CPEN 455: Deep Learning

Lecture 4.2: Backpropagation II

Renjie Liao

University of British Columbia

Winter, Term 1, 2026

Outline

- Learning Algorithm for Feedforward Neural Networks:
  - Backpropagation
  - Weight Initialization
  - Learning Rate, Momentum, Adam & Schedules
  - Weight Decay
  - Matrix Optimization: Muon
  - Early Stopping

Outline

- Learning Algorithm for Feedforward Neural Networks:
  - Backpropagation
  - Weight Initialization
  - **Learning Rate, Momentum, Adam & Schedules**
  - Weight Decay
  - Matrix Optimization: Muon
  - Early Stopping

Backpropagation (BP)

Now we know how to compute gradients, let us see how to perform gradient-based learning (e.g., SGD)

Algorithm 1 SGD

Input: step T, learning Rate η, initial W^0, batch size B
For $t = 1, \cdots, T$
  Get mini-batch data (X^t, Y^t)
  $L(W^{t-1}, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$
  $W^t = W^{t-1} - \eta \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$
Return W^T

Backpropagation (BP)

Now we know how to compute gradients, let us see how to perform gradient-based learning (e.g., SGD)

Algorithm 1 SGD

1: **Input**: step T, learning Rate η, initial W^0, batch size B
2: **For** $t = 1, \cdots, T$
3: Get mini-batch data (X^t, Y^t)
4: $L(W^{t-1}, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$
5: $W^t = W^{t-1} - \eta \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$
6: **Return** W^T

Forward pass

Backpropagation (BP)

Now we know how to compute gradients, let us see how to perform gradient-based learning (e.g., SGD)

Algorithm 1 SGD

1: **Input**: step T, learning Rate η, initial W^0, batch size B
2: **For** $t = 1, \cdots, T$
3: Get mini-batch data (X^t, Y^t)
4: $L(W^{t-1}, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$ — Forward pass
5: $W^t = W^{t-1} - \eta \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$ — Backward pass
6: **Return** W^T

Backpropagation (BP)

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1, 2].

Backpropagation (BP)

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1, 2].

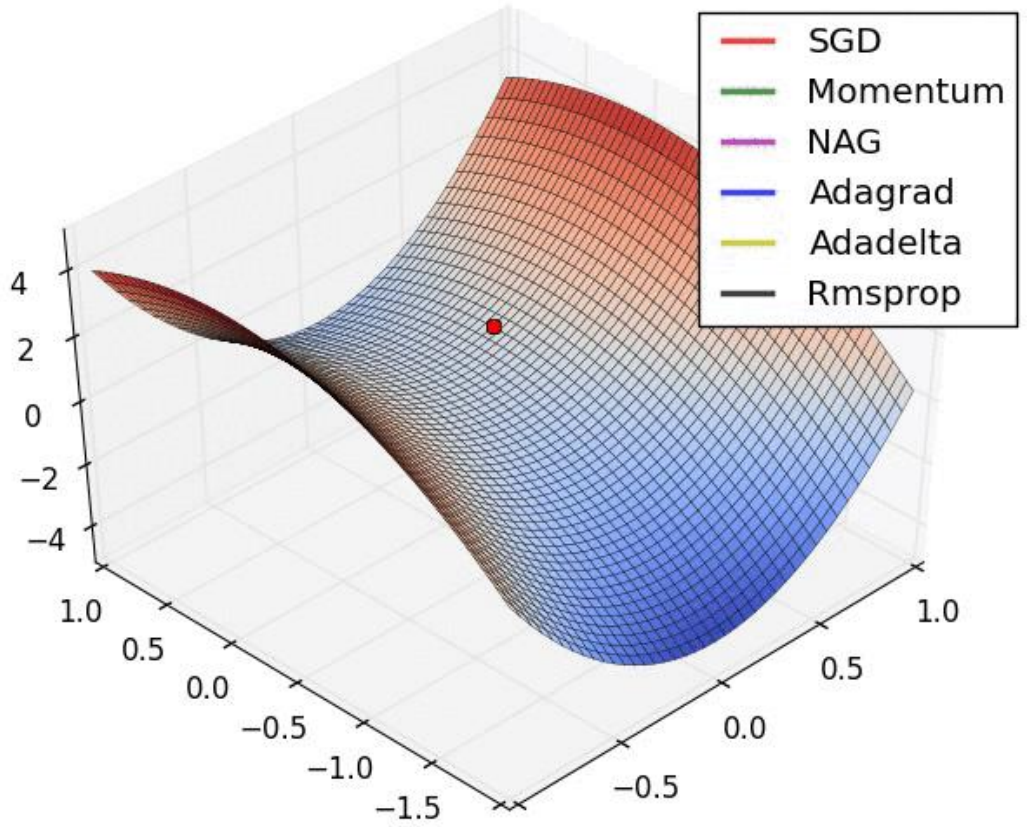


Image Credit: [1]

BP + Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1, 2].

[3]: Gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima.

Image Credit: [1]

BP + Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1, 2].

[3]: Gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima.

Algorithm 1 SGD with Momentum

Input: step T, learning Rate η, initial W^0, batch size B, momentum β, initial M^0

For $t = 1, \cdots, T$

  Get mini-batch data (X^t, Y^t)

  $L(W^{t-1}, X^t, Y^t) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$

  $M^t = \beta M^{t-1} + \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$

  $W^t = W^{t-1} - \eta M^t$

Return W^T

BP + Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1, 2].

[3]: Gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima.

Algorithm 1 SGD with Momentum

1: **Input**: step T, learning Rate η, initial W^0, batch size B, momentum β, initial M^0
2: **For** $t = 1, \cdots, T$
3: Get mini-batch data (X^t, Y^t)
4: $L(W^{t-1}, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$
5: $M^t = \beta M^{t-1} + \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$
6: $W^t = W^{t-1} - \eta M^t$
7: **Return** W^T

$$M^t = \beta M^{t-1} + \eta \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$$
$$W^t = W^{t-1} - M^t$$

Alternative form (so-called Nesterov momentum) in the literature, see, e.g., [4]

BP + Momentum

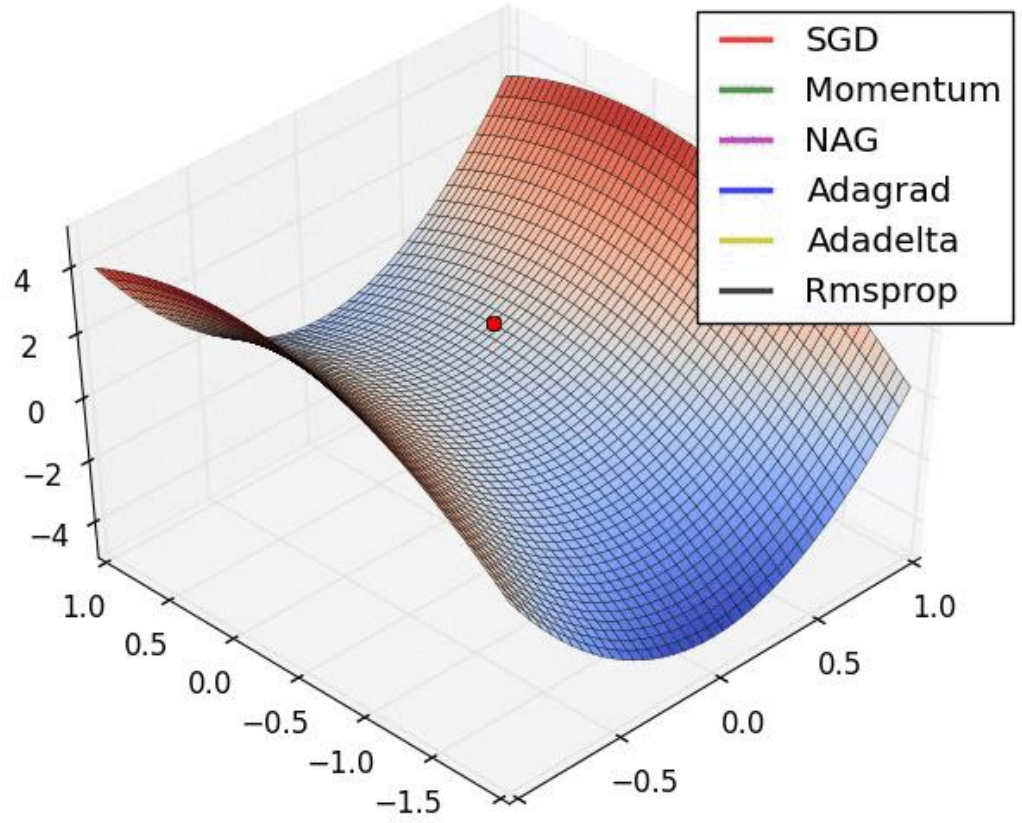


Image Credit: [1]

BP + Momentum

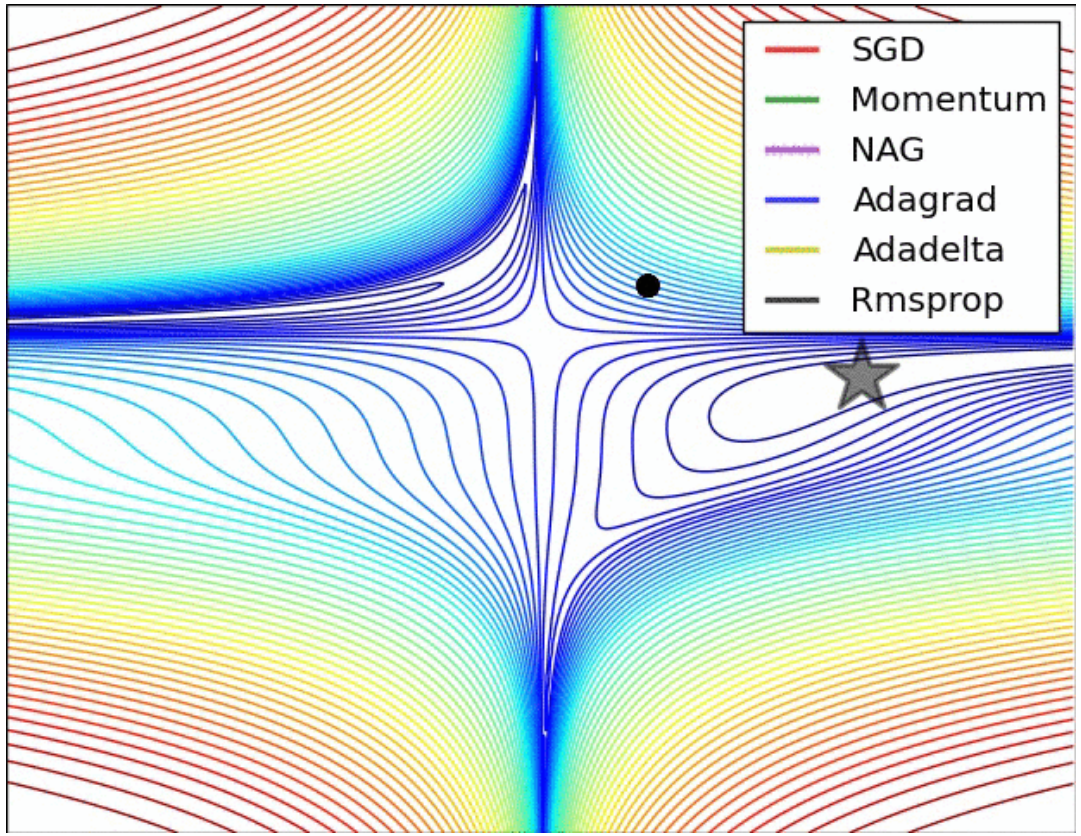


Image Credit: [1]

BP + Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1, 2].

[3]: Gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima.

Algorithm 1 SGD with Momentum

1: **Input**: step T, learning Rate η, initial W^0, batch size B, momentum β, initial M^0
2: **For** $t = 1, \cdots, T$
3: Get mini-batch data (X^t, Y^t)
4: $L(W^{t-1}, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$
5: $M^t = \beta M^{t-1} + \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$
6: $W^t = W^{t-1} - \eta M^t$
7: **Return** W^T

Demo: https://distill.pub/2017/momentum/

BP + Adaptive Learning Rate

Denoting $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$, recall in GD/SGD, we have the following update rule:

$$W^t = W^{t-1} - \eta g^t$$

How can we tune the learning rate?

BP + Adaptive Learning Rate

Denoting $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$, recall in GD/SGD, we have the following update rule:

$$W^t = W^{t-1} - \eta g^t$$

How can we tune the learning rate?

AdaGrad (Adaptive Gradient) [5] proposes to assign larger learning rates to infrequently updated weights and smaller ones to frequently updated weights:

$$G^t = \sum_{\tau=1}^{t} g^\tau g^{\tau\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

BP + Adaptive Learning Rate

Denoting $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$, recall in GD/SGD, we have the following update rule:

$$W^t = W^{t-1} - \eta g^t$$

How can we tune the learning rate?

AdaGrad (Adaptive Gradient) [5] proposes to assign larger learning rates to infrequently updated weights and smaller ones to frequently updated weights:

$$G^t = \sum_{\tau=1}^{t} g^\tau g^{\tau\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

Let us look at the element-wise update rule of AdaGrad:

$$W^t[i] = W^{t-1}[i] - \frac{\eta}{\sqrt{\epsilon + G^t[i,i]}} g^t[i]$$

BP + Adaptive Learning Rate

Denoting $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$, recall in GD/SGD, we have the following update rule:

$$W^t = W^{t-1} - \eta g^t$$

How can we tune the learning rate?

AdaGrad (Adaptive Gradient) [5] proposes to assign larger learning rates to infrequently updated weights and smaller ones to frequently updated weights:

$$G^t = \sum_{\tau=1}^{t} g^\tau g^{\tau\top}$$

In practice, we only need to compute diagonal terms $g^t \odot g^t$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

Let us look at the element-wise update rule of AdaGrad:

$$W^t[i] = W^{t-1}[i] - \frac{\eta}{\sqrt{\epsilon + G^t[i,i]}} g^t[i]$$

BP + Adaptive Learning Rate

AdaGrad (Adaptive Gradient) [5] proposes to assign larger learning rates to infrequently updated weights and smaller ones to frequently updated weights:

$$G^t = \sum_{\tau=1}^{t} g^\tau g^{\tau\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

Since we accumulate (squared) gradients from the beginning, our learning rate is always decaying and could vanish before we converge.

BP + Adaptive Learning Rate

AdaGrad (Adaptive Gradient) [5] proposes to assign larger learning rates to infrequently updated weights and smaller ones to frequently updated weights:

$$G^t = \sum_{\tau=1}^{t} g^\tau g^{\tau\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

Since we accumulate (squared) gradients from the beginning, our learning rate is always decaying and could vanish before we converge. To deal with this, RMSProp [6] proposes to use exponential moving average (EMA) of past squared gradients:

$$G^t = \rho G^{t-1} + (1-\rho) g^t g^{t\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

BP + Adaptive Learning Rate

AdaGrad (Adaptive Gradient) [5] proposes to assign larger learning rates to infrequently updated weights and smaller ones to frequently updated weights:

$$G^t = \sum_{\tau=1}^{t} g^\tau g^{\tau\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

Since we accumulate (squared) gradients from the beginning, our learning rate is always decaying and could vanish before we converge. To deal with this, RMSProp [6] proposes to use exponential moving average (EMA) of past squared gradients:

$$G^t = \rho G^{t-1} + (1-\rho) g^t g^{t\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

Adadelta [7] leverages the same EMA trick (denominator) and additional weighting (numerator) based on parameter updates.

Comparison Again

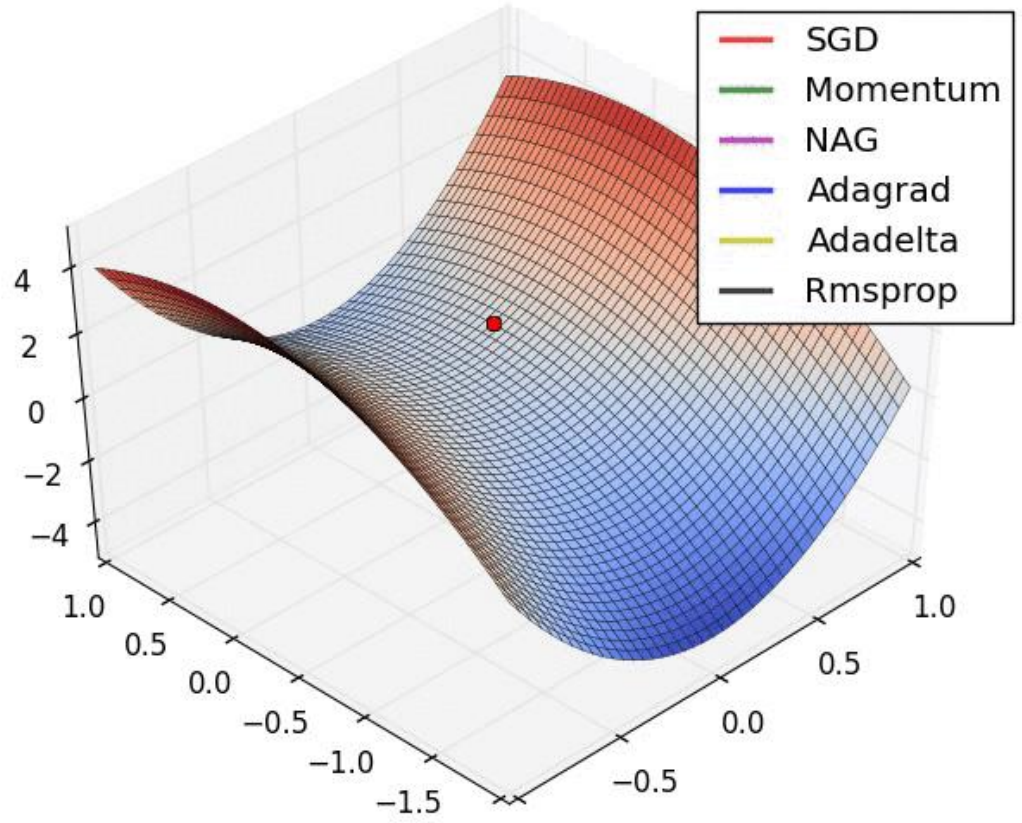


Image Credit: [1]

Comparison Again

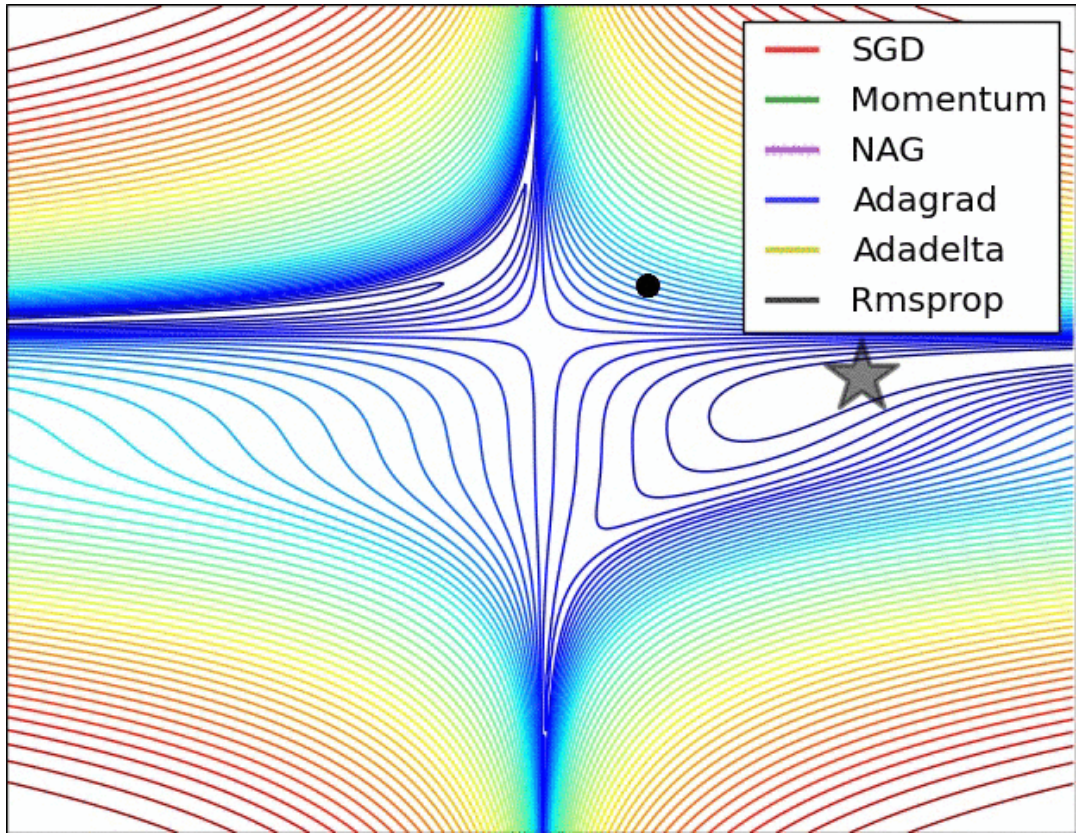


Image Credit: [1]

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t {g^t}^\top$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t {g^t}^\top$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

In practice, we only need to compute diagonal terms $g^t \odot g^t$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

In practice, we only need to compute diagonal terms $g^t \odot g^t$

These are powers of scalars (abuse of superscript)

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1 - \beta_1) g^t \quad \Longrightarrow \quad \tilde{g}^t = g^t$$

$$G^t = \beta_2 G^{t-1} + (1 - \beta_2) g^t {g^t}^\top$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t}$$

When $\beta_1 = 0$, Adam is almost the same as RMSProp besides the correction term (red box)

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1 - \beta_1) g^t \quad \Longrightarrow \quad \tilde{g}^t = g^t$$

$$G^t = \beta_2 G^{t-1} + (1 - \beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t}$$

When $\beta_1 = 0$, Adam is almost the same as RMSProp besides the correction term (red box):

$$G^t = \rho G^{t-1} + (1 - \rho) g^t g^{t\top}$$

$$W^t = W^{t-1} - \eta \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} g^t$$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t {g^t}^\top$$

$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$
$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t {g^t}^\top$$
$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$
$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

$$\begin{aligned}
\mathbb{E}\left[\tilde{g}^t\right] &= \mathbb{E}\left[\beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t\right] = \mathbb{E}\left[\beta_1\left(\beta_1 \tilde{g}^{t-2} + (1-\beta_1) g^{t-1}\right) + (1-\beta_1) g^t\right] \\
&= \mathbb{E}\left[\beta_1^2 \tilde{g}^{t-2} + \beta_1(1-\beta_1) g^{t-1} + (1-\beta_1) g^t\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{t} \beta_1^{t-i}(1-\beta_1) g^i + \beta_1^t \tilde{g}^0\right]
\end{aligned}$$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$
$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$
$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$
$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

$$\begin{aligned}
\mathbb{E}\left[\tilde{g}^t\right] &= \mathbb{E}\left[\beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t\right] = \mathbb{E}\left[\beta_1\left(\beta_1 \tilde{g}^{t-2} + (1-\beta_1) g^{t-1}\right) + (1-\beta_1) g^t\right] \\
&= \mathbb{E}\left[\beta_1^2 \tilde{g}^{t-2} + \beta_1(1-\beta_1) g^{t-1} + (1-\beta_1) g^t\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{t} \beta_1^{t-i}(1-\beta_1) g^i + \beta_1^t \tilde{g}^0\right]
\end{aligned}$$

We ignore this term due to the zero initialization and assume gradients at each time step are stationary!

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$
$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t {g^t}^\top$$
$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$
$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

$$\begin{aligned}
\mathbb{E}\left[\tilde{g}^t\right] &= \mathbb{E}\left[\beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t\right] = \mathbb{E}\left[\beta_1\left(\beta_1 \tilde{g}^{t-2} + (1-\beta_1) g^{t-1}\right) + (1-\beta_1) g^t\right] \\
&= \mathbb{E}\left[\beta_1^2 \tilde{g}^{t-2} + \beta_1(1-\beta_1) g^{t-1} + (1-\beta_1) g^t\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{t} \beta_1^{t-i}(1-\beta_1) g^i + \beta_1^t \tilde{g}^0\right] \\
&\approx \sum_{i=1}^{t} \beta_1^{t-i}(1-\beta_1)\mathbb{E}\left[g^i\right] \\
&= \left(\sum_{i=0}^{t-1} \beta_1^i\right)(1-\beta_1)\mathbb{E}\left[g^i\right] = (1-\beta_1^t)\mathbb{E}\left[g^i\right]
\end{aligned}$$

We ignore this term due to the zero initialization and assume gradients at each time step are stationary!

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$
$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$
$$W^t = W^{t-1} - \eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t$$
$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\mathbb{E}\left[G^t\right] \approx (1-\beta_2^t)\mathbb{E}\left[g^i g^{i\top}\right]$ $\mathbb{E}\left[g^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \boxed{\eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t} \qquad \text{denoted as } \Delta W^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\mathbb{E}\left[G^t\right] \approx (1-\beta_2^t)\mathbb{E}\left[g^i g^{i\top}\right]$ $\mathbb{E}\left[g^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]$

Check the update (blue box) element-wise:

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \boxed{\eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t} \qquad \text{denoted as } \Delta W^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\mathbb{E}\left[G^t[j,j]\right] \approx (1-\beta_2^t)\mathbb{E}\left[\left(g^i[j]\right)^2\right]$ $\mathbb{E}\left[\tilde{g}^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]$

Check the update (blue box) element-wise:

$$\mathbb{E}[\Delta W^t[j]] = \mathbb{E}\left[\eta_t \frac{\tilde{g}^t[j]}{\sqrt{\epsilon + G^t[j,j]}}\right] \boxed{\approx \eta_t \mathbb{E}\left[\frac{\tilde{g}^t[j]}{\sqrt{G^t[j,j]}}\right]}$$

ignore epsilon

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \boxed{\eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t} \qquad \text{denoted as } \Delta W^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\mathbb{E}\left[G^t[j,j]\right] \approx (1-\beta_2^t)\mathbb{E}\left[\left(g^i[j]\right)^2\right]$ $\mathbb{E}\left[\tilde{g}^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]$

Check the update (blue box) element-wise:

$$\mathbb{E}[\Delta W^t[j]] = \mathbb{E}\left[\eta_t \frac{\tilde{g}^t[j]}{\sqrt{\epsilon + G^t[j,j]}}\right] \approx \eta_t \mathbb{E}\left[\frac{\tilde{g}^t[j]}{\sqrt{G^t[j,j]}}\right]$$

Independence assumption

$$\boxed{\approx \eta_t \mathbb{E}[\tilde{g}^t[j]]\, \mathbb{E}\left[\frac{1}{\sqrt{G^t[j,j]}}\right]}$$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \boxed{\eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t} \qquad \text{denoted as } \Delta W^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\mathbb{E}\left[G^t[j,j]\right] \approx (1-\beta_2^t)\mathbb{E}\left[\left(g^i[j]\right)^2\right]$ $\mathbb{E}\left[\tilde{g}^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]$

Check the update (blue box) element-wise:

$$\mathbb{E}[\Delta W^t[j]] = \mathbb{E}\left[\eta_t \frac{\tilde{g}^t[j]}{\sqrt{\epsilon + G^t[j,j]}}\right] \approx \eta_t \mathbb{E}\left[\frac{\tilde{g}^t[j]}{\sqrt{G^t[j,j]}}\right]$$

$$\approx \eta_t \mathbb{E}[\tilde{g}^t[j]]\, \mathbb{E}\left[\frac{1}{\sqrt{G^t[j,j]}}\right] \boxed{\approx \eta_t \frac{\mathbb{E}[\tilde{g}^t[j]]}{\sqrt{\mathbb{E}[G^t[j,j]]}}}$$

Second moment concentrated away from zero

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \boxed{\eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t} \qquad \text{denoted as } \Delta W^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\boxed{\mathbb{E}\left[G^t[j,j]\right] \approx (1-\beta_2^t)\mathbb{E}\left[\left(g^i[j]\right)^2\right]}$ $\boxed{\mathbb{E}\left[\tilde{g}^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]}$

Check the update (blue box) element-wise:

$$\mathbb{E}[\Delta W^t[j]] = \mathbb{E}\left[\eta_t \frac{\tilde{g}^t[j]}{\sqrt{\epsilon + G^t[j,j]}}\right] \approx \eta_t \mathbb{E}\left[\frac{\tilde{g}^t[j]}{\sqrt{G^t[j,j]}}\right]$$

Substitute the moment estimates

$$\approx \eta_t \mathbb{E}[\tilde{g}^t[j]]\, \mathbb{E}\left[\frac{1}{\sqrt{G^t[j,j]}}\right] \approx \eta_t \frac{\mathbb{E}[\tilde{g}^t[j]]}{\sqrt{\mathbb{E}[G^t[j,j]]}}$$

$$\boxed{\approx \eta_t \frac{(1-\beta_1^t)\mathbb{E}[g^i[j]]}{\sqrt{1-\beta_2^t}\sqrt{\mathbb{E}[(g^i[j])^2]}}}$$

BP + Adaptive Learning Rate + Momentum (Adam)

Now that we can tune the learning rate adaptively, how can we incorporate momentum?

Based on RMSProp, Adam (adaptive moment estimation) [8] proposes to keep another EMA of past gradients:

$$\tilde{g}^t = \beta_1 \tilde{g}^{t-1} + (1-\beta_1) g^t$$

$$G^t = \beta_2 G^{t-1} + (1-\beta_2) g^t g^{t\top}$$

$$W^t = W^{t-1} - \boxed{\eta_t \mathrm{diag}\left(\epsilon I + G^t\right)^{-1/2} \tilde{g}^t} \qquad \text{denoted as } \Delta W^t$$

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

Where does this bias-correction term (red box) come from?

Similar reasoning applies to G^t. We have $\mathbb{E}\left[G^t[j,j]\right] \approx (1-\beta_2^t)\mathbb{E}\left[\left(g^i[j]\right)^2\right]$ $\mathbb{E}\left[\tilde{g}^t\right] \approx (1-\beta_1^t)\mathbb{E}\left[g^i\right]$

Check the update (blue box) element-wise:

$$\mathbb{E}[\Delta W^t[j]] = \mathbb{E}\left[\eta_t \frac{\tilde{g}^t[j]}{\sqrt{\epsilon + G^t[j,j]}}\right] \approx \eta_t \mathbb{E}\left[\frac{\tilde{g}^t[j]}{\sqrt{G^t[j,j]}}\right]$$

$$\approx \eta_t \mathbb{E}[\tilde{g}^t[j]]\, \mathbb{E}\left[\frac{1}{\sqrt{G^t[j,j]}}\right] \approx \eta_t \frac{\mathbb{E}[\tilde{g}^t[j]]}{\sqrt{\mathbb{E}[G^t[j,j]]}}$$

$$\approx \eta_t \frac{(1-\beta_1^t)\mathbb{E}[g^i[j]]}{\sqrt{1-\beta_2^t}\sqrt{\mathbb{E}[(g^i[j])^2]}} = \eta \frac{\mathbb{E}[g^i[j]]}{\sqrt{\mathbb{E}[(g^i[j])^2]}}$$

Choose the correction factor

$$\boxed{\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}}$$

BP + Adaptive Learning Rate + Momentum (Adam)

In summary, Adam is shown in below

Algorithm 1 Adam

Input: step T, initial learning Rate η, initial W^0, batch size B, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $m^0 = 0$, $v^0 = 0$

For $t = 1, \cdots, T$

  Get mini-batch data (X^t, Y^t)

  $L(W^{t-1}, X^t, Y^t) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$

  $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$

  $m^t = \beta_1 m^{t-1} + (1 - \beta_1) g^t$

  $v^t = \beta_2 v^{t-1} + (1 - \beta_2)(g^t \odot g^t)$

  $W^t = W^{t-1} - \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \frac{m^t}{\sqrt{v^t} + \epsilon}$

Return W^T

BP + Adaptive Learning Rate + Momentum (Adam)

In summary, Adam is shown in below

Algorithm 1 Adam

1: **Input**: step T, initial learning Rate η, initial W^0, batch size B, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $m^0 = 0$, $v^0 = 0$
2: **For** $t = 1, \cdots, T$
3: Get mini-batch data (X^t, Y^t)
4: $L(W^{t-1}, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$
5: $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$
6: $m^t = \beta_1 m^{t-1} + (1-\beta_1) g^t$
7: $v^t = \beta_2 v^{t-1} + (1-\beta_2)(g^t \odot g^t)$
8: $W^t = W^{t-1} - \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \frac{m^t}{\sqrt{v^t}+\epsilon}$
9: **Return** W^T

Adam [8] combines first- and second-moment estimates of the gradient.

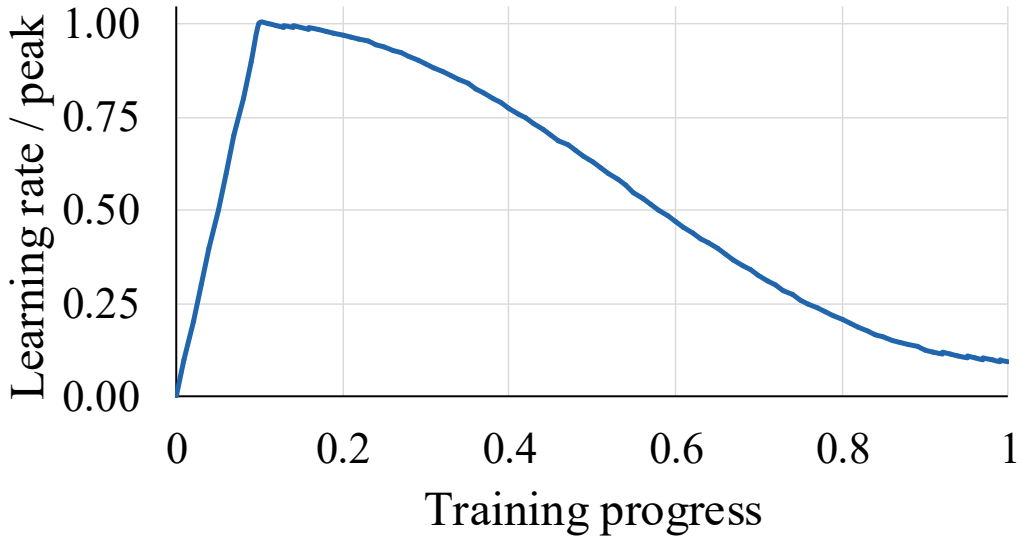
Learning Rate Schedules

Adaptive updates still use a global learning rate.

Warmup: increase the step size gradually at the start [12].

Cosine decay: take progressively smaller steps later [11].

Choose the warmup length and the total training budget.



$$\eta_t = \begin{cases} \eta_{\max}\, t/T_w, & 0 \le t \le T_w, \\ \eta_{\min} + \dfrac{\eta_{\max} - \eta_{\min}}{2}\left[1 + \cos\left(\pi \dfrac{t - T_w}{T - T_w}\right)\right], & T_w < t \le T. \end{cases}$$

Illustrative schedule: 10% warmup and a final rate equal to 10% of the peak. No restarts shown.

Outline

- Learning Algorithm for Feedforward Neural Networks:
  - Backpropagation
  - Weight Initialization
  - Learning Rate, Momentum, Adam & Schedules
  - **Weight Decay**
  - Matrix Optimization: Muon
  - Early Stopping

Weight Decay

Suppose we observe overfitting and want to regularize the complexity of our neural networks to reduce it. We can penalize the weights so that they are not far from 0, thus restricting the set of models we are considering.

Weight Decay

Suppose we observe overfitting and want to regularize the complexity of our neural networks to reduce it. We can penalize the weights so that they are not far from 0, thus restricting the set of models we are considering.

$$L(W, X, Y) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W, X[i]), Y[i])$$

$$\tilde{L}(W, X, Y) = L(W, X, Y) + \frac{\lambda}{2} \|\text{Vec}(W)\|_2^2$$

We can control the lambda to trade-off model complexity and overfitting.

Weight Decay

Suppose we observe overfitting and want to regularize the complexity of our neural networks to reduce it. We can penalize the weights so that they are not far from 0, thus restricting the set of models we are considering.

$$L(W, X, Y) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W, X[i]), Y[i])$$

$$\tilde{L}(W, X, Y) = L(W, X, Y) + \frac{\lambda}{2} \|\text{Vec}(W)\|_2^2$$

We can control the lambda to trade-off model complexity and overfitting.

It only adds a term to the existing gradient:

$$\frac{\partial \tilde{L}}{\partial W} = \frac{\partial L}{\partial W} + \lambda W$$

For methods like SGD, SGD + Momentum, we directly add this term to the gradient and then perform gradient update

Weight Decay

Suppose we observe overfitting and want to regularize the complexity of our neural networks to reduce it. We can penalize the weights so that they are not far from 0, thus restricting the set of models we are considering.

$$L(W, X, Y) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W, X[i]), Y[i])$$

$$\tilde{L}(W, X, Y) = L(W, X, Y) + \frac{\lambda}{2} \|\text{Vec}(W)\|_2^2$$

We can control the lambda to trade-off model complexity and overfitting.

It only adds a term to the existing gradient: $$\frac{\partial \tilde{L}}{\partial W} = \frac{\partial L}{\partial W} + \lambda W$$

For methods like SGD, SGD + Momentum, we directly add this term to the gradient and then perform gradient update

With Adam: 1) include the penalty in both moment estimates; 2) shrink the weights separately.

Option 2 keeps weight decay out of the moment estimates; this is AdamW [9].

AdamW

Let us look at Adam again:

Algorithm 1 Adam

Input: step T, initial learning Rate η, initial W^0, batch size B, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $m^0 = 0$, $v^0 = 0$

For $t = 1, \cdots, T$

  Get mini-batch data (X^t, Y^t)

  $L(W^{t-1}, X^t, Y^t) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$

  $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$

  $m^t = \beta_1 m^{t-1} + (1 - \beta_1) g^t$

  $v^t = \beta_2 v^{t-1} + (1 - \beta_2)(g^t \odot g^t)$

  $W^t = W^{t-1} - \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \frac{m^t}{\sqrt{v^t} + \epsilon}$

Return W^T

We can add it to the gradient in the red box:

$$g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W} + \lambda W^{t-1}$$

But then weight decay also goes through EMA!

AdamW

We can add it in the final gradient update:

Algorithm 1 AdamW

Input: step T, initial learning Rate η, initial W^0, batch size B, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $m^0 = 0$, $v^0 = 0$, weight decay λ

For $t = 1, \cdots, T$

  Get mini-batch data (X^t, Y^t)

  $L(W^{t-1}, X^t, Y^t) = \frac{1}{B} \sum_{i=1}^{B} \ell(f(W^{t-1}, X^t[i]), Y^t[i])$

  $g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$

  $m^t = \beta_1 m^{t-1} + (1 - \beta_1) g^t$

  $v^t = \beta_2 v^{t-1} + (1 - \beta_2)(g^t \odot g^t)$

  $W^t = W^{t-1} - \eta \left(\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \frac{m^t}{\sqrt{v^t} + \epsilon} + \lambda W^{t-1} \right)$

Return W^T

This change works significantly better in some cases [9]!

AdamW: Decoupled Weight Decay

The difference is where the regularization enters [9].

$$g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$$

Adam + L_2 penalty

$$r^t = g^t + \lambda W^{t-1}$$

$$m^t = \beta_1 m^{t-1} + (1-\beta_1) r^t$$

$$v^t = \beta_2 v^{t-1} + (1-\beta_2)(r^t \odot r^t)$$

$$W^t = W^{t-1} - \eta_t \frac{\hat{m}^t}{\sqrt{\hat{v}^t} + \epsilon}$$

AdamW

$$m^t = \beta_1 m^{t-1} + (1-\beta_1) g^t$$

$$v^t = \beta_2 v^{t-1} + (1-\beta_2)(g^t \odot g^t)$$

$$W^t = (1 - \eta_t \lambda) W^{t-1} - \eta_t \frac{\hat{m}^t}{\sqrt{\hat{v}^t} + \epsilon}$$

$$\hat{m}^t = \frac{m^t}{1-\beta_1^t}, \qquad \hat{v}^t = \frac{v^t}{1-\beta_2^t}$$

The penalty changes both moment estimates. AdamW shrinks the weights separately.

Outline

- Learning Algorithm for Feedforward Neural Networks:
  - Backpropagation
  - Weight Initialization
  - Learning Rate, Momentum, Adam & Schedules
  - Weight Decay
  - **Matrix Optimization: Muon**
  - Early Stopping

Muon: Matrix Updates

Muon uses the matrix structure of a layer's update [13].

Here, Muon applies only to hidden linear-layer weight matrices.

$$W^{t-1},\ g^t,\ m^t \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \qquad g^t = \frac{\partial L(W^{t-1}, X^t, Y^t)}{\partial W}$$

$$m^t = \beta m^{t-1} + (1-\beta) g^t, \qquad m^0 = 0$$

$$\text{Momentum:} \quad W^t = W^{t-1} - \eta_t\, m^t$$

$$\text{Muon:} \quad W^t = W^{t-1} - \eta_t\ \text{Orth}(m^t)$$

These matrices map input to output features, giving meaningful singular directions.

What does it mean to orthogonalize a momentum matrix?

Here g^t denotes the matrix gradient. Adam rescales individual entries; Muon transforms the whole matrix.

Muon: Singular Value Decomposition

Separate a matrix's directions from its stretching.

$$m^t = U\Sigma V^\top, \qquad \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_q)$$

Read the product from right to left:

$V^\top$: input coordinates

Σ : stretch each direction

U : output directions

$$q = \min(d_{\text{out}}, d_{\text{in}}), \quad U \in \mathbb{R}^{d_{\text{out}} \times q}, \quad V \in \mathbb{R}^{d_{\text{in}} \times q}$$

Animation: Lucas Vieira (2010), public domain [14].

For the next example, assume the matrix has full rank.

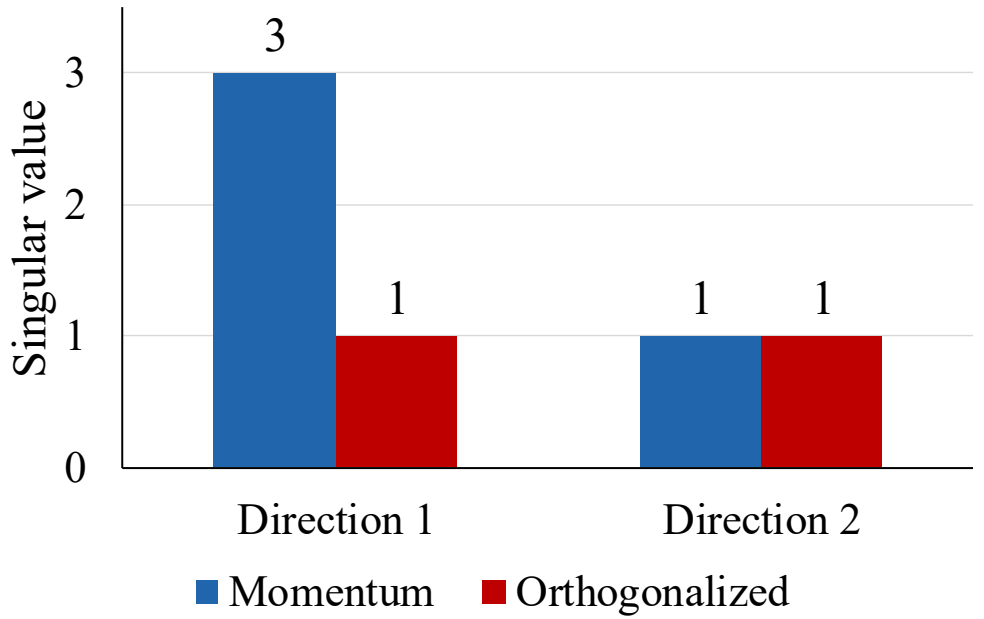
Muon: A Small Matrix Example

Replace unequal singular values by a common scale [13].

$$m^t = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

$$U = V = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$Q^t = UV^\top = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$W^t = W^{t-1} - \eta_t Q^t$$

Orthogonalization acts on the update matrix, not on the weights.

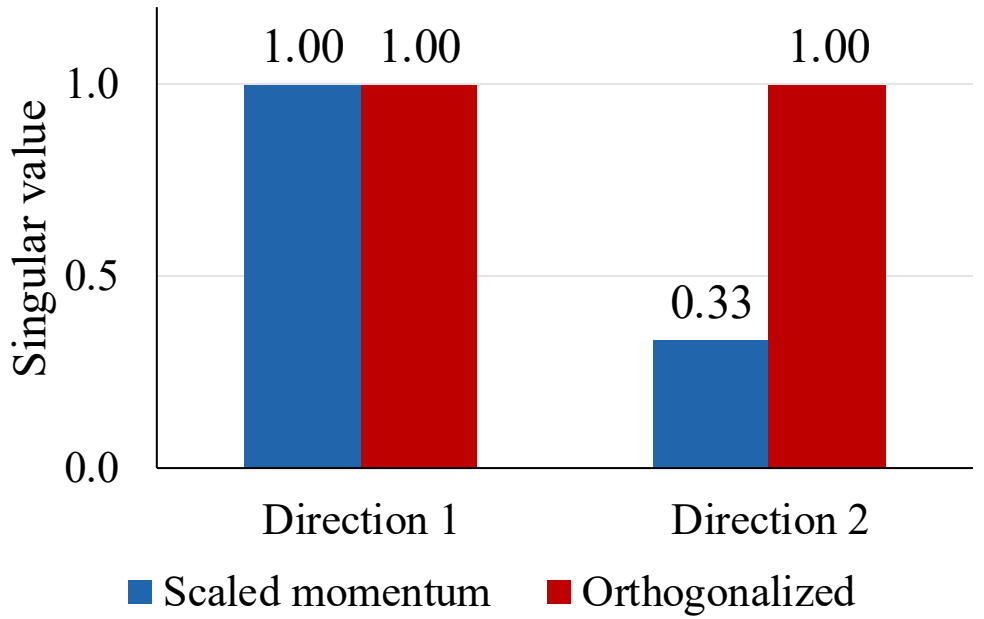
Muon: Motivation for Orthogonalization

Balance the update across its singular directions [13, 16].

A few strong modes can dominate the update. A single scale keeps this imbalance.

$$\frac{m^t}{\sigma_{\max}(m^t)} = \frac{m^t}{3}$$

Compare updates with the same largest singular value: one.



Useful weak modes can contribute more. Noisy modes may also gain importance.

Orthogonalization preserves the singular-vector pairs. The weights themselves need not be orthogonal.

Muon: Efficient Orthogonalization

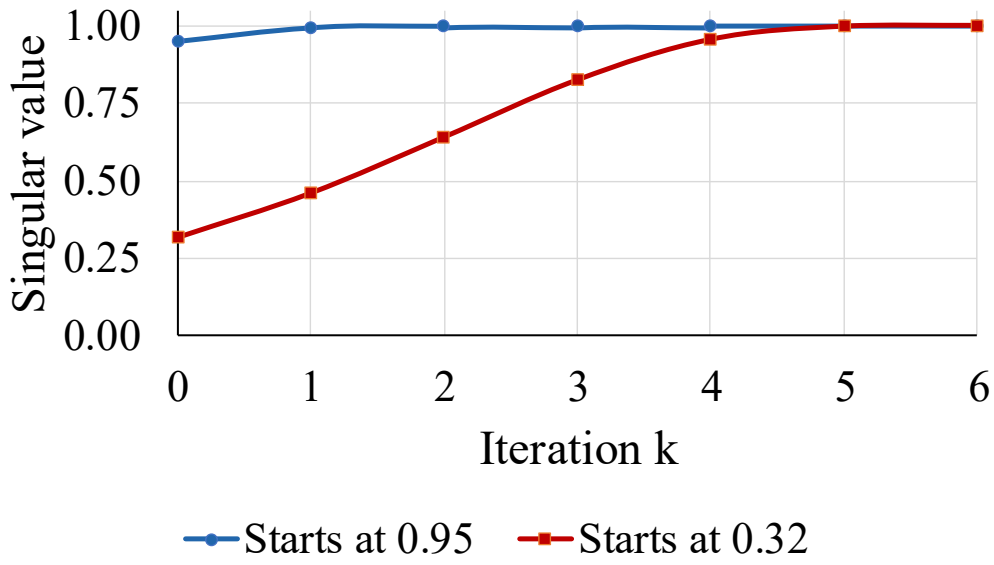
Use matrix products instead of a full SVD at every step [13, 16].

Classical cubic Newton–Schulz iteration:

$$X_0 = \frac{m^t}{\|m^t\|_F} \quad (m^t \neq 0)$$

$$X_{k+1} = \frac{3}{2}X_k - \frac{1}{2}(X_k X_k^\top)X_k$$

$$s_{k+1} = \frac{3}{2}s_k - \frac{1}{2}s_k^3$$



The iteration changes singular values without computing singular vectors.

A few iterations give an inexpensive approximation on GPUs.

The plot uses the 2 × 2 example. Small singular values need more iterations.

Muon: Newton–Schulz Convergence

The classical cubic converges to the polar factor [16].

Normalize first, so each singular value starts in [0, 1]. Then

$$s_{i,k+1} = f(s_{i,k}), \qquad f(s) = \frac{3}{2}s - \frac{1}{2}s^3$$

$$f(s) - s = \frac{1}{2}s(1 - s^2) \geq 0, \qquad 1 - f(s) = \frac{1}{2}(1 - s)^2(s + 2) \geq 0$$

Positive singular values increase to 1. Zero values stay zero.

Using the compact SVD for a matrix of rank r_0,

$$X_k = U_{r_0} \operatorname{diag}(s_{i,k}) V_{r_0}^\top \longrightarrow U_{r_0} V_{r_0}^\top = Q$$

Return the zero matrix when the momentum is zero.

Practical Muon uses a few tuned quintic steps. Exact convergence is not required [13].

The inner iteration k orthogonalizes a fixed momentum. The index t counts training steps.

Muon: The Learning Algorithm

Combine momentum, orthogonalization, and a parameter update [13, 15].

Algorithm 1 Muon for one hidden weight matrix (simplified)

Input: W^0, $m^0 = 0$, momentum β, learning rates η_t, decay λ
for $t = 1, \ldots, T$ **do**
    Get mini-batch (X^t, Y^t) and compute $g^t = \partial L(W^{t-1}, X^t, Y^t)/\partial W$
    $m^t = \beta m^{t-1} + (1 - \beta) g^t$
    $Q^t \approx \mathrm{Orth}(m^t)$ via matrix-polynomial steps (use $Q^t = 0$ if $m^t = 0$)
    $W^t = (1 - \eta_t \lambda) W^{t-1} - \eta_t Q^t$
end for
Return W^T

Here, Muon: hidden linear-layer weight matrices only.

AdamW: embeddings, output head, biases, and normalization parameters.

The simplified form uses ordinary momentum. Practical variants add Nesterov momentum and shape scaling.

Muon: Results and Extensions

Promising LLM results depend on the training setup [15].

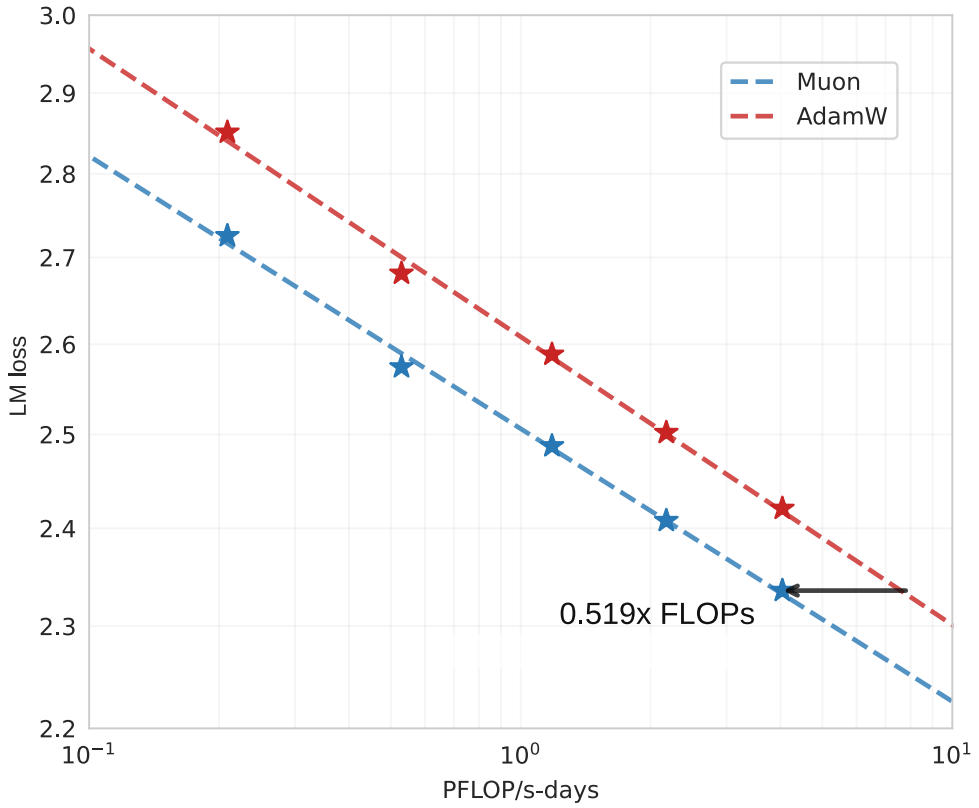


Figure: Liu et al. (2025), Fig. 1a [15].

About 52% of AdamW's training FLOPs

At matched loss in fitted scaling curves for dense language models with 399M–1.5B non-embedding parameters.

Polar Express improves finite-step polynomial approximations [16].

MuonClip adds clipping to stabilize attention scores in large Transformers [17].

FLOP savings do not directly specify wall-clock savings or guarantee gains on a new task.

Outline

- Learning Algorithm for Feedforward Neural Networks:
  - Backpropagation
  - Weight Initialization
  - Learning Rate, Momentum, Adam & Schedules
  - Weight Decay
  - Matrix Optimization: Muon
  - **Early Stopping**

Early Stopping

We do not necessarily need to run the optimization until the maximum number of iterations or until its convergence.

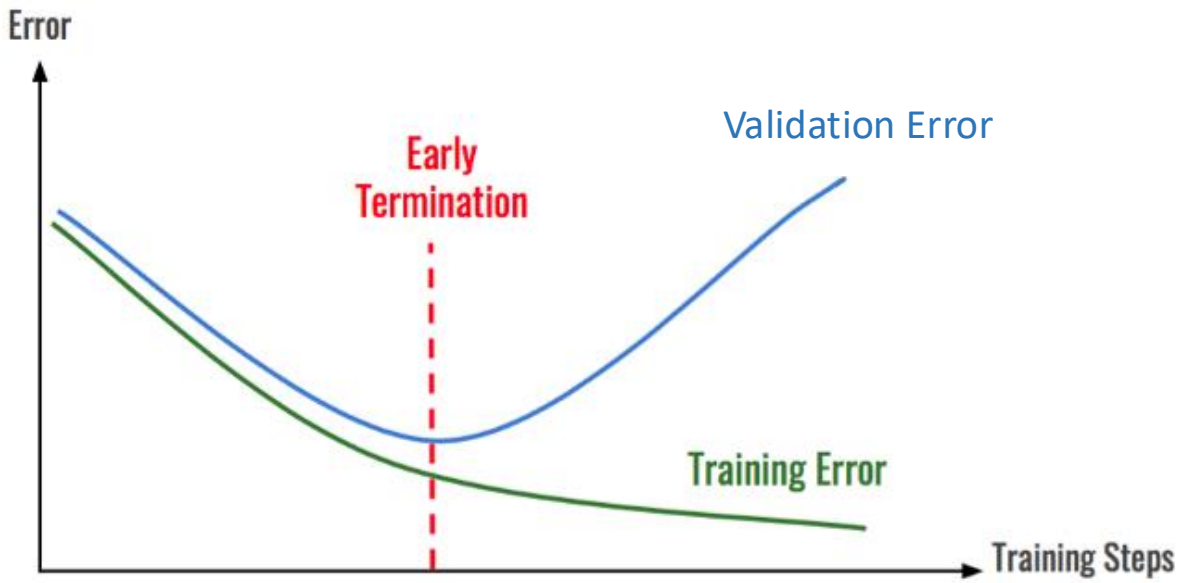
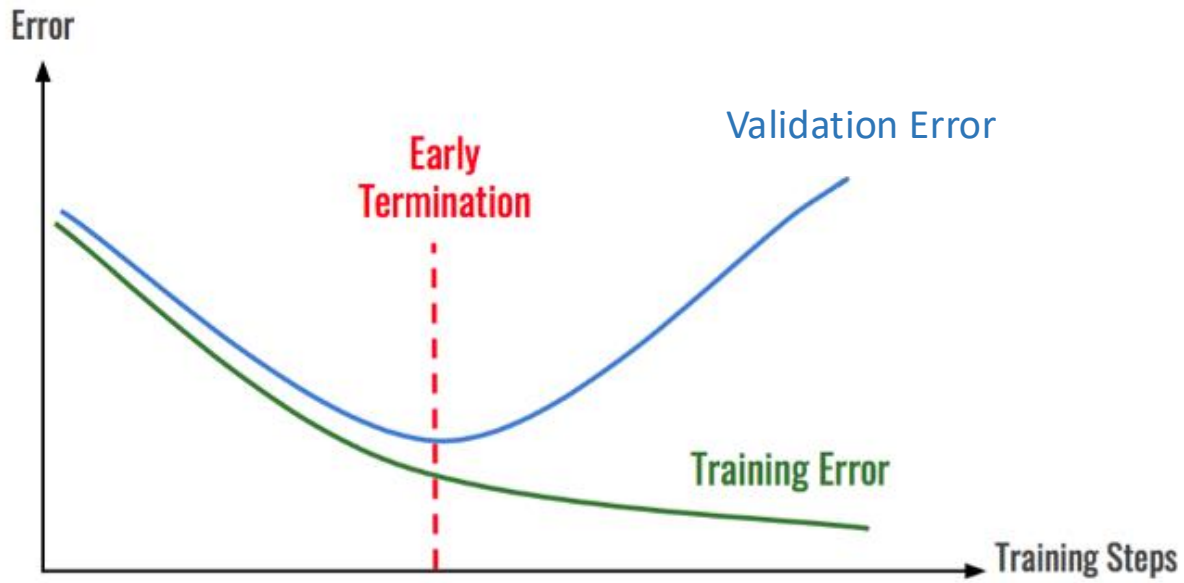


Image Credit: [10]

Early Stopping

We do not necessarily need to run the optimization until the maximum number of iterations or until its convergence.



Stop after a chosen number of validation checks without meaningful improvement, then restore the best checkpoint.

Alternatively, train for the full budget and select the checkpoint with the lowest validation loss.

Image Credit: [10]

Early Stopping: A Practical Rule

Use validation loss to decide when to stop [18].

Checkpoint: save the weights whenever validation loss reaches a new minimum.

Patience: reset the counter after a significant improvement. Otherwise add 1.

$$L_{\text{val}}(W^t) < b - \delta$$

Here b is the loss at the last significant improvement; δ is the minimum decrease.

Stop after p checks without significant improvement, then restore the best checkpoint.

Example: p = 3, δ = 0.01. Losses 0.500, 0.460, 0.455, 0.456, 0.457.

Stop at check 5 and restore the weights from check 3.

Patience counts validation checks. Choose the rule using validation data; keep the test set for final evaluation.

References

[1] https://www.ruder.io/optimizing-gradient-descent/

[2] Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. In Proc. of Eighth Annual Conference of the Cognitive Science Society (pp. 823-831).

[3] https://distill.pub/2017/momentum/

[4] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, May). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147). PMLR.

[5] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[6] Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report.

[7] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[8] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[9] Loshchilov, I., & Hutter, F. (2017). Fixing weight decay regularization in adam.

[10] https://medium.com/analytics-vidhya/early-stopping-with-pytorch-to-restrain-your-model-from-overfitting-dce6de4081c5

References

[11] Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. ICLR.

[12] Goyal, P., Dollár, P., Girshick, R., et al. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677.

[13] Jordan, K., Jin, Y., Boza, V., et al. (2024). Muon: An optimizer for hidden layers in neural networks.

[14] Vieira, L. (2010). Singular value decomposition [Animation]. Wikimedia Commons. Public domain.

[15] Liu, J., Su, J., Yao, X., et al. (2025). Muon is Scalable for LLM Training. arXiv:2502.16982.

[16] Amsel, N., Persson, D., Musco, C., & Gower, R. (2025). The Polar Express: Optimal Matrix Sign Methods and Their Application to the Muon Algorithm. arXiv:2505.16932.

[17] Kimi Team (2025). Kimi K2: Open Agentic Intelligence. arXiv:2507.20534.

[18] Keras Team. EarlyStopping. Keras API documentation.

Questions?