Learning the Learning Rate for Gradient Descent by Gradient Descent

O. Majumder, M. Donini, P. Chaudhari {orchid,donini,prtic}@amazon.com
Amazon Web Services (AWS)

Abstract

This paper introduces an algorithm inspired from the work of Franceschi et al. (2017) for automatically tuning the learning rate while training neural networks. We formalize this problem as minimizing a given performance metric (e.g. validation error) at a future epoch using its "hyper-gradient" with respect to the learning rate at the current iteration. Such a hyper-gradient is difficult to estimate and we discuss how approximations and Hessian-vector products allow us to develop a **R**eal-**T**ime method for **H**yper-**P**arameter **O**ptimization (RT-HPO). We present a comparison between RT-HPO and other popular HPO techniques and show that our approach performs better in terms of the final accuracy of the trained model. Online adaptation of the learning introduces two extra hyper-parameters, the initial value of the learning rate and the hyper-learning rate; our empirical results demonstrate that the accuracy obtained by RT-HPO is largely insensitive to these hyper-parameters.

Keywords: AutoML, Hyper-Parameter Optimization, Gradient-based methods

1. Introduction

This paper considers the problem of Hyper-Parameter Optimization (HPO). It exploits the observation that for some hyper-parameters like learning rate, momentum and weight averaging, one can take the *gradient* of a performance metric. In contrast, other hyper-parameters like convolutional and pooling kernel sizes, dropout probability, batch-size etc. cannot be adapted easily during training. We focus on the former and discuss gradient-based HPO techniques using the learning rate as the prototypical example. Our method involves taking the derivative of a performance metric, say the validation error, at a future epoch with respect to the learning rate; we call this the "hyper-gradient". A secondary optimization routine updates the learning rate using this hyper-gradient.

We discuss a general formulation that encapsulates known heuristics for adapting the learning rate in HPO. In general, the hyper-gradient is difficult to estimate: (i) the performance metric can be sensitive to small changes and (ii) computing the hyper-gradient involves back-propagation through time which is sensitive to vanishing or exploding gradients. We construct approximations to enable us to perform gradient-based HPO in practice. Our experiments on medium-scale image classification problems (e.g. CIFAR-100) demonstrate that gradient-based HPO is a promising technique and can outperform a fixed exponentially decaying learning rate strategy or other adaptive methods like AdaDelta.

2. Related Work

Grid-based hyper-parameter search quickly becomes prohibitive as the number of hyper-parameters grows (Bergstra and Bengio, 2012) and even random search may perform better. Model-based approaches like Bayesian Optimization (BO) (Hutter et al., 2011; Snoek et al., 2012; Perrone et al., 2017) have recently shown improved performance compared to random and grid search by fitting a function approximator to estimate the performance. These techniques are typically sensitive to the choice of the function class and do not work well for discrete hyper-parameters. A complementary approach to HPO is papers like Hyperband (Li et al., 2018) which uses multi-arm bandit theory to discard less-promising configurations and allocate more resources to the promising ones (Jamieson and Talwalkar, 2016).

Adaptively selecting the learning rate has been well-studied in the context of optimization algorithms (Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2015). These algorithms are extremely popular in practice but still require the user to pick the learning rate and an annealing schedule for it. Let us note that there are effective and popular heuristics for learning rate tuning, e.g., Smith (2017); Loshchilov and Hutter (2017). Our approach tunes the learning rate schedule using gradients and can be used in conjunction with these algorithms.

Our method is closely related to gradient-based HPO methods of (Baydin et al., 2018; Franceschi et al., 2017; Domke, 2012; Maclaurin et al., 2015; Pedregosa, 2016; Franceschi et al., 2018). These methods update fixed hyper-parameters such as regularization co-efficients using the hyper-gradient of the performance metric. This paper uses similar ideas, discussed further in Section 3, to adapt hyper-parameters that change with time. For the case of learning rate, we can make approximations that allow us to efficiently compute the hyper-gradient.

3. Approach

We will work in the standard supervised learning setting. Let the dataset be X which consists of samples and their ground-truth labels. We will denote the training data by X_{train} and the validation data by X_{val}. Let $w \in \mathbb{R}^N$ denote the weights of our model. Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951) is a first-order gradient-based optimization procedure, it can be written as $w_{t+1} = w_t - \eta_t \, \nabla \ell(w_t;\ \mathcal{b}_t)$.

Here w_t are the parameters at t^{th} iteration, η_t is the learning rate, $\ell(w_t;\ \mathcal{b}_t)$ is the loss on the mini-batch $\mathcal{b}_t \subset X_{\text{train}}$ for the parameters w_t. The mini-batch $\mathcal{b}_t$ is chosen by sampling a few instances from the training data X_{train} without replacement. We will denote the training loss over the entire training dataset as $\ell(w_t;\ X_{\text{train}})$. The sequence of updates converges to a local minimizer of the training loss. If the loss function ℓ is convex in the parameters w_t we obtain convergence to a global minimum. We will occasionally write $\eta(t)$ instead of η_t to think of the learning rate schedule as a function of the training iterations.

For the purposes of HPO, let us define an energy $E(w_t)$ which we would like to minimize. This is often taken to the validation loss $E(w_t) = \ell(w_t;\ X_{\text{val}})$, but it can also be some other performance metric, e.g., the Bayesian Information Criterion (BIC), or the training error $\ell(w_t; X_{\text{train}})$ itself. Our formulation only requires that $E(w_t)$ be differentiable with respect to the parameters w_t. The optimization problem that we would like to solve for tuning the

learning rate η_t can then be formally written as follows:

$$\begin{aligned} \underset{\eta_1,\ldots,\eta_{T-1}}{\text{minimize}} \quad & E(w_T) \\ \text{such that} \quad & w_{t+1} = w_t - \eta_t \, \nabla \ell(w_t;\, \mathcal{b}_t) \quad \forall\, t = 0, \ldots, T-1 \\ & w_0 = \overline{w},\ \eta_0 = \overline{\eta} \text{ (initial weights and initial learning rate).} \end{aligned} \tag{1}$$

In other words, we would like to minimize the energy $E(w_T)$ at some terminal time T while performing stochastic gradient descent updates on the parameters w_t to minimize the training loss $\ell(w_t; \mathcal{b}_t)$. The variables in the above optimization problem are the learning rate at iteration $\eta_1, \ldots, \eta_{T-1}$. The weights w_t are only controlled indirectly through the SGD update in the constraint in (1). An oracle for solving the above problem would find the learning rate schedule $\eta(\cdot)$ that achieves the smallest value of the performance metric $E(w_T)$ in the budget of T iterations. Roughly speaking, the oracle adapts the step-size in SGD such that updates to w_t that lead to a good performance at time T are bolstered.

3.1 Real-Time Hyper-Parameter Optimization (RT-HPO)

This section introduces the Real-Time Hyper-Parameter Optimization (RT-HPO) algorithm that approximates the solution of Problem (1). It will be helpful to rewrite the weight update as $w_{t+1} = w_t - \eta_t \nabla \ell(w_t;\, \mathcal{b}_t) =: \Phi(w_t, \eta_t)$.

We have made the dependency on the mini-batch $\mathcal{b}_t$ implicit. The mapping $\Phi(w_t, \eta_t)$ is defined above for SGD but in principle it can be any update procedure, e.g. SGD with momentum (Qian, 1999), that updates the weights. In that case, we also need to maintain a state variable, the momentum vector, and take the gradient with respect to it. This is a natural extension to the computation in this section.

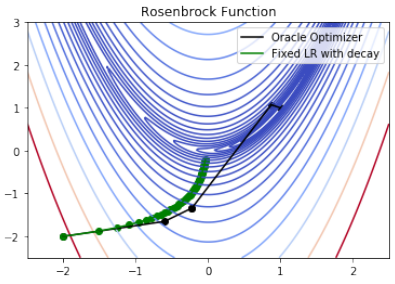

Figure 1: A toy-example of the solution to (1) using the two-dimensional Rosenbrock function (Rosenbrock, 1960) (left). For the same initial condition, gradient descent with decaying learning rate of $1/t$ converges to a sub-optimal value. The solution of (1) reaches the global optimum $[1, 1]$ in 4 steps.

Approximation 1 (Sequence of optimization problems). We approximate (1) as a sequence of optimization problems whereby at each iteration t, we update the current learning rate η_t using information from the past and the gradient of the future trajectory. We have practical motivations for such an approximation. First, instead of solving for the optimal learning rate schedule which requires solving (1) directly, we can exploit the causal structure of the modified problem above to change η_t in real-time. Second, automatic differentiation techniques allow us to compute the gradient $\frac{\partial E(w_T)}{\partial \eta_t}$ efficiently.

We now exploit the above observation to set up a gradient descent scheme for updating η_t as

$$\eta_{t+1} = \eta_t + \beta \Delta \eta_t. \tag{2}$$

The parameter β is called the "hyper-learning rate" and is a hyper-parameter of our approach; this will be discussed in detail shortly. The change in learning rate $-\Delta\eta_t$ is the hyper-gradient and (2) therefore corresponds to gradient descent on the hyper-parameter η using

the hyper-gradient. The following development mirrors that of Franceschi et al. (2017) and will exploit the chain-rule to compute the hyper-gradient. Consider

$$\frac{\mathrm{d}E(w_T)}{\mathrm{d}\eta_t} = \nabla E(w_T)\ \frac{\mathrm{d}w_T}{\mathrm{d}\eta_t}; \tag{3}$$

the second term refers to the total derivative of the weights at time T with respect to the learning rate at time t. We can now use the chain rule to get

$$\frac{\mathrm{d}w_{t+1}}{\mathrm{d}\eta_t} = \underbrace{\frac{\partial \Phi(w_t, \eta_t)}{\partial w_t}}_{:=A_t}\ \underbrace{\frac{\mathrm{d}w_t}{\mathrm{d}\eta_t}}_{:=z_t} + \underbrace{\frac{\partial \Phi(w_t, \eta_t)}{\partial \eta_t}}_{:=B_t};$$

which is a linear recursion $z_{t+1} = A_t z_t + B_t$ for all $t \in \{1, \dots, T-1\}$. The gradient of the objective with respect to the current learning rate is now written as

$$\frac{\mathrm{d}E(w_T)}{\mathrm{d}\eta_t} = \nabla E(w_T)\ z_T = \nabla E(w_T)\left(A_{T-1} z_{T-1} + B_{T-1}\right) = \nabla E(w_T) \sum_{t=1}^{T-1} \left(\prod_{s=t+1}^{T-1} A_s \right) B_t. \tag{4}$$

Let us make a few observations. First, the gradient can be computed recursively. Second, the hyper-gradient at time t depends on both the past trajectory and the future trajectory. This makes the hyper-gradient very difficult to compute since we would need to know the optimal learning rate in the future η_s to compute the terms z_s for $s > t$. We make the following approximation to alleviate this.

Approximation 2 (**Hyper-gradient does not depend on the future**)**.** We will replace the exact gradient of the sequential HPO problem in (4) by

$$\frac{\mathrm{d}E(w_T)}{\mathrm{d}\eta_t} := \nabla E(w_t) \sum_{s=1}^{t-1} \left(\prod_{u=s+1}^{t-1} A_u \right)\ B_s. \tag{5}$$

We replaced the future gradient $\nabla E(w_T)$ by the gradient of the validation loss on the current weights $\nabla E(w_t)$. Further, we cull the summation up to the current time t. This is a bold approximation because it entails that the hyper-gradient does not give us any information of the future trajectory. Note however that the hyper-gradient is not a statistic of the entire past trajectory up to time t. Our experiments show that in spite of this, the hyper-gradient is an effective way to adapt the learning rate.

Computational complexity. In the update rule for z_t, the first term $A_t z_t$ contains a Hessian-vector product $A_t\ z_t = (1 - \eta_t \nabla^2 \ell(w_t; \cdot))\ z_t = z_t - \eta_t \nabla^2 \ell(w_t; \cdot)\ z_t$. The Hessian is a matrix of $N \times N$ elements (where N is the number of the weights) and computing this term therefore requires the second order derivative of all the operators in the architecture. This complex multiplication has to be performed at every step which could be prohibitive for large models if not implemented efficiently. We implement the Hessian-vector product using the method described in Pearlmutter (1994).

Hyper-learning rate β. The hyper-learning rate is a hyper-parameter of RT-HPO. What is the advantage of selecting the learning rate η_t automatically if we have another hyper-parameter β to pick in (2)? As our experiments in Section 4 and the ones in Franceschi

et al. (2017, 2018) show, the benefit of this formulation is that the hyper-learning-rate is more stable and obtaining state-of-the-art performance does not require modifications to β.

First-order approximation for the training loss and the performance metric. The authors in Baydin et al. (2018) propose a way to adapt the learning rate during training. They are motivated by the observation that if the gradient of the past few iterations are correlated with each other, then we can potentially increase the learning rate and hope to make faster progress. Formally, in our notation, their hyper-gradient can be written as

$$\Delta\eta_t = \Big\langle \nabla\ell(w_{t-1}, \mathcal{b}_{t-1}), \nabla\ell(w_t, \mathcal{b}_t) \Big\rangle. \tag{6}$$

Note that the above expression is inexpensive to compute and we only need to store a copy of the past gradient $\nabla\ell(w_{t-1}, \mathcal{b}_{t-1})$. It can also be easily seen that this is the one-step, first order approximation of the gradient in (5). Another rationale for the above update rule is that it approximately minimizes the expected training error. We will use this method as a baseline named as "First-order (Train)".

We can also construct an alternative hyper-gradient update rule inspired from First-order (Train) and use the performance metric in place of the training loss by substituting $\nabla\ell(w_t, \mathcal{b}_t)$ in Eq. 6 with $\nabla E(w_t, \mathcal{v}_t)$. The motivation for this update rule is that updating the learning rate which results in minimization of the validation loss as well as the training loss. We call this method "First-order (Val)".

4. Experiments

This section provides experimental results to show that: (i) we can efficiently compute the hyper-gradient and use it for adapting the learning rate and (ii) when everything else remains constant, RT-HPO leads to improved performances as compared to the other methods. Our implementation of RT-HPO, as yet, scales to medium-size convolutional neural networks. Our results are therefore presented on MNIST (LeCun, 1998), CIFAR-10 and the CIFAR-100 datasets (Krizhevsky et al., 2014), where we show competitive accuracies.

Setup: We do not use dropout or regularization while training to reduce total number of tunable parameters which can impact the final accuracy. We use vanilla SGD without momentum as the inner optimizer for all experiments. We also consider a set of widely used baselines: fixed learning rate along with an exponentially decaying learning rate scheduler, BO technique leveraging SageMaker Automatic Model Tuning and AdaDelta with its default parameters (as suggested in Zeiler (2012)). We perform experiments using (i) Multi-Layer Perceptron (MLP) on MNIST: a three-layer perceptron with 512, 256 hidden nodes, 784 input neurons and 10 output neurons, (ii) LeNet (LeCun et al., 2001) on MNIST, (iii) ResNet-18 (He et al., 2016) on CIFAR-10, and (iv) ResNet-18 on CIFAR-100.

A summary of all the results can be found in Table 1 showing the classification performances with respect to quality and robustness of the classification model. RT-HPO outperforms the First-order (Train) and First-order (Val) algorithm on CIFAR-10 and CIFAR-100; all three algorithms perform almost equally on MNIST. Further, RT-HPO is most robust to the choice of the $\bar{\eta}$ while First-order (Val) is most robust to the choice of β. We found RT-HPO to be stable in terms of both $\bar{\eta}$ and β as long as the value of β is chosen to be small. The accuracy obtained by RT-HPO is better than that of both First-order (Train) and First-order (Val). In terms of running time, due to the Hessian-Vector products (HVPs),

	Fixed $\bar{\eta}$	Fixed β	Top	Time
MLP on MNIST	(%)	(%)	(%)	(min)
First-order (Train)	98.37 ± 0.05	98.19 ± 0.17	98.43	14
First-order (Val)	98.27 ± 0.24	98.23 ± 0.16	98.43	32
RT-HPO	98.42 ± 0.08	98.30 ± 0.01	98.49	14
Grid Search	-	97.49 ± 1.60	<u>98.67</u>	-
AdaDelta	-	-	98.62	14
BO	-	-	98.55	140
LeNet on MNIST				
First-order (Train)	99.06 ± 0.29	98.69 ± 0.89	99.32	12
First-order (Val)	99.22 ± 0.17	99.08 ± 0.17	<u>99.38</u>	30
RT-HPO	99.32 ± 0.04	99.24 ± 0.13	99.36	12
Grid Search	-	98.97 ± 0.52	99.24	-
AdaDelta	-	-	<u>99.38</u>	12
BO	-	-	99.23	120
ResNet-18 on CIFAR-10				
First-order (Train)	87.21 ± 3.99	86.15 ± 1.91	92.33	150
First-order (Val)	90.86 ± 1.11	88.96 ± 4.83	92.37	360
RT-HPO	89.23 ± 5.88	92.79 ± 0.36	<u>93.04</u>	1920
Grid Search	-	89.92 ± 2.56	92.33	-
AdaDelta	-	-	92.46	150
BO	-	-	92.86	1500
ResNet-18 on CIFAR-100				
First-order (Train)	61.79 ± 6.97	62.71 ± 0.32	70.47	150
First-order (Val)	67.84 ± 1.66	64.16 ± 5.43	69.60	360
RT-HPO	66.82 ± 4.73	71.55 ± 0.03	<u>71.57</u>	1920
Grid Search	-	66.29 ± 5.48	71.06	-
AdaDelta	-	-	70.31	150
BO	-	-	70.5	1500

Table 1: **Summary of the results**: (i) average (and std) of the test accuracy when the best initial learning rate $\bar{\eta}$ is selected and changing the hyper-learning rate β (i.e. robustness with respect to β), (ii) the same with the best β and changing $\bar{\eta}$ (i.e. robustness with respect to $\bar{\eta}$), (iii) the best result among all the combinations of $\bar{\eta}$ and β, (iv) the cost of a single training run in minutes. For BO, it refers to the total time of all training runs as part of the BO process.

RT-HPO is significantly (12.8 times First-order (Train) and 5.3 times First-order (Val)) slower for deeper networks. We believe this gap can be made closer by sub-sampling the HVP. Let us note that Bayesian HPO, when run for similar time as that of RT-HPO obtains marginally worse accuracies.

5. Discussion

We demonstrated that gradient-based HPO techniques like RT-HPO can be made to work and can even outperform current approaches. Extending gradient-based techniques to adapt other hyper-parameters like momentum, weight decay as well as extend RT-HPO for other optimizers like SGD With momentum or Adam is a promising direction. We believe these techniques are a first-step towards completely automating the training process of deep neural networks.

References

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BkrsAzWAb`.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Justin Domke. Generic methods for optimization-based modeling. In *AISTATS*, volume 22, pages 318–326, 2012.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173, 2017.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1563–1572, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, page 4, 2014.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1): 147–160, 1994.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*, volume 48, pages 737–746, 2016.

Valerio Perrone, Rodolphe Jenatton, Matthias Seeger, and Cedric Archambeau. Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start. *arXiv preprint arXiv:1712.02902*, 2017.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, mar 1960. doi: 10.1093/comjnl/3.3.175. URL `https://doi.org/10.1093/comjnl/3.3.175`.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.