Time-varying Learning Rates for Deep Learning.

Ajeghrir, Mustapha
mustapha.ajeghrir@epfl.ch

Baldwin, Nicolas
nicolas.baldwin@epfl.ch

Sula, Gerald
gerald.sula@epfl.ch

***Abstract*—When it comes to building machine learning models, a lot of importance comes down to what and how you pick the parameters of the model. Perhaps the most important choice comes from the learning rate parameter, which can have a huge impact on performance. Nowadays, we have the tools to very precisely pick a learning rate that evolves during the training/testing steps. In this project we will explore the behaviour of four different types of time-varying learning rates, and their relation to some of the most used optimizers (SGD, Adam and Adadelta). We use the MNIST digit classification dataset to test their performance on a simple deep learning architecture. We conclude which types of learning rates are more favorable depending on the optimizer.**

I. Introduction

Learning rate is the parameter of a model that specifies how quickly or how slowly the model is being adapted to a problem.[1] Picking a rate that is too small, and the model will not be able to find an optimal configuration in time. Pick a large rate, and the model might miss and go beyond an optimum. Additionally, the learning rate does not necessarily need to be a fixed value, but it can be a function that varies during the training.[2]
Furthermore, Neural Network models take advantage of adaptive optimization algorithms, that adjust their parameters based on the data. The choice of both the learning rate and optimization algorithm, is extremely crucial to the success of a project. In this report, we will study how the choice between four different families of time-varying learning rate functions, in combination with three optimization algorithms, impact the performance of a deep leaning network.
In particular we will be testing this, on a three-layered fully connected neural network, one of the most simple and widely used types of architectures of deep learning. The families of time-varying learning rates we will be studying include: Step Learning Rate, Reduce-On-Plateau, Cosine Annealing and a custom lambda function. The functions will be tested when used in tandem with 3 optimization algorithms: SGD, Adam [3] and Adadelta [4].

II. Time-varying functions

In order to implement different time-varying learning rates, we have used the *Scheduler* class available in the pyTorch framework. [5]

A. Step Learning Rate

The step learning rate scheduler will multiply the learning rate by a factor γ every $step - size$ epochs. The figure [1, a] shows the evolution of the learning rate using this scheduler and the shown parameters.

B. Cosine Annealing

The Cosine Annealing scheduler uses the following equations to set the learning rate for each epoch.

$$\begin{cases} if \quad T_{cur} \neq (2k+1)T_{max} : \\ \eta_t = \eta_{min} + \dfrac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\dfrac{T_{cur}}{T_{max}}\pi\right)\right) \\ if \quad T_{cur} = (2k+1)T_{max} : \\ \eta_{t+1} = \eta_t + \dfrac{1}{2}(\eta_{max} - \eta_{min})\left(1 - \cos\left(\dfrac{1}{T_{max}}\pi\right)\right) \end{cases}$$

where :

$$\begin{cases} T_{cur} & : \text{ number of epochs since the last restart} \\ \eta_{max} & : \text{ set to initial learning rate} \\ T_{max} & : \text{ a parameter, half of the oscillation period} \end{cases}$$

The figure [1, b] shows how the learning rate evolves using this scheduler

C. ReduceLR On Plateau

This scheduler is more advanced than others. It continuously verifies a quantity that witnesses the improvements of the model, this quantity could be accuracy or the loss function. After noticing that this quantity doesn't improve for more than $patience$ amount of epochs, it then multiplies the learning rate by $factor$. The figure [1, c] shows how the learning rate is evolving in a concrete multi-epoch train of a simple example model.

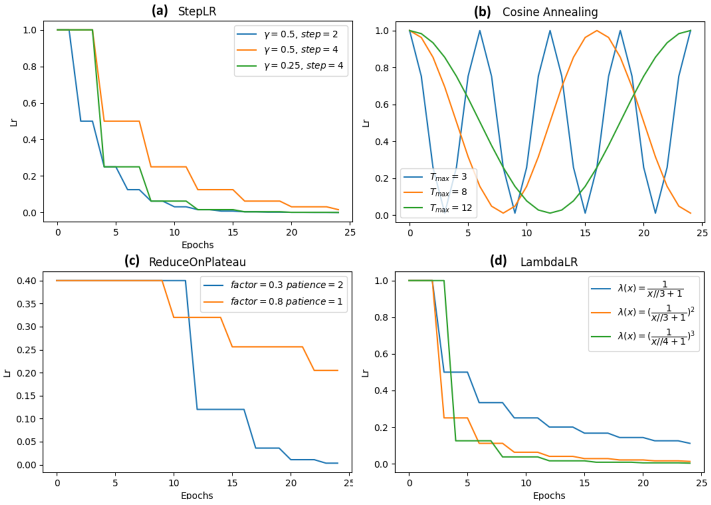

Fig. 1. Evolution of the learning rate using the four schedulers

One thing that we should know is that the evolution of the learning rate for this scheduler depends also on the optimizer. If the model continues to improve at least once every $patience$ amount of epochs, the learning rate will never be changed.

D. Lambda

The lambda scheduler takes a function ($epoch \mapsto factor$) as a parameter, this function will dictate the value of the learning rate for each epoch. We used some functions of type $x \mapsto \left(\frac{1}{\lfloor x/a \rfloor + 1}\right)^b$, where a and b are some parameters to tune.

The figure [1, d] shows some examples for those functions. The role of a is to not decrease the learning rate for every epoch, while b dictates how fast a decrease will be.

III. Model Architecture

The model we are using in order to run the experiments is a simple fully connected neural network. It has a total of 3 layers, starting with the input layer that takes as input an image of size $28 * 28$ and outputs a tensor of size 256; the second layer $256- > 100$; and the final layer $100- > 10$. After the final layer, a ReLU activation function is applied to the output.

IV. Dataset

The well known MNIST dataset [6] has been used as input for the experiments. It consists 60000 samples in the training set and 10000 samples in the test set, each one a 28*28px black and white image representing a single hand written number. We standardize the dataset using the mean and standard deviation of the images to potentially improve convergence of the results.

V. Experimental setup

The selection of hyperparameters can have a significant impact on a neural network performance. In order to make the best choice from a predefined set of parameters we implemented a grid search algorithm which iterates on all possible combinations of parameters, trains the neural network architecture and evaluates its performance using a K-fold cross-validation step. The K-fold-CV splits the train set into K splits and evaluates the network performance K times, each time using a different split for the validation and K-1 splits for training. This allows us to robustly pick the best hyperparameters by only using the original train set as input. Once the best hyperparameters have been selected, we can use them to fully train a model using the entire train set of 6000 samples as input and test it on the test set.

We repeat this process using as input each one of the learning-rate functions and optimizers separately, in order to fairly compare their best results. For our setup we have decided to run the models for 25 epochs, and perform 5-fold cross validation.

VI. Results

After cross validating our hyper parameters, we are able to evaluate the performances of our schedulers. The results will be separated in 3 parts. We will present the performances of each scheduler on our three optimizers: SGD, Adadelta and Adam. In each case, we have also included the performance when using a hyper-parameter tuned constant learning rate. Note That all the plots and performance metrics are estimated through 10 separate rounds of training.

A. Results using SGD as Optimizer

The basic performance metrics of our model using SGD as optimizer and varying the schedulers are presented on Table[I]. The test loss plot is shown on figure [2]. As we can see, for SGD the best performing scheduler is the Lambda LR scheduler. However, the reduceLR On Plateau, Step Learning Rate schedulers and Cosine Annealing still perform better than using a constant learning rate which leads us to think that using them still makes sense even if they don't have the best performances.

We can notice the performance of the most oscillating time-varying function, that is the Cosine Annealing one, varies a lot through the epochs. While The rest of the schedulers become somewhat stable after some epochs. Note that, for the Lambda LR scheduler using SGD as an optimizer, when cross validating on different formulas to reduce the learning rate throughout epochs, we found that the formula: $lr_{epoch} = (\frac{1}{\lfloor \frac{epoch}{5} \rfloor + 1})^3 \cdot lr_{epoch=0}$ was the best.

It's also important to note that our models might all be overfitting. As we can see, in figure [2], at around epoch 3-5, the test loss starts increasing. It would be wise to consider early-stopping in a future experiment.

TABLE I

Statistics of model using SGD as optimizer and varying the schedulers. All statistics were estimated through 10 separate rounds of training.

Scheduler with SGD	**Average Train Accuracy**	**Average Test Accuracy**	**Average Train Loss**	**Average Test Loss**	**std Train Loss**	**std Test Loss**
Constant	0.9997	0.9834	0.00113	0.1048	5.99e-6	2.19e-4
Lambda	0.9998	0.9838	0.00250	**0.0665**	3.99e-8	1.18e-5
R_Plateau	**1.0**	0.9838	0.00031	0.0865	1.84e-8	9.16e-5
Cosine	**1.0**	**0.9840**	0.00018	0.0864	5.1e-10	2.26e-5
Step	**1.0**	0.9839	**9.66e-5**	0.0883	3e-11	1.35e-5

B. Results using Adadelta as Optimizer

The basic performance metrics of our model using Adadelta as optimizer and varying the schedulers are presented on Table[II]. The test loss plot is presented on figure [3]. As we can see, for Adadelta, just like SGD, the best performing scheduler is the Lambda LR scheduler. However, the reduceLR On Plateau, Step Learning Rate schedulers and Cosine Annealing schedulers once again all perform better than using a constant learning rate.

With Adadelta the performance of most schedulers is similar between them, and we can notice less variance to the

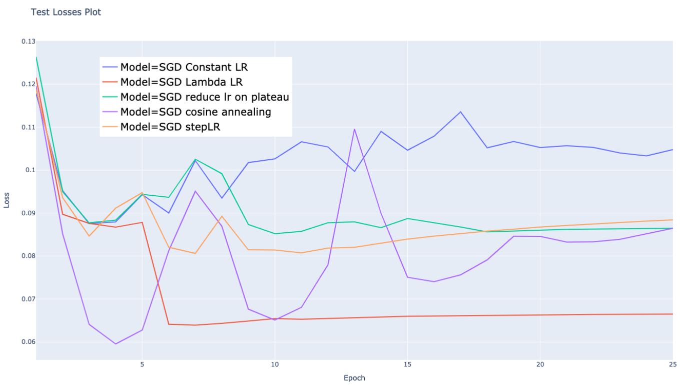

Fig. 2. Test loss of various schedulers with SGD as optimizer.

performance through the epochs due to the nature of the optimizer. Note that this time, for the Lambda LR scheduler using Adadelta as an optimizer we found that the formula: $lr_{epoch} = (\frac{1}{\lfloor \frac{epoch}{4} \rfloor + 1})^3 \cdot lr_{epoch=0}$ was the best.

Its again important to note that our models might all be overfitting. As we can see, in figure [3], at around epoch 5-10, the test loss starts increasing.

TABLE II

Statistics of model using Adadelta as optimizer and varying the schedulers. All statistics were estimated through 10 seperate rounds of training.

Scheduler with Adadelta	**Average Train Accuracy**	**Average Test Accuracy**	**Average Train Loss**	**Average Test Loss**	**std Train Loss**	**std Test Loss**
Constant	**1.0**	0.9826	**8.86e-5**	0.0963	1e-10	4.08e-5
Lambda	0.9979	0.9822	0.0110	**0.0604**	5.35e-8	5.18e-6
R_Plateau	**1.0**	**0.9830**	1e-4	0.0963	5e-10	2.59e-5
Cosine	**1.0**	0.9821	3.6e-4	0.0872	1.25e-9	3.82e-6
Step	**1.0**	0.9821	1.58e-4	0.0935	2e-10	1.78e-5

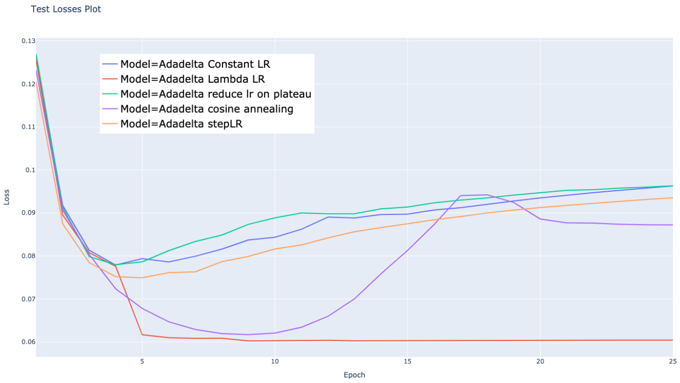

Fig. 3. Test loss of various schedulers with Adadelta as an optimizer.

C. Results using Adam as Optimizer

The basic performance metrics of our model using ADAM as optimizer and varying the schedulers are presented on Table[III]. The test loss plot is presented on figure [4]. As we can see for Adam the results are a little different from when using SGD or Adadelta as optimizers. The best performing scheduler is the StepLR scheduler, with the other schedulers also performing better than a constant learning rate.

By observing the plots, we notice that this time the trajectories resemble to a degree those of the SGD optimizer. We can once again notice the oscillating behavior of the Cosine Annealing function, and in general the performances are varied through the epochs. Note that, for the Lambda LR scheduler using ADAM as an optimizer, when cross validating on different formulas, we found that the formula: $lr_{epoch} = e^{-\frac{epoch}{4}} \cdot lr_{epoch=0}$ was the best.

TABLE III

Statistics of model using Adam as optimizer and varying the schedulers. All statistics were estimated through 10 separate rounds of training.

Scheduler with Adam	**Average Train Accuracy**	**Average Test Accuracy**	**Average Train Loss**	**Average Test Loss**	**std Train Loss**	**std Test Loss**
Constant	0.9568	0.9443	0.1838	0.3311	3.76e-4	1.53e-3
Lambda	**0.9921**	**0.9662**	**0.0277**	0.1857	6.59e-5	1.44e-4
R_Plateau	0.9823	0.9607	0.0600	0.2210	1.98e-4	1.63e-4
Cosine	0.9713	0.9563	0.1098	0.2398	1.66e-4	2.14e-4
Step	0.9858	0.9656	0.0479	**0.1535**	1.42e-4	1.07e-4

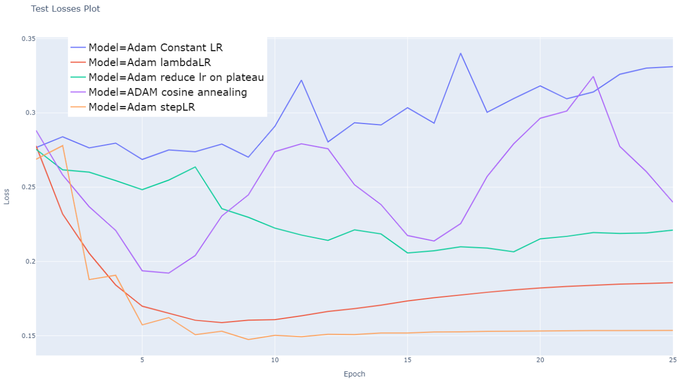

Fig. 4. Test loss of various schedulers with Adam as optimizer.

VII. Summary

To summarize, we observed the effect of various schedulers (Lambda LR, Reduce Lr On Plateau, Cosine Annealing and Step LR) on a fully connected network while using 3 different optimizers (SGD, ADAM and Adadelta). Our results show that more often than not using evolving learning rates improves the model's overall performance. In fact, independently of what optimizer was used, training the model with time-varying schedulers give smaller test losses than training the model with a constant learning rate. Additionally we noticed that varying the optimizer does affect the overall performance of the various time-varying schedulers. For example, the Lambda LR scheduler (which is the most hand-crafted scheduler out of the four tested) performs the best when using SGD or Adadelta as optimizers whereas the Step LR gives the best performances when using ADAM. Therefore, we can conclude that using time-varying learning rate functions can increase the performance of our model, and it's something every engineer should consider when working on deep learning models.

References

[1] A. Gotmare, N. S. Keskar, C. Xiong, and R. Socher, “A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation,” 2018.

[2] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang, “Demystifying learning rate policies for high accuracy training of deep neural networks,” 2019.

[3] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

[4] M. D. Zeiler, “Adadelta: An adaptive learning rate method,” 2012.

[5] “Torch torch.optim documentation.” [Online]. Available: https://pytorch.org/docs/stable/optim.html

[6] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

Appendix

A. Full description of all hyperparameters

For each of the combinations of learning rate scheduler and optimizer we presented, we tried different combinations of input parameters through a 5-fold Cross-Validation step. We present here all the parameters we tested for each one, as well as the best parameters. The best combinations of parameters for each of the optimizers can be found in the supplementary files.

Step LR	
Parameter	Values
Optimizer	SGD, Adadelta, Adam
Step size	2, 4
Step gamma	0.3, 0.5, 0.7, 0.8, 0.85, 0.95
Learning Rate	0.4, 0.2, 0.05
Batch Size	30, 60, 90
Criterion	Cross Entropy Loss

Cosine annealing	
Parameter	Values
Optimizer	SGD, Adadelta, Adam
Tmax	3, 5, 8, 12
eta min	0.01, 0.03
Learning Rate	0.4, 0.2, 0.05
Batch Size	30, 60, 90
Criterion	Cross Entropy Loss

Reduce on plateau	
Parameter	Values
Optimizer	SGD, Adadelta, Adam
Patience	2, 4
Gamma	0.1, 0.3, 0.5
Learning Rate	0.4, 0.2, 0.05
Batch Size	30, 60, 90
Criterion	Cross Entropy Loss

Lambda Function	
Parameter	Values
Optimizer	SGD, Adadelta, Adam
Variable a	3, 4, 5
Variable b	1, 2, 3, 4, 5
Learning Rate	0.4, 0.2, 0.05
Batch Size	30, 60, 90
Criterion	Cross Entropy Loss