Online Learning Rate Adaptation with Hypergradient Descent

Mohbat, Momin Abbas, Huzaifa Arif, Tobias Park, Xinyan Sun

Rensselaer Polytechnic Institute

December 06, 2021

Table of Contents

- CODE: https://github.com/mohbattharani/Hypergradient-Descent

The Problem

- Conventionally, the learning rate for an optimizer is tuned manually
  - Grid search, Bayesian optimization - expensive and tedious
- RMSProp, Adagrad dynamically update learning rates
  - Problem: these algorithms are initialized with a fixed global learning rate that still needs tuning

Solution

- How do we find optimal learning rate with lesser manual tuning ?
  - This can be achieved by adaptively optimizing the learning rate using first order information
  - Hypergradient descent (Baydin et al., 2018) uses this idea to achieve better convergence for various gradient based optimizers

Hypergradient Descent (HD)

- Paper: "Online Learning Rate Adaptation with Hypergradient Descent"
  - Published Feb 26, 2018
  - Conference paper at ICLR 2018 item Authors: Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, Frank Wood
- In this presentation, we demonstrate the effectiveness of hypergradient descent on various optimizers, including Stochastic Gradient Descent (SGD), SGD with Nesterov, and Adam
- Through substantial empirical evaluation on various models and datasets, we show the generalization and limitations of hyper-gradient descent method

Literature

- Finding good hyperparameters is itself an optimization problem which could be solved using another level of gradient descent (Bengio)
- Per parameters based adaptive learning rate used by:
  - AdaGrad
  - RMSProp
  - vSGD
  - Issues: These methods require an initial learning rate which also need tuning

Hyper-gradient Descent (HD)

- We derive HD using the regular gradient descent update rule:

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

where α is the learning rate

Hyper-gradient Descent (HD)

- We derive HD using the regular gradient descent update rule:

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

where α is the learning rate

- We compute the partial derivative:

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

$$\frac{\partial f(\theta_{t-1})}{\partial \alpha} = \nabla f(\theta_{t-1}) . \frac{\partial(\theta_{t-2} - \alpha \nabla f(\theta_{t-2}))}{\partial \alpha} = \nabla f(\theta_{t-1}) . (-\nabla f(\theta_{t-2})) \tag{2}$$

Hyper-gradient Descent (HD)

- Hence, the update for α becomes:

$$\alpha_t = \alpha_{t-1} - \beta(\frac{\partial f(\theta_{t-1})}{\partial \alpha})$$

$$\alpha_t = \alpha_{t-1} + \beta \nabla f(\theta_{t-1}).(\nabla f(\theta_{t-2})) \tag{3}$$

where β is the hypergradient learning rate

- Finally, the parameter update then becomes:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla f(\theta_{t-1}) \tag{4}$$

Cost and Performance

- Memory: only need to store one extra variable (product of the gradients)
- Computation: only need to perform one additional dot product
- HD does not add significant additional performance costs compared to unoptimized methods

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

Algorithm: SGD vs SGD-HD

Algorithm 1 Stochastic gradient descent (SGD)

Require: α, $f(\theta)$, $\theta_{\mathbf{0}}$
t $\leftarrow 0$
while θ_t not converged **do**
t $\leftarrow t+1$
$g_t \leftarrow \nabla f_t(\theta_{t-\mathbf{1}})$
$u_t \leftarrow -\alpha g_t$
$\theta_t = \theta_{t-\mathbf{1}} + u_t$
end while
return θ_t

Algorithm 2 SGD with hyp. desc. (SGD-HD)

Require: α, $f(\theta)$, $\theta_{\mathbf{0}}$, β
t, $\nabla_\alpha u_{\mathbf{0}} \leftarrow 0, 0$
while θ_t not converged **do**
t $\leftarrow t+1$,
$g_t \leftarrow \nabla f_t(\theta_{t-\mathbf{1}})$
$h_t \leftarrow \mathbf{g}_t \nabla_\alpha u_{t-\mathbf{1}}$
$\alpha_t \leftarrow \alpha_{t-\mathbf{1}}$ - βh_t
$u_t \leftarrow -\alpha g_t$
$\nabla_\alpha u_t \leftarrow -\mathbf{g}_t$
$\theta_t = \theta_{t-\mathbf{1}} + u_t$
end while
return θ_t

Algorithm: SGDN vs SGDN-HD

Algorithm 3 SGD with Nesterov (SGDN)

Require: μ : *momentum*
t, $v_{\mathbf{0}} \leftarrow 0, 0$
Update rule:
$v_t \leftarrow \mu v_{t-\mathbf{1}} + g_t$
$u_t \leftarrow -\alpha(g_t + \mu v_t)$

Algorithm 4 SGDN with hyp. desc.(SGDN-HD)

Require: μ : *momentum*
t, $v_{\mathbf{0}}$, $\nabla_{\alpha} u_{\mathbf{0}} \leftarrow 0, 0, 0$
Update rule:
$v_t \leftarrow \mu v_{t-\mathbf{1}} + g_t$
$u_t \leftarrow -\alpha_t(g_t + \mu v_t)$
$\nabla_{\alpha} u_t \leftarrow -(\mathbf{g}_t + \mu v_t)$

Algorithm: Adam vs Adam-HD

Algorithm 5 Adam

Require: β_1, $\beta_2 \in [0, 1)$: decay rates for Adam
t, m_0, $v_0 \leftarrow 0, 0, 0$
Update rule:
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t \leftarrow \mathrm{m}_t / (1 - \beta_1^t)$
$\hat{v}_t \leftarrow \mathrm{v}_t / (1 - \beta_2^t)$
$u_t \leftarrow -\alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

Algorithm 6 Adam with hyp. desc. (Adam-HD)

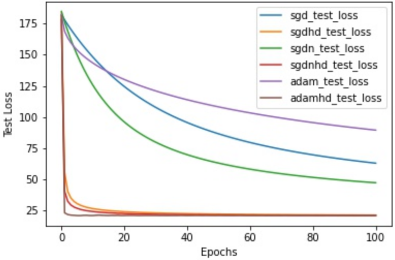
Require: β_1, $\beta_2 \in [0, 1)$: decay rates for Adam
t, m_0, v_0, $\nabla_\alpha u_0 \leftarrow 0, 0, 0, 0$
Update rule:
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t \leftarrow \mathrm{m}_t / (1 - \beta_1^t)$
$\hat{v}_t \leftarrow \mathrm{v}_t / (1 - \beta_2^t)$
$u_t \leftarrow -\alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
$\nabla_\alpha u_t \leftarrow -\mathrm{m}_t \ / \ (\sqrt{\hat{v}_t} + \epsilon)$

Results

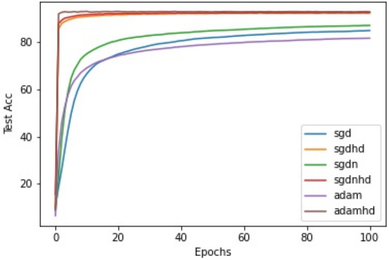
Model	Dataset		
	MNIST	CIFAR10	CIFAR100
Logistic Regression	✓	-	-
Multi-layer Preceptron	✓	✓	-
WideResNet	-	✓	-
ResNet18	-	✓	✓
ResNet101	-	✓	✓

Table 1: Dataset and models selected for evaluation

Results: Logistic Regression on MNIST

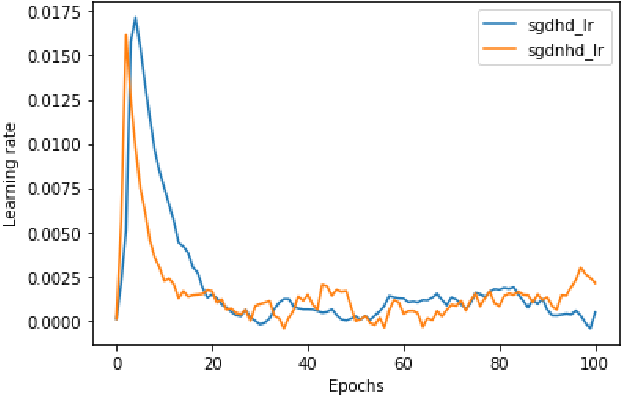


(a) Test Loss.

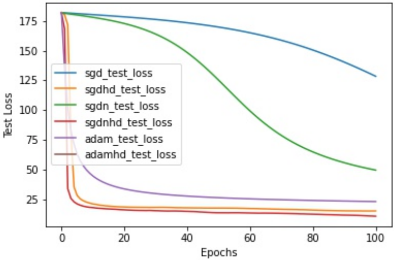


(b) Test accuracy.

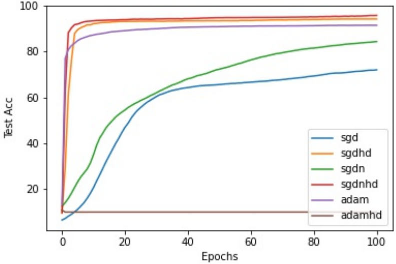
Results: Multi layer Preceptron on MNIST (Learning rate)



Results: Multi layer Preceptron on MNIST

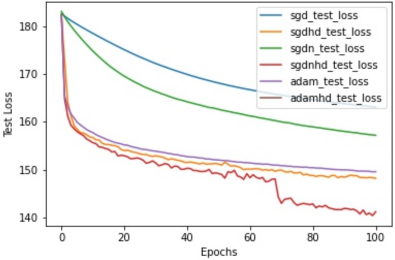


(a) Test Loss.

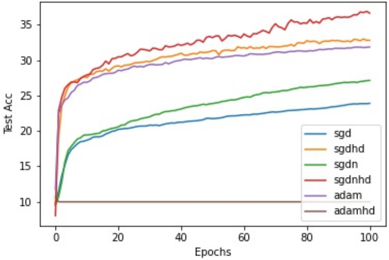


(b) Test accuracy.

Results: Multi layer Preceptron on CIFAR10

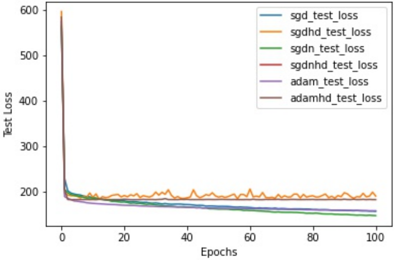


(a) Test Loss.

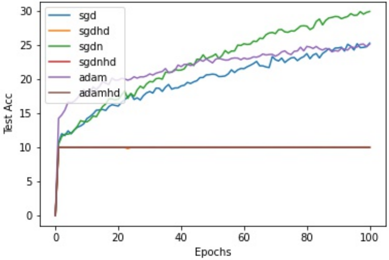


(b) Test accuracy.

Results: WideResNet on CIFAR10

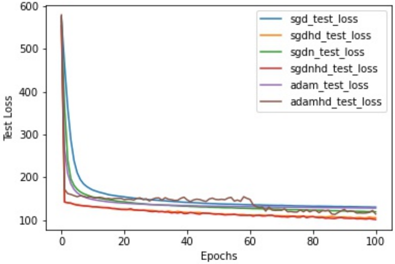


(a) Test Loss.

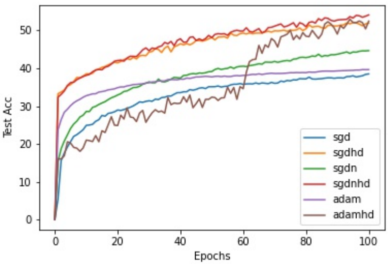


(b) Test accuracy.

Results: ResNet18 on CIFAR10

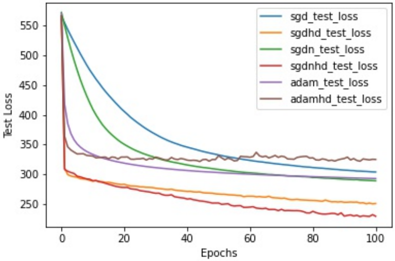


(a) Test Loss.

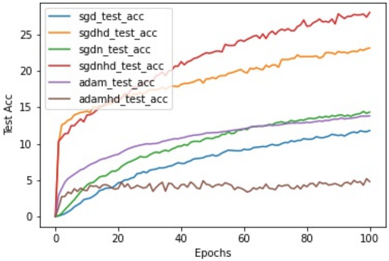


(b) Test accuracy.

Results: ResNet18 on CIFAR100

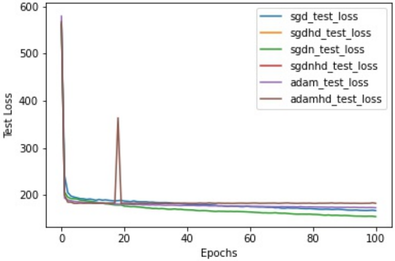


(a) Test Loss.

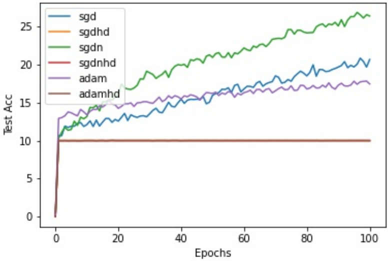


(b) Test accuracy.

Results: ResNet101 on CIFAR10

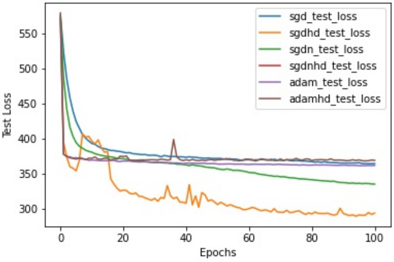


(a) Test Loss.

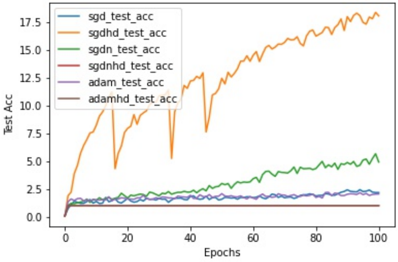


(b) Test accuracy.

Results: ResNet101 on CIFAR100



(a) Test Loss.



(b) Test accuracy.

Observations

- Performance on convex objectives shows a marked improvement but it does not generalise very well to non-convex objectives
- Addition of β hyper-parameter may seem like an additional tuning but for a good value of β the convergence becomes less sensitive to α_0
- The improvement is demonstrated even on algorithms like Adam that use adaptive learning rates
- Usage: For a reasonable range of α_0 hypergradient descent shows even better performance in terms of convergence when we use the β parameter .This can be thought of as a fine tuning parameter for finding optimal learning rate

Proof of Covergence

- We observe how the learning rate behaves when using hypergradient descent
- The learning rate grows initially and then shrinks to fluctuate around a small value
- In practice, we can smoothly transition to a fixed learning rate as the algorithm progresses
- For $\alpha_\infty \leq 1/L$ where
- Concretely, the step size we choose to update our parameters is defined as follows:

$$\gamma_t = \delta(t)\alpha_t + (1 - \delta(t))\alpha_\infty \tag{1}$$

Proof of Covergence

- $\delta(t)$ is chosen such that $t\delta(t) \to 0$ as $t \to \infty$
- Theorem: If the function is f-convex and L-Lipschitz smooth such that $\|\nabla f(\theta)\| \leq$ M then HD guarantees convergence if α

$$|\alpha_t| \leq |\alpha_0| + \beta \sum_{i=0}^{t-1} |\nabla f(\theta_{i+1})^T \nabla f(\theta_i)| \tag{2}$$

$$|\alpha_t| \leq |\alpha_0| + \beta \sum_{i=0}^{t-1} \|\nabla f(\theta_{i+1})\| \, \|\nabla f(\theta_i)\| \tag{3}$$

$$|\alpha_t| \leq |\alpha_0| + t\beta M^2 \tag{4}$$

Proof of Convergence

- Since α_t is upper bounded by a linear term, we have $\gamma_t \to \alpha_\infty$ as $\alpha_t \delta(t) \to 0$
- For large enough t, $\gamma_t \leq 1/L$, and we have convergence
- We have already shown (Lecture 11) for constant step size, the gradient descent converges if the above assumptions are held true (Karimi et al., 2016)

Thank you!
Any Question Please.

Reference I

Atılım Güneș Baydin, Robert Cornish, David Martínez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *Sixth International Conference on Learning Representations (ICLR), Vancouver, Canada, April 30 – May 3, 2018*, 2018.

Y. Bengio. Gradient-based optimization of hyperparameters. In *Neural Computation*.

Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer, 2016.