# HYPERGRADIENT BASED OPTIMIZATION METHODS

## ABSTRACT

The method proposed in the paper ”Online Learning Rate Adaptation with Hypergradient Descent” automatically adjusts the learning rate to minimize some estimate of the expectation of the loss, by introducing the “hypergradient” - the gradient of any loss function w.r.t hyperparameter “eta” (the optimizer learning rate). We expect that the hypergradient based learning rate update could be more accurate and aim to exploit the gains much better by boosting the learning rate updates with momentum and adaptive gradients, experimenting with Hypergradient descent with momentum, and Adam with Hypergradient. The results of our experiments show that the new optimizers when evaluated against their hypergradient-descent baselines provide better generalization, faster convergence and better training stability.

## 1 INTRODUCTION

The performance of nearly all gradient based optimizers depends critically on how learning rates are tuned and in practice, getting good performance with the optimizers requires some manual adjustment of the initial value of the learning rate (or step size) as well as the design of an annealing schedule. Many attempts have been made to eliminate this need of manual tuning, in order to achieve faster and improved convergence of the objective function.

One such method proposed in the paper ”Online Learning Rate Adaptaion with Hypergradient Descent” (Atilim Gunes Baydin & Wood (2018)) automatically adjusts the learning rate (possibly different learning rates for different parameters), so as to minimize some estimate of the expectation of the loss. The paper introduces “hypergradient” - the gradient of any objective function w.r.t hyperparameter “eta” (the optimizer learning rate) and proposes the idea to learn the step-size via an update from gradient descent of the hypergradient at each training iteration. This learning rate update procedure when used alongside the model optimizers SGD, SGD with Nesterov (SGDN)(Ilya Sutskever) and Adam (Kingma & Ba (2014)) results in their hypergradient counterparts SGD-HD, SGDN-HD and Adam-HD, which demonstrate faster convergence of the loss and better generalization than solely using the original (plain) optimizers, and hence improving training.

But we expect that the hypergradient based learning rate update could be more accurate and aim to exploit the gains much better by boosting the learning rate updates with momentum and adaptive gradients, experimenting with **(i)** Hypergradient descent with momentum and **(ii)** Adam with Hypergradient, alongside the model optimizers SGD, SGD with Nesterov(SGDN) and Adam.

The algorithm first calculates the hypergradient depending on the model optimizer used, and then uses it to update the learning rate for near optimum value by either following sgd-momentum (Hypergradient descent with momentum) or adam (Adam with hypergradient) update rule.

The naming convention used is: $\{\textit{model optimizer}\}_{op}$-$\{\textit{learning rate optimizer}\}_{lop}$, following which we have $\{\textit{model optimizer}\}_{op}$-$\textit{SGDN}_{lop}$ (when the l.r. optimizer is hypergradient descent with momentum) and $\{\textit{model optimizer}\}_{op}$-$\textit{Adam}_{lop}$ (when the l.r. optimizer is adam with hypergradient).

The new optimizers and the respective hypergradient-descent baselines from which their performance is compared are:

- $\mathrm{SGD_{op}}$-$\mathrm{SGDN_{lop}}$, with baseline SGD-HD (i.e. $\mathrm{SGD_{op}}$-$\mathrm{SGD_{lop}}$)
- $\mathrm{SGDN_{op}}$-$\mathrm{SGDN_{lop}}$, with baseline SGDN-HD (i.e $\mathrm{SGDN_{op}}$-$\mathrm{SGD_{lop}}$)
- $\mathrm{Adam_{op}}$-$\mathrm{Adam_{lop}}$, with baseline Adam-HD (i.e $\mathrm{Adam_{op}}$-$\mathrm{SGD_{lop}}$)

The results show that these optimizers when evaluated against their respective hypergradient-descent baselines provide the following advantages:

- Better generalization
- Faster convergence
- Better training stability (less sensitive to the initial chosen learning rate)

## 2 Background

The parameter update equation for the three model optimizers given the objective function $f(\theta)$ and the gradient $g_t$ w.r.t. the parameters can be given as follows.

$\theta_t \leftarrow \theta_{t-1} - \alpha g_t$ (Stochaistic Gradient Descent)

$\theta_t \leftarrow \theta_{t-1} - \alpha(g_t + \mu v_t)$ where $v_t = \mu v_{t-1} + g_t$ (SGD with Nesterov)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ where $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$(Adam)

The three update rules can be summarised as : $\theta_t \leftarrow \theta_{t-1} + u_t$, where $u_t$ is the parameter update term, used in the hypergradient calculation.

### 2.1 The Hypergradient

The hypergradient is easy to calculate requiring only one extra copy of the gradient to be held in memory and very little computational overhead, allowing all the hypergradient based optimizers to be easily applicable in practice. Consider the parameter update as $\theta_t \leftarrow \theta_{t-1} + u_t$ and the objective function $f(\theta)$. Using $\theta_{t-1} \leftarrow \theta_{t-2} + u_{t-1}$ (the result of the previous update step), the hypergradient calculation (by the authors) is given as

$$h_t = \frac{\partial f(\theta_{t-1})}{\partial \alpha} = \nabla f(\theta_{t-1}) \cdot \frac{\partial(\theta_{t-2} + u_{t-1})}{\partial \alpha} = g_t \cdot \nabla_\alpha u_{t-1}$$

where $g_t = \nabla f(\theta_{t-1})$ is the gradient at timestep *t* and $\nabla_\alpha u_{t-1}$ for any optimizer represents the derivative of the parameter update term ($u_t$) w.r.t the learinng rate $\alpha$ at timestep *t-1*. Consequently, the hypergradient for the three optimizers can be written as

$\nabla_\alpha u_t = -g_t \Rightarrow$ $\quad \mathrm{h}_t = -g_t \cdot g_{t-1}$ (SGD)

$\nabla_\alpha u_t = -g_t - \mu v_t \Rightarrow$ $\quad \mathrm{h}_t = -g_t \cdot (g_{t-1} + \mu v_{t-1})$ (SGD with Nesterov (SGDN))

$\nabla_\alpha u_t = -\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon) \Rightarrow$ $\quad \mathrm{h}_t = -g_t \cdot \hat{m}_{t-1}/(\sqrt{\hat{v}_{t-1}} + \epsilon)$ (Adam)

### 2.2 Learning Rate Update

We now use the calculated hypergradient to get the new learning rate. The update rule for the learning rate proposed by the authors is given as $\alpha_t \leftarrow \alpha_{t-1} - \beta h_t$, where $h_t$ is the hypergradient at time step *t* and $\beta$ is the hypergradient learning rate.
The new algorithms for the learning rate update proposed by this work are given as 1 and 2.

**Algorithm 1** *Learning rate update using Hypergradient with momentum*, $\mu$ = 0.9 (default setting)

**Require:** $\alpha_0, \beta$: Initial learning rate, Hypergradient learning rate
**Require:** $\mu \in [0, 1]$: Momentum coefficient for the Hypergradient "velocity"
$\mathbf{v_0} \leftarrow 0$ (Initialize "velocity" for the hypergradient)
$t \leftarrow 0$ (Initial timestep)
**for** $t$ = 1,...,$T$ **do**
    $t \leftarrow t + 1$
    $h_t \leftarrow g_t \cdot \nabla_\alpha u_{t-1}$ (Hypergradient at timestep $t$ based on optimizer used for objective function)
    $v_t \leftarrow \mu v_{t-1} + h_t$ ("velocity")
    $\alpha_t \leftarrow \alpha_{t-1} - \beta(h_t + \mu v_t)$ (Update the learning rate)
**end for**

**Algorithm 2** *Learning rate update using adaptive Hypergradient*, $\beta_1$ = 0.9, $\beta_2$ = 0.999, $\epsilon$ = 1e-8 (default setting)

**Require:** $\alpha_0, \beta$: Initial learning rate, Hypergradient learning rate
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for Hypergradient moment estimates
$\mathbf{m_0} \leftarrow 0$ (Initialize first moment for the hypergradient)
$\mathbf{v_0} \leftarrow 0$ (Initialize second moment for the hypergradient)
$t \leftarrow 0$ (Initial timestep)
**for** $t$ = 1,...,$T$ **do**
    $t \leftarrow t + 1$
    $h_t \leftarrow g_t \cdot \nabla_\alpha u_{t-1}$ (Hypergradient at timestep $t$ based on optimizer used for objective function)
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) h_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) h_t^2$
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\alpha_t \leftarrow \alpha_{t-1} - \beta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update the learning rate)
**end for**

For a particular iteration, first the updated learning rate $\alpha_t$ at timestep $t$ is calculated and then for parameter update. The modified parameter update equation for instance, for the SGD model optimizer becomes

$$\theta_t \leftarrow \theta_{t-1} - \alpha_t g_t$$

## 3 Experimental Results

For comparison, the experiment configurations (hyperparameter $\alpha_0$ and $\beta$) are set same as the hypergradient-descent baseline versions of the optimizers proposed by the authors.
Training is performed for 20 epochs for Logistic Regression, 100 epochs for Multi-Layer Perceptron and 200 epochs for VGGNet. For all the optimizers, the value of $\alpha_0$ is taken as 0.001 in each model. For the optimizers SGD-HD, $\text{SGD}_{\text{op}}$-$\text{SGDN}_{\text{lop}}$, SGDN-HD and $\text{SGDN}_{\text{op}}$-$\text{SGDN}_{\text{lop}}$, $\beta$ is set as 0.001 for each model. For Adam-HD and $\text{Adam}_{\text{op}}$-$\text{Adam}_{\text{lop}}$, $\beta$ is set as 1e-7 for Logreg and MLP models, and 1e-8 for VGGNet.
The results show that the new optimizers perform better than their hypergradient-descent baselines by achieving faster loss convergence and better generalization when evaluated for Logistic Regression and Multi-Layer Perceptron (MLP) on MNIST, and VGGNet(Simonyan & Zisserman (2014)) on CIFAR-10. This behaviour is consistent across the training and validation loss for the optimizers. The trends for the learning rate are also plotted as shown in Figure 1.

## 4 Insensitivity to $\alpha_0$

$\alpha_0$ refers to the initial value of the learning rate and experiments reveal that the performance of the new optimizers are even less sensitive to $\alpha_0$ compared to their hypergradient counterparts for a given value of beta, as the algorithm quickly adjusts it to a good value.

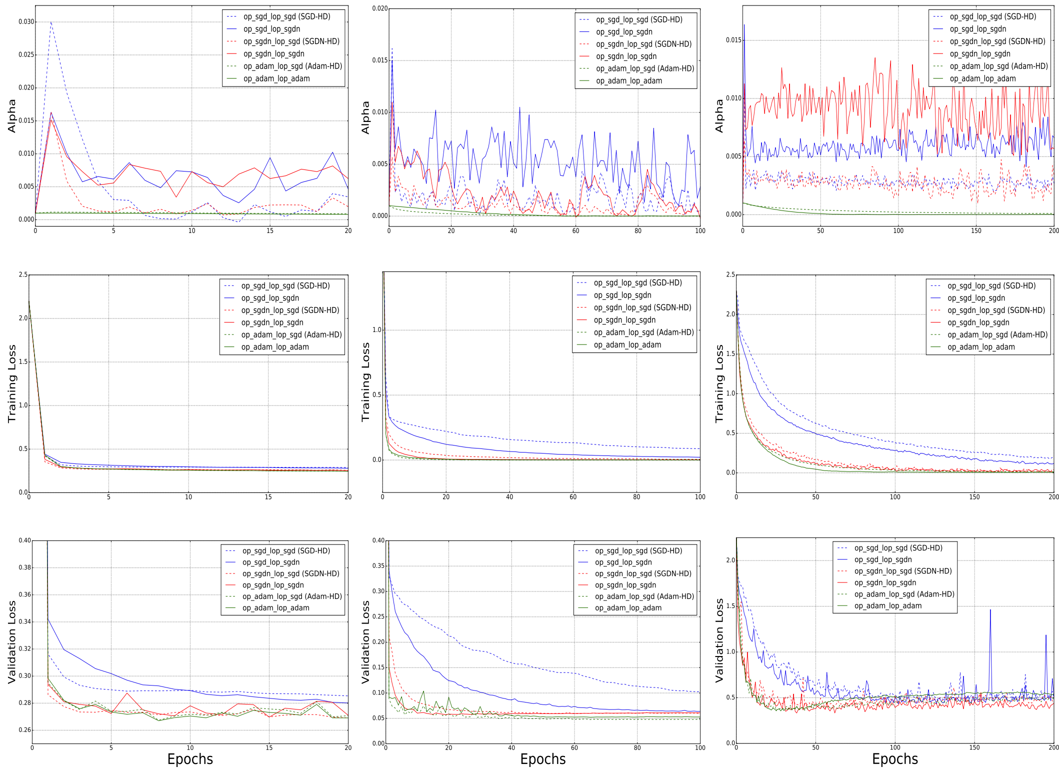


Figure 1: Behavior of the optimizers compared with their hypergradient-descent baselines. Columns: left: logistic regression on MNIST; middle: multi-layer neural network on MNIST; right: VGG Net on CIFAR-10. Rows: top: evolution of the learning rate $\alpha_t$ ; middle: training loss; bottom: validation loss.

For each of the new optimizers, experiments are run for five different values of $\alpha_0$ in (0.01, 0.005, 0.001, 0.0005, 0.0001), and number of iterations required for the training loss to reduce below a certain threshold - 0.75 for SGD-HD & $SGD_{op}$-$SGDN_{lop}$, 0.3 for SGDN-HD & $SGDN_{op}$-$SGDN_{lop}$ and 0.2 for Adam-HD & $Adam_{op}$-$Adam_{lop}$ is observed.

## References


David Martinez Rubio Mark Schmidt Atilim Gunes Baydin, Robert Cornish and Frank Wood. Online learning rate adaptation with hypergradient descent, 2018.

George Dahl Geoffrey Hinton Ilya Sutskever, James Martens. On the importance of initialization and momentum in deep learning.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.