

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

- Problem Statement
- Hypergradient Descent (HD) Algorithm
- Optimization Algorithms
- Results
- Convergence Analysis
- CODE: <https://github.com/mohbattharani/Hypergradient-Descent>

- 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

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

- 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
 - 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

Hyper-gradient Descent (HD)

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

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

where α is the learning rate

Hyper-gradient Descent (HD)

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

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

where α is the learning rate

- We compute the partial derivative:

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

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

Hyper-gradient Descent (HD)

- Hence, the update for α becomes:

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

$$\alpha_t = \alpha_{t-1} + \beta \nabla f(\theta_{t-1})^T (\nabla f(\theta_{t-2})) \quad (3)$$

where β is the hypergradient learning rate

- Finally, the parameter update then becomes:

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

- 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

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

Algorithm: SGD vs SGD-HD

Algorithm 1 Stochastic gradient descent (SGD)

```
Require:  $\alpha, f(\theta), \theta_0$ 
 $t \leftarrow 0$ 
while  $\theta_t$  not converged do
   $t \leftarrow t + 1$ 
   $g_t \leftarrow \nabla f_t(\theta_{t-1})$ 
   $u_t \leftarrow -\alpha g_t$ 
   $\theta_t = \theta_{t-1} + u_t$ 
end while
return  $\theta_t$ 
```

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

```
Require:  $\alpha, f(\theta), \theta_0, \beta$ 
 $t, \nabla_{\alpha} u_0 \leftarrow 0, 0$ 
while  $\theta_t$  not converged do
   $t \leftarrow t + 1$ ,
   $g_t \leftarrow \nabla f_t(\theta_{t-1})$ 
   $h_t \leftarrow g_t \nabla_{\alpha} u_{t-1}$ 
   $\alpha_t \leftarrow \alpha_{t-1} - \beta h_t$ 
   $u_t \leftarrow -\alpha_t g_t$ 
   $\nabla_{\alpha} u_t \leftarrow -g_t$ 
   $\theta_t = \theta_{t-1} + u_t$ 
end while
return  $\theta_t$ 
```

Algorithm: SGDN vs SGDN-HD

Algorithm 3 SGD with Nesterov (SGDN)

Require: μ : momentum

$t, v_0 \leftarrow 0, 0$

Update rule:

$v_t \leftarrow \mu v_{t-1} + g_t$

$u_t \leftarrow -\alpha(g_t + \mu v_t)$

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

Require: μ : momentum

$t, v_0, \nabla_{\alpha} u_0 \leftarrow 0, 0, 0$

Update rule:

$v_t \leftarrow \mu v_{t-1} + g_t$

$u_t \leftarrow -\alpha_t(g_t + \mu v_t)$

$\nabla_{\alpha} u_t \leftarrow -(g_t + \mu v_t)$

Algorithm: Adam vs Adam-HD

Algorithm 5 Adam

Require: $\beta_1, \beta_2 \in [0, 1]$: decay rates for Adam
 $\mathbf{t}, \mathbf{m}_0, \mathbf{v}_0 \leftarrow 0, 0, 0$

Update rule:

$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$$

$$\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$$

$$\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$$

$$\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$$

$$\mathbf{u}_t \leftarrow -\alpha \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$$

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

Require: $\beta_1, \beta_2 \in [0, 1]$: decay rates for Adam
 $\mathbf{t}, \mathbf{m}_0, \mathbf{v}_0, \nabla_{\alpha} \mathbf{u}_0 \leftarrow 0, 0, 0, 0$

Update rule:

$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$$

$$\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$$

$$\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$$

$$\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$$

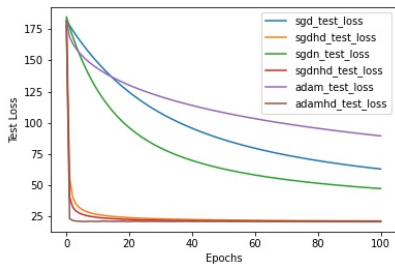
$$\mathbf{u}_t \leftarrow -\alpha \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$$

$$\nabla_{\alpha} \mathbf{u}_t \leftarrow -\mathbf{m}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$$

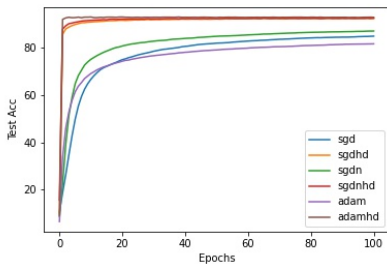
Model	Dataset		
	MNIST	CIFAR10	CIFAR100
Logistic Regression	✓	-	-
Multi-layer Preceptron	✓	✓	-
WideResNet	-	✓	-
ResNet18	-	✓	✓
ResNet101	-	✓	✓

Table 1: Dataset and models selected for evaluation

Results: Logistic Regression on MNIST

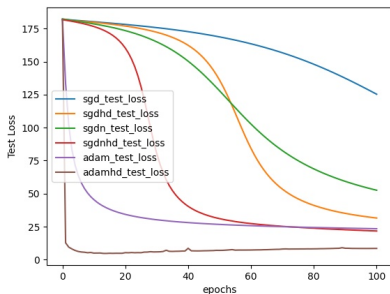


(a) Test Loss.

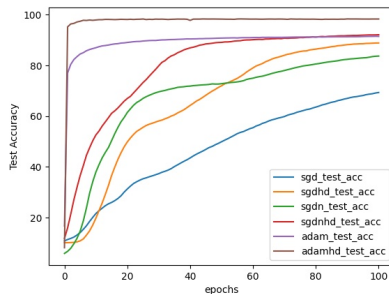


(b) Test accuracy.

Results: Multi layer Preceptron on MNIST

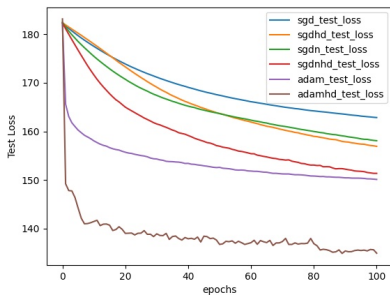


(a) Test Loss.

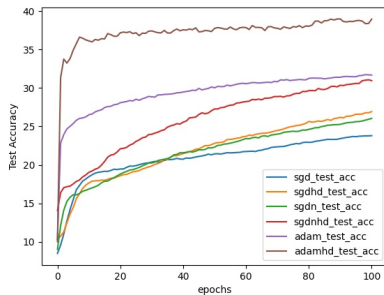


(b) Test accuracy.

Results: Multi layer Preceptron on CIFAR10

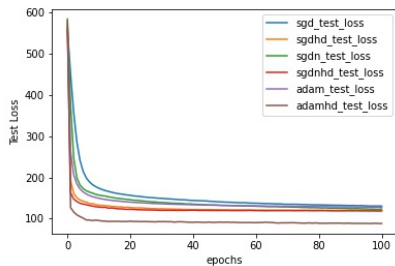


(a) Test Loss.

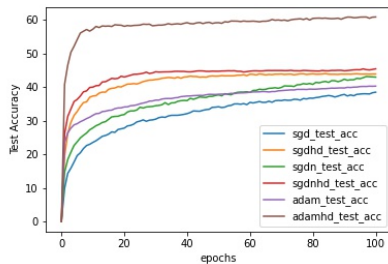


(b) Test accuracy.

Results: ResNet18 on CIFAR10

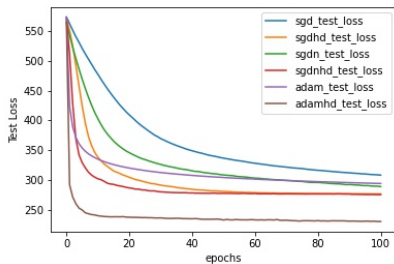


(a) Test Loss.

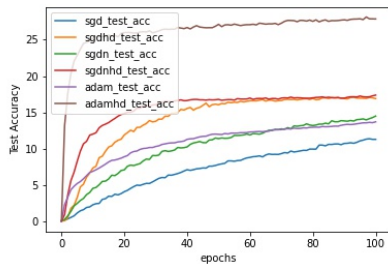


(b) Test accuracy.

Results: ResNet18 on CIFAR100

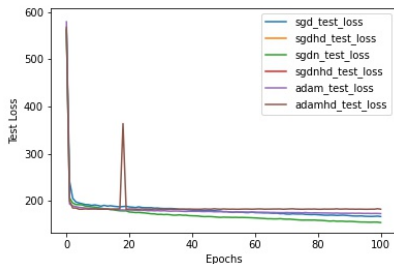


(a) Test Loss.

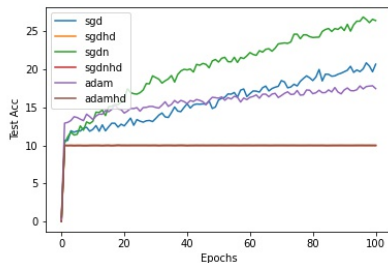


(b) Test accuracy.

Results: ResNet101 on CIFAR10

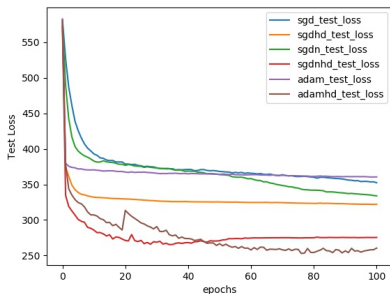


(a) Test Loss.

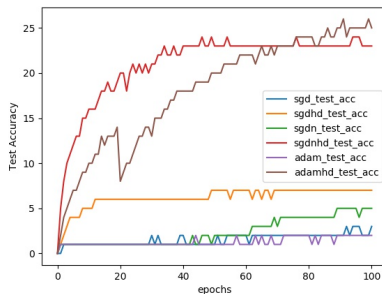


(b) Test accuracy.

Results: ResNet101 on CIFAR100

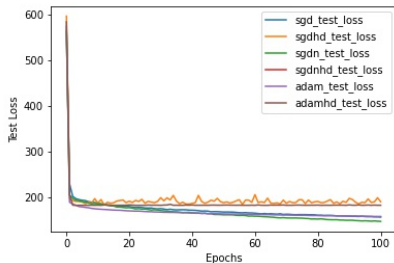


(a) Test Loss.

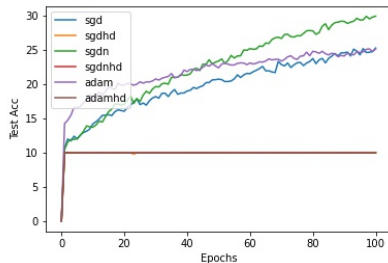


(b) Test accuracy.

Results: WideResNet on CIFAR10

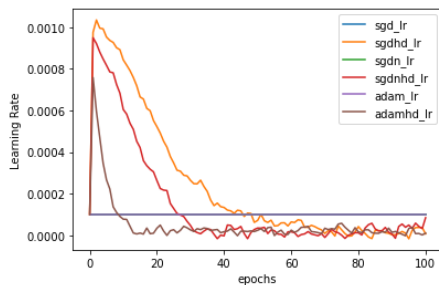


(a) Test Loss.

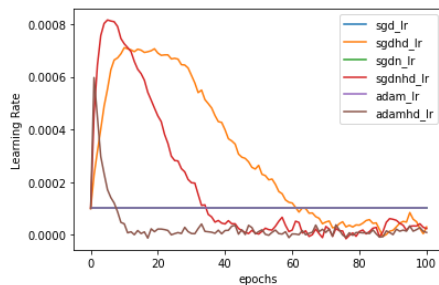


(b) Test accuracy.

Observations



(a) ResNet18 on CIFAR10.



(b) ResNet18 on CIFAR100.

Figure 9: For HD version of the optimizer, learning rate increases initially and then then fluctuates around a small value.

Observations



(a) SGD.



(b) SGD-HD.

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
- For proving convergence, we make a slight modification and smoothly transition to a fixed learning rate as the algorithm progresses
- Concretely, the step size we choose to update our parameters is defined as follows:

$$\gamma_t = \delta(t)\alpha_t + (1 - \delta(t))\alpha_\infty \quad (1)$$

Proof of Covergence

- For α_∞ chosen to be $\leq 1/L$
- $\delta(t)$ is chosen such that $t\delta(t) \rightarrow 0$ as $t \rightarrow \infty$
- Theorem: If the function is f -convex and L -Lipschitz smooth such that $\|\nabla f(\theta)\| \leq M$ then HD guarantees convergence if α

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

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

$$|\alpha_t| \leq |\alpha_0| + t\beta M^2 \quad (4)$$

Proof of Convergence

- Since α_t is upper bounded by a linear term, we have $\gamma_t \rightarrow \alpha_\infty$ as $\alpha_t \delta(t) \rightarrow 0$
- 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.

- Atılım Güneş 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.