based on the experiments and analysis in the report, we come to this conclusion that the proposed method does not have special improvements over baselines which is clear even in the paper figures. Also, although we add some parts to the algorithm to decrease divergence occurrence, still it is a main issue for the algorithm. In terms of reproducibility, we had several issues with the method which can be because of the lack of clearance in the details of the experiments.