Online Learning Rate Adaptation with Hypergradient Descent

Atılım Güneş Baydin,[1] Robert Cornish,[1] David Martínez Rubio,[1] Mark Schmidt,[2] Frank Wood[2]
[1]Department of Engineering Science, University of Oxford, [2]Department of Computer Science, University of British Columbia
{gunes,rcornish,damaru}@robots.ox.ac.uk, {schmidtm,fwood}@cs.ubc.ca

What is a "hypergradient"?

A *hypergradient* is a derivative—of the loss or another quantity—taken with respect to a hyperparameter such as the learning rate, momentum, or regularization parameters of an optimization algorithm [1].

Hypergradient descent

Hypergradient descent (HD) is the gradient-based optimization of a hyperparameter (in this work, a scalar learning rate). Given a regular gradient descent algorithm with loss function f, parameters θ_t, and learning rate α

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

we derive the partial derivative of f with respect to the learning rate α:

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

requiring only a dot product and the memory cost of an extra copy of the gradient. Using this hypergradient, we construct a higher-level update rule for the learning rate as

$$\alpha_t = \alpha_{t-1} - \beta \frac{\partial f(\theta_{t-1})}{\partial \alpha} = \alpha_{t-1} + \beta \nabla f(\theta_{t-1}) \cdot \nabla f(\theta_{t-2}) \,, \tag{3}$$

introducing β as the hypergradient learning rate. We then modify Eq. 1 to use the sequence α_t to become $\theta_t = \theta_{t-1} - \alpha_t \nabla f(\theta_{t-1})$.

The technique is general: we demonstrate applying it to stochastic gradient descent (SGD), SGD with Nesterov momentum, and Adam [2].

Algorithm 1. Stochastic gradient descent (SGD)

Require: α: learning rate
Require: $f(\theta)$: loss function
Require: θ_0: initial parameter vector
$t \leftarrow 0$ ▷ Initialization
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla f_t(\theta_{t-1})$ ▷ Gradient
$u_t \leftarrow -\alpha\, g_t$ ▷ Parameter update
$\theta_t \leftarrow \theta_{t-1} + u_t$ ▷ Apply parameter update
end while
return θ_t

Algorithm 2. SGD with Nesterov momentum (SGDN)

Require: μ: momentum
$t, v_0 \leftarrow 0, 0$ ▷ Initialization
Update rule:
$v_t \leftarrow \mu\, v_{t-1} + g_t$ ▷ "Velocity"
$u_t \leftarrow -\alpha\,(g_t + \mu\, v_t)$ ▷ Parameter update

Algorithm 3. Adam

Require: $\beta_1, \beta_2 \in [0, 1)$: decay rates for Adam
$t, m_0, v_0 \leftarrow 0, 0, 0$ ▷ Initialization
Update rule:
$m_t \leftarrow \beta_1\, m_{t-1} + (1 - \beta_1)\, g_t$ ▷ 1st mom. estimate
$v_t \leftarrow \beta_2\, v_{t-1} + (1 - \beta_2)\, g_t^2$ ▷ 2nd mom. estimate
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Bias correction
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Bias correction
$u_t \leftarrow -\alpha\, \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ ▷ Parameter update

Algorithm 4. SGD with hypergradient descent (SGD-HD)

Require: α_0: initial learning rate
Require: $f(\theta)$: loss function
Require: θ_0: initial parameter vector
Require: β: hypergradient learning rate
$t, \nabla_\alpha u_0 \leftarrow 0, 0$ ▷ Initialization
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla f_t(\theta_{t-1})$ ▷ Gradient
$h_t \leftarrow g_t \cdot \nabla_\alpha u_{t-1}$ ▷ Hypergradient
$\alpha_t \leftarrow \alpha_{t-1} - \beta\, h_t$ ▷ Learning rate update
Or, alternative to the line above:
$\alpha_t \leftarrow \alpha_{t-1} \left(1 - \beta \frac{h_t}{g_t \nabla_\alpha u_{t-1}}\right)$ ▷ Multiplicative update
$u_t \leftarrow -\alpha_t\, g_t$ ▷ Parameter update
$\nabla_\alpha u_t \leftarrow -g_t$
$\theta_t \leftarrow \theta_{t-1} + u_t$ ▷ Apply parameter update
end while
return θ_t

Algorithm 5. SGDN with hypergradient descent (SGDN-HD)

Require: μ: momentum
$t, v_0, \nabla_\alpha u_0 \leftarrow 0, 0, 0$ ▷ Initialization
Update rule:
$v_t \leftarrow \mu\, v_{t-1} + g_t$ ▷ "Velocity"
$u_t \leftarrow -\alpha_t\,(g_t + \mu\, v_t)$ ▷ Parameter update
$\nabla_\alpha u_t \leftarrow -g_t - \mu\, v_t$

Algorithm 6. Adam with hypergradient descent (Adam-HD)

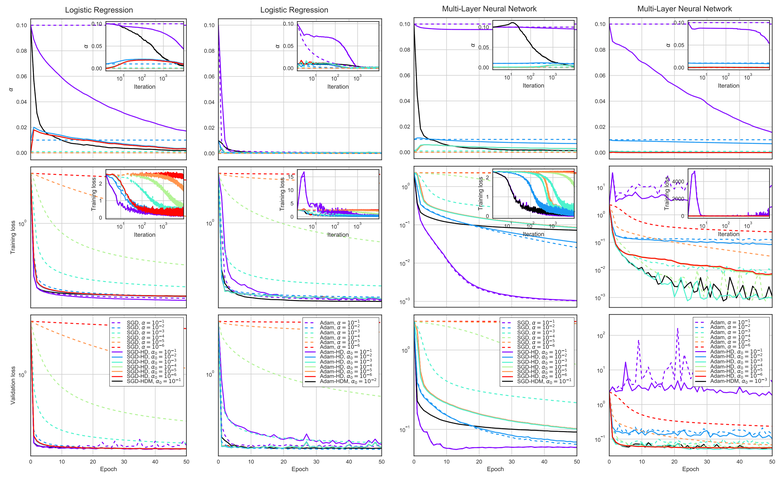
Require: $\beta_1, \beta_2 \in [0, 1)$: decay rates for Adam
$t, m_0, v_0, \nabla_\alpha u_0 \leftarrow 0, 0, 0, 0$ ▷ Initialization
Update rule:
$m_t \leftarrow \beta_1\, m_{t-1} + (1 - \beta_1)\, g_t$ ▷ 1st mom. estimate
$v_t \leftarrow \beta_2\, v_{t-1} + (1 - \beta_2)\, g_t^2$ ▷ 2nd mom. estimate
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Bias correction
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Bias correction
$u_t \leftarrow -\alpha_t\, \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ ▷ Parameter update
$\nabla_\alpha u_t \leftarrow -\widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$

Experiments with online learning rate adaptation

By performing online updates with hypergradients, we get algorithms that

- **adapt α_t on-the-fly within the main optimization loop**, and
- **significantly reduce the need to tune the initial learning rate α_0.**

For a given untuned initial learning rate, HD algorithms consistently bring the loss trajectory closer to the optimal one that would be attained by the baseline algorithm with a tuned initial learning rate.



Top row: learning rate, middle row: training loss, bottom row: validation loss. Dashed: regular SGD and Adam, Solid: SGD-HD and Adam-HD. $\alpha_0 \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ and $\beta = 10^{-4}$.

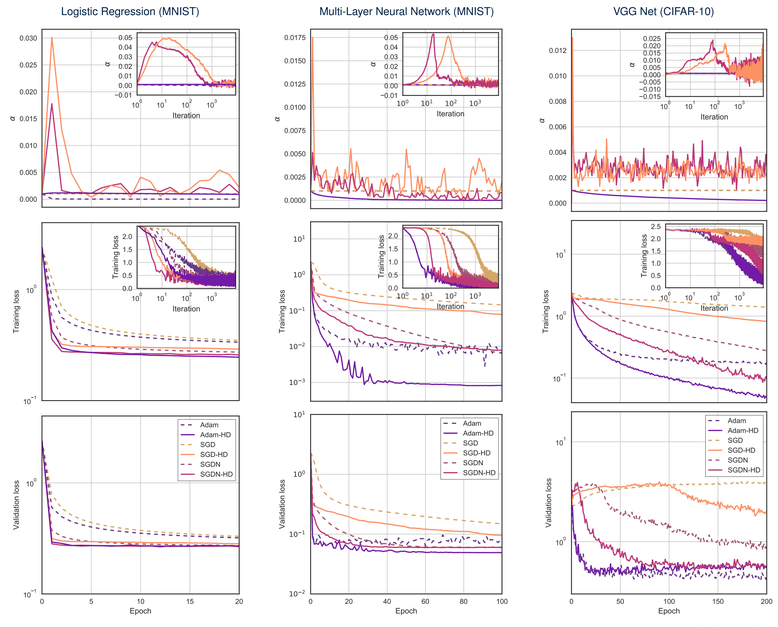
Everywhere except very large β and α_0 (shaded red below), HD (bottom) performs significantly better than, or in the worst case equal to, the baseline (top). In the limit $\beta \to 0$, HD is identical to the baseline, making it safe to pick arbitrary small β. With an arbitrary small β (no tuning), HD always improves upon the baseline started at the same α_0. In practice, **even starting from $\alpha_0 = 0$ works.**

Iterations to convergence for logistic regression (similar results hold for multi-layer NN and CNN).

Tuning example

We present tuning examples for logistic regression (on MNIST), multi-layer neural network (on MNIST), and VGG Net (on CIFAR-10). Starting from a low $\alpha_0 = 0.001$, we observe:

- **Increase phase**: α_t spontaneously increases in a bounded way.
- **Decay phase**: α_t decays to a low non-zero value (epoch average) and fluctuates around it. $\alpha_t < 0$ (unlearning) can happen for some minibatches within the epoch.
- The behavior of α_t in Adam-HD is a spontaneous smooth decay towards zero, which makes a significant difference in the training loss.



Top row: learning rate, middle row: training loss, bottom row: validation loss. Dashed: regular SGD, SGDN, and Adam, Solid: SGD-HD, SGDN-HD, and Adam-HD. $\alpha_0 = 0.001$, $\beta = 0.001$ (SGD-HD and SGDN-HD), and $\beta = 10^{-8}$ (Adam-HD).

We are working towards explaining these empirical findings and taking HD further with higher-order hypergradients. Please tell us what you think!

References

[1] D. Maclaurin, D. K. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015.
[2] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
[3] Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.
[4] J. Domke. Generic methods for optimization-based modeling. In *AISTATS*, 2012.
[5] L. B. Almeida, T. Langlois, J. D. Amaral, and A. Plakhov. Parameter adaptation in stochastic optimization. In D. Saad, editor, *On-Line Learning in Neural Networks*. Cambridge University Press, 1998.

Code

https://github.com/gbaydin/hypergradient-descent