Optimization

CS4782: Intro to Deep Learning
Quiz on Canvas!
(code-1234)
Agenda

- Backpropagation
- Optimizers
  - Gradient Descent
  - Stochastic Gradient Descent
  - SGD w. Momentum
  - AdaGrad
  - RMSProp
  - Adam
- Learning rate scheduling
- Hyperparameter Optimization
Recap

MLPs can learn complex decision boundaries!
Forward Pass - MLP

\[ a^{[1]} = W^{[1]} z^{[0]} + b^{[1]} \]

\[ z^{[0]} = x \]
Discuss

What are the dimensions of $W^{[1]}$ and $b^{[1]}$?

$x$: (2, 1)

$W^{[1]}$:

$b^{[1]}$:

$a^{[1]}$:
Discuss

What are the dimensions of $W^{[1]}$ and $b^{[1]}$?

$x$: (2, 1)

$W^{[1]}$: (3, 2)

$b^{[1]}$: (3, 1)

$a^{[1]}$: (3, 1)

$a^{[1]} = W^{[1]}z^{[0]} + b^{[1]}$

$z^{[0]} = x$
Forward Pass - MLP

\[ a^{[1]} = W^{[1]} z^{[0]} + b^{[1]} \]

\[ z^{[0]} = x \quad z^{[1]} = \sigma(a^{[1]}) \]
Forward Pass - MLP

\[ a^{[1]} = W^{[1]}z^{[0]} + b^{[1]} \quad a^{[2]} = W^{[2]}z^{[1]} + b^{[2]} \]

\[ z^{[0]} = x \quad z^{[1]} = \sigma(a^{[1]}) \quad z^{[2]} = \sigma(a^{[2]}) \quad z^{[3]} = \sigma(a^{[3]}) \]

\[ a^{[3]} = W^{[3]}z^{[2]} + b^{[3]} \]
**Algorithm Forward Pass through MLP**

1. **Input:** input $x$, weight matrices $W^{[1]}, \ldots, W^{[L]}$, bias vectors $b^{[1]}, \ldots, b^{[L]}$
2. $z^{[0]} = x$ \text{ ▶ Initialize input}
3. for $l = 1$ to $L$ do
4. \hspace{1em} $a^{[l]} = W^{[l]}z^{[l-1]} + b^{[l]}$ \text{ ▶ Linear transformation}
5. \hspace{1em} $z^{[l]} = \sigma^{[l]}(a^{[l]})$ \text{ ▶ Nonlinear activation}
6. end for
7. **Output:** $z^{[L]}$

\[ a^{[1]} = W^{[1]}z^{[0]} + b^{[1]} \quad a^{[2]} = W^{[2]}z^{[1]} + b^{[2]} \]
\[ a^{[3]} = W^{[3]}z^{[2]} + b^{[3]} \]

$z^{[0]} = x \quad z^{[1]} = \sigma(a^{[1]}) \quad z^{[2]} = \sigma(a^{[2]}) \quad z^{[3]} = \sigma(a^{[3]})$
Backpropagation - MLPs

\[ a^{[1]} = W^{[1]} z^{[0]} + b^{[1]} \]
\[ a^{[2]} = W^{[2]} z^{[1]} + b^{[2]} \]
\[ a^{[3]} = W^{[3]} z^{[2]} + b^{[3]} \]

\[ z^{[0]} = x \quad z^{[1]} = \sigma(a^{[1]}) \quad z^{[2]} = \sigma(a^{[2]}) \quad z^{[3]} = \sigma(a^{[3]}) \]

Loss = \mathcal{L}

We can directly compute \( \frac{\partial \mathcal{L}}{\partial z^{[3]}} \)!
Backpropagation - MLPs

\[ a^{[1]} = W^{[1]}z^{[0]} + b^{[1]} \quad a^{[2]} = W^{[2]}z^{[1]} + b^{[2]} \]

\[ a^{[3]} = W^{[3]}z^{[2]} + b^{[3]} \]

\[ \delta^{[3]} = \frac{\partial L}{\partial a^{[3]}} = \frac{\partial L}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[3]}} = \frac{\partial L}{\partial z^{[3]}} \odot \sigma^{[3]'} \]

\[ z^{[0]} = x \quad z^{[1]} = \sigma(a^{[1]}) \quad z^{[2]} = \sigma(a^{[2]}) \quad z^{[3]} = \sigma(a^{[3]}) \]
Backpropagation - MLPs

\[ a^{[1]} = W^{[1]}z^{[0]} + b^{[1]} \quad a^{[2]} = W^{[2]}z^{[1]} + b^{[2]} \]

\[ a^{[3]} = W^{[3]}z^{[2]} + b^{[3]} \]

\[ z^{[0]} = x \quad z^{[1]} = \sigma(a^{[1]}) \quad z^{[2]} = \sigma(a^{[2]}) \quad z^{[3]} = \sigma(a^{[3]}) \]

\[ \frac{\partial L}{\partial W^{[3]}} = \frac{\partial L}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial W^{[3]}} \]

Simplify further!
Backpropagation - MLPs

\[
\begin{align*}
a^{[1]} &= W^{[1]}z^{[0]} + b^{[1]} \\
a^{[2]} &= W^{[2]}z^{[1]} + b^{[2]} \\
a^{[3]} &= W^{[3]}z^{[2]} + b^{[3]} \\
\end{align*}
\]

\[
\begin{align*}
z^{[0]} &= x \\
z^{[1]} &= \sigma(a^{[1]}) \\
z^{[2]} &= \sigma(a^{[2]}) \\
z^{[3]} &= \sigma(a^{[3]}) \\
\end{align*}
\]

\[
\begin{align*}
\frac{\partial L}{\partial W^{[3]}} &= \frac{\partial L}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial W^{[3]}} \\
&= \delta^{[3]} (z^{[2]})^T \\
\frac{\partial L}{\partial b^{[3]}} &= \frac{\partial L}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial b^{[3]}} \\
&= \delta^{[3]} 
\end{align*}
\]
Backpropagation - MLPs

\[ a^{[1]} = W^{[1]}z^{[0]} + b^{[1]} \]
\[ a^{[2]} = W^{[2]}z^{[1]} + b^{[2]} \]
\[ a^{[3]} = W^{[3]}z^{[2]} + b^{[3]} \]

\[ z^{[0]} = x \]
\[ z^{[1]} = \sigma(a^{[1]}) \]
\[ z^{[2]} = \sigma(a^{[2]}) \]
\[ z^{[3]} = \sigma(a^{[3]}) \]
Backpropagation - MLPs

Algorithm Backward Pass through MLP (Detailed)

1: **Input:** \{\(z^{[1]}, \ldots, z^{[L]}\)}, \{\(a^{[1]}, \ldots, a^{[L]}\)}, loss gradient \(\frac{\partial L}{\partial z^{[L]}}\)
2: \(\delta^{[L]} = \frac{\partial L}{\partial a^{[L]}} = \frac{\partial L}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial a^{[L]}} = \frac{\partial L}{\partial z^{[L]}} \odot \sigma^{[L]}'(a^{[L]})\)  \(\triangleright\) Error term
3: **for** \(l = L\) **to 1** **do**
4: \(\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial W^{[l]}} = \delta^{[l]} (z^{[l-1]})^T\)  \(\triangleright\) Gradient of weights
5: \(\frac{\partial L}{\partial b^{[l]}} = \frac{\partial L}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial b^{[l]}} = \delta^{[l]}\)  \(\triangleright\) Gradient of biases
6: \(\frac{\partial L}{\partial z^{[l-1]}} = \frac{\partial L}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial z^{[l-1]}} = (W^{[l]})^T \delta^{[l]}\)
7: \(\delta^{[l-1]} = \frac{\partial L}{\partial a^{[l-1]}} = \frac{\partial L}{\partial z^{[l-1]}} \frac{\partial z^{[l-1]}}{\partial a^{[l-1]}} = ((W^{[l]})^T \delta^{[l]}) \odot \sigma^{[l-1]}'(a^{[l-1]})\)
8: **end for**
9: **Output:** \(\frac{\partial L}{\partial W^{[1:L]}}, \frac{\partial L}{\partial b^{[1:L]}}\)
Algorithm Backward Pass through MLP

1: **Input:** \{\(z^{[1]}, \ldots, z^{[L]}\), \(a^{[1]}, \ldots, a^{[L]}\}\}, loss gradient \(\frac{\partial L}{\partial z^{[L]}}\)
2: \(\delta^{[L]} = \frac{\partial L}{\partial z^{[L]}} \odot \sigma^{[L]'}(a^{[L]})\) \hspace{1cm} \triangleright \text{Error term}
3: **for** \(l = L\) **to** 1 **do**
4: \(\frac{\partial L}{\partial w^{[l]}} = \delta^{[l]} (z^{[l-1]})^T\) \hspace{1cm} \triangleright \text{Gradient of weights}
5: \(\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]}\) \hspace{1cm} \triangleright \text{Gradient of biases}
6: \(\delta^{[l-1]} = ((w^{[l]})^T \delta^{[l]}) \odot \sigma^{[l-1]'}(a^{[l-1]})\)
7: **end for**
8: **Output:** \(\frac{\partial L}{\partial w^{[1:L]}}, \frac{\partial L}{\partial b^{[1:L]}}\)
Discuss: Activation functions

- How do different activation functions behave during backprop?
  - Visualize their derivatives!
What is Optimization?

In deep learning, optimization methods attempt to find model weights that minimize the loss function.
Loss function

Empirical Risk:

\[ \mathcal{L}(w_t) = \frac{1}{n} \sum_{i=1,\ldots,n} \ell(w_t, x_i) \]

- \( t \): at time step \( t \)
- \( w_t \): Model weights (parameters) at time \( t \)
- \( x_i \): The \( i \)-th input training data

\( \mathcal{L} \): the Loss function (optimization target)
\( \ell \): per-sample loss
Gradient Descent (GD)

\[ w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t) \]

\( \alpha \): the learning rate
\( \nabla \mathcal{L}(w_t) \): the gradient of Loss w.r.t. \( w_t \)
What are some potential problems with gradient descent?
Convexity

- A function on a graph is **convex** if a line segment drawn through any two points on the line of the function, then it never lies below the curved line segment.
- Convexity implies that every local minimum is **global minimum**.
- Neural networks are **not** convex!
Challenges in Non-Convex Optimization

Local Minima vs. Global Minima

Saddle Points

Vanishing gradient
Gradient Descent (GD)

\[ \mathcal{L}(w_t) = \frac{1}{n} \sum_{i=1}^{n} \ell(w_t, x_i) \]

\[ \nabla \mathcal{L}(w_t) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(w_t, x_i) \]

\[ w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t) \]

Full gradient: \( O(n) \) time => \textbf{Too expensive!}

- Statistically, why don’t we use 1 or a few samples from the training dataset to approximate the full gradient?
Gradient Descent (GD)

\[ w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t) \]
Gradient Descent (GD)

\[ \mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t) \]

Select 1 example randomly each time
Gradient Descent (GD)

\[ \mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t) \]

Select 1 example randomly each time

Per-sample gradient is equivalent to full gradient in expectation!

\[ \mathbb{E}[\nabla \ell(\mathbf{w}_t, \mathbf{x}_i)] = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i) = \nabla \mathcal{L}(\mathbf{w}_t) \]
Stochastic Gradient Descent (SGD)

$$w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t)$$

Select 1 example randomly each time

$$w_{t+1} = w_t - \alpha \nabla \ell(w_t, x_i)$$

Per-sample gradient is equivalent to full gradient in expectation!

$$\mathbb{E}[\nabla \ell(w_t, x_i)] = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(w_t, x_i) = \nabla \mathcal{L}(w_t)$$
Stochastic Gradient Descent (SGD)

\[ w_{t+1} = w_t - \alpha \nabla L(w_t) \]

Select 1 example randomly each time

\[ w_{t+1} = w_t - \alpha \nabla \ell(w_t, x_i) \]

Trade off convergence!

Per-sample gradients not necessarily points to the local minimum, introducing a noise ball...
Stochastic Gradient Descent (SGD)

\[ w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t) \]

Select 1 example randomly each time

\[ w_{t+1} = w_t - \alpha \nabla \ell(w_t, x_i) \]

Select a batch \( B_t \) of examples randomly each time, with batch size \( b \)
Cornell Bowers CIS

Minibatch SGD

\[ w_{t+1} = w_t - \alpha \nabla L(w_t) \]

Select 1 example randomly each time

\[ w_{t+1} = w_t - \alpha \nabla \ell(w_t, x_i) \]

Select a batch \( B_t \) of examples randomly each time, with batch size \( b \)

\[ w_{t+1} = w_t - \alpha \cdot \frac{1}{b} \sum_{i \in B_t} \nabla \ell(w_t, x_i) \]
Minibatch SGD

Best of both worlds: Computational and Statistical!

\[ w_{t+1} = w_t - \alpha \cdot \frac{1}{b} \sum_{i \in B_t} \nabla \ell(w_t, x_i) \]
Let's look at an example!
Local Minimum
Minibatch SGD
SGD with Momentum (Polyak, 1964)

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.
Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.
SGD with Momentum (Polyak, 1964)

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

**SGD Update Rule**

\[
\begin{align*}
w_{t+1} &= w_t - \alpha \nabla \ell(w_t, x_i) \\
m_{t+1} &= \mu m_t - \alpha \nabla \ell(w_t; x_i) \\
w_{t+1} &= w_t + m_{t+1}
\end{align*}
\]

where \( \mu \in [0, 1] \) is the momentum coefficient.
SGD with Momentum (Polyak, 1964)

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

\[
\begin{align*}
\mathbf{m}_{t+1} &= \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \mathbf{m}_{t+1}
\end{align*}
\]

where \( \mu \in [0, 1] \) is the momentum coefficient.
SGD with Momentum

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

\[
\begin{align*}
    m_{t+1} &= \mu m_t - \alpha \nabla l(w_t; x_i) \\
    w_{t+1} &= w_t + m_{t+1}
\end{align*}
\]
SGD with Momentum

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

\[
g_t = \nabla l(w_t; x_i)
\]
\[
m_{t+1} = \mu m_t - \alpha g_t
\]
\[
w_{t+1} = w_t + m_{t+1}
\]
SGD with Momentum

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

\[
g_t = \nabla l(w_t; x_i)
\]

\[
m_{t+1} = \mu m_t - \alpha g_t
\]

\[
w_{t+1} = w_t + \mu m_t - \alpha g_t
\]
SGD with Momentum

Compute an **Exponentially Weighted Moving Average (EWMA)** of the gradients as momentum and use that to update the weight instead.

\[
\begin{align*}
g_t &= \nabla l(w_t; x_i) \\
m_{t+1} &= \mu m_t - \alpha g_t \\
w_{t+1} &= w_t + \mu m_t - \alpha g_t \\
    &= w_t + \mu (\mu m_{t-1} - \alpha g_{t-1}) - \alpha g_t
\end{align*}
\]
SGD with Momentum

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

\[
g_t = \nabla l(w_t; x_i)
\]

\[
m_{t+1} = \mu m_t - \alpha g_t
\]

\[
w_{t+1} = w_t + \mu m_t - \alpha g_t
\]

\[
= w_t + \mu (\mu m_{t-1} - \alpha g_{t-1}) - \alpha g_t
\]

\[
= w_t + \mu (\mu (\mu m_{t-2} - \alpha g_{t-2}) - \alpha g_{t-1}) - \alpha g_t
\]

\[
= w_t - \alpha g_t - \mu \alpha g_{t-1} - \mu^2 \alpha g_{t-2} - \mu^3 \alpha g_{t-3} - \ldots
\]
SGD with Momentum

Compute an *Exponentially Weighted Moving Average (EWMA)* of the gradients as *momentum* and use that to update the weight instead.

\[
g_t = \nabla l(w_t; x_i)
\]

\[
m_{t+1} = \mu m_t - \alpha g_t
\]

\[
w_{t+1} = w_t + \mu m_t - \alpha g_t
\]

\[
= w_t + \mu (\mu m_{t-1} - \alpha g_{t-1}) - \alpha g_t
\]

\[
= w_t + \mu (\mu m_{t-2} - \alpha g_{t-2}) - \alpha g_{t-1} - \alpha g_t
\]

\[
= w_t - \alpha g_t - \mu \alpha g_{t-1} - \mu^2 \alpha g_{t-2} - \mu^3 \alpha g_{t-3} - \ldots
\]

\[
= w_t - \alpha \sum_{i=0}^{t} \mu^i g_{t-i}
\]
Local Minimum
Minibatch SGD
Local Minimum Minibatch SGD

\[ g_t = \nabla l(w_t; x_i) \]
\[ m_{t+1} = \mu m_t - \alpha g_t \]
\[ w_{t+1} = w_t + m_{t+1} \]
\[ m_2 = \mu m_1 - \alpha g_1 \]

\[ \nabla l(w_t; x_i) = g_t \]

\[ m_{t+1} = \mu m_t - \alpha g_t \]

\[ w_{t+1} = w_t + m_{t+1} \]
\[ m_2 = \mu m_1 - \alpha g_1 \]

Local Minimum
Minibatch SGD
SGD w. Momentum

\[ g_t = \nabla l(w_t; x_i) \]
\[ m_{t+1} = \mu m_t - \alpha g_t \]
\[ w_{t+1} = w_t + m_{t+1} \]
Local Minimum
SGD w. Momentum
Momentum converges almost always faster than standard SGD!
Momentum converges almost always faster than standard SGD!
Quick Recap

**Gradient Descent**

\[ w_{t+1} = w_t - \alpha \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(w_t, x_i) \]

**Stochastic Gradient Descent**

\[ w_{t+1} = w_t - \alpha \nabla \ell(w_t, x_i) \]

**Minibatch SGD**

\[ w_{t+1} = w_t - \alpha \cdot \frac{1}{b} \sum_{i \in B_t} \nabla \ell(w_t, x_i) \]

**SGD w. Momentum**

\[ m_{t+1} = \mu m_t - \alpha \nabla l(w_t; x_i) \]
\[ w_{t+1} = w_t + m_{t+1} \]
Importance of Learning Rate

- SGD (learning_rate=0.01)
  step: 0: (-6.63, 43.953)

- SGD (learning_rate=0.1)
  step: 0: (-6.63, 43.953)

- SGD (learning_rate=0.95)
  step: 0: (-6.63, 43.953)

- SGD (learning_rate=1.01)
  step: 0: (-6.63, 43.953)
Another example

\[ \alpha = 0.05 \]

\[ \alpha = 0.015 \]

\[ \alpha_x = 0.015 \text{ and } \alpha_y = 0.05 \]
Adaptive Learning Rate

Maybe we don’t want the SAME learning rate for ALL ELEMENTS of the Weight!
Adaptive Optimizers

Different Learning Rate for each element of the Model Weights!
AdaGrad (Duchi et al. 2011)

More updates \(\rightarrow\) more decay

- Handle sparse gradients well
  - Sparse: The vector has 0 in most of the entries

\[
\begin{align*}
    \mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t) \\
    \mathbf{v}_{t+1} &= \mathbf{v}_t + g^2_t \\
    \mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot g_t
\end{align*}
\]

SGD

Adagrad
AdaGrad (Duchi et al. 2011)

More updates → more decay

- Handle sparse gradients well
  - Sparse: The vector has 0 in most of the entries

\[ w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t) \]

GD

\[ v_{t+1} = v_t + g_t^2 \]
\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1} + \epsilon}} \odot g_t \]

Adagrad

Exercise:
What's could be wrong with this optimizator? (What would happen to the denominator.)
AdaGrad (Duchi et al. 2011)

More updates → more decay

- Handle sparse gradients well
  - Sparse: The vector has 0 in most of the entries

\[
\begin{align*}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t) \\
\mathbf{v}_{t+1} &= \mathbf{v}_t + g^2_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot g_t
\end{align*}
\]

Issue: decays too aggressively!
RMSProp (Graves, 2013)

Keep an exponential moving average of the squared gradient for each element

\[
\begin{align*}
    v_{t+1} &= v_t + g_t^2 \\
    w_{t+1} &= w_t - \frac{\alpha}{\sqrt{v_{t+1} + \epsilon}} \odot g_t
\end{align*}
\]

Adagrad

\[
\begin{align*}
    v_{t+1} &= \beta v_t + (1 - \beta) g_t^2 \\
    w_{t+1} &= w_t - \frac{\alpha}{\sqrt{v_{t+1} + \epsilon}} \odot g_t
\end{align*}
\]

RmsProp

where $\beta \in [0, 1]$ the exponential moving average constant.
\[ m_{t+1} = \mu m_t - \alpha \nabla l(w_t; x_i) \]
\[ w_{t+1} = w_t + m_{t+1} \]

**Momentum**

\[ v_{t+1} = \beta v_t + (1 - \beta) g_t^2 \]
\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1} + \epsilon}} \odot g_t \]

**RMSProp**
**Momentum**

\[ m_{t+1} = \mu m_t - \alpha \nabla l(w_t; x_i) \]

\[ w_{t+1} = w_t + m_{t+1} \]

**RMSProp**

\[ v_{t+1} = \beta v_t + (1 - \beta) g_t^2 \]

\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \odot g_t \]
RMSProp

\[ v_{t+1} = \beta v_t + (1 - \beta) g_t^2 \]
\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \odot g_t \]

ADAM (Adaptive Moment Estimate)

\[ v_{t+1} = \beta_2 v_t + (1 - \beta_2) g_t^2 \]
\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_{t+1}} + \epsilon} \odot \hat{m}_{t+1} \]
RMSProp

\[ m_{t+1} = \mu m_t - \alpha \nabla l(w_t; x_i) \]
\[ w_{t+1} = w_t + m_{t+1} \]

Momentum

\[ v_{t+1} = \beta v_t + (1 - \beta) g_t^2 \]
\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \odot g_t \]

RMSProp

ADAM (Adaptive Moment Estimate)

\[ m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_t \]
\[ v_{t+1} = \beta_2 v_t + (1 - \beta_2) g_t^2 \]
\[ w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \odot \hat{m}_{t+1} \]
\[ \begin{align*}
\mathbf{m}_{t+1} & = \beta_1 \mathbf{m}_t + (1 - \beta_1) \mathbf{g}_t \\
\mathbf{v}_{t+1} & = \beta_2 \mathbf{v}_t + (1 - \beta_2) \mathbf{g}_t^2 \\
\hat{\mathbf{m}}_{t+1} & = \frac{\mathbf{m}_{t+1}}{1 - \beta_1^{t+1}} \\
\hat{\mathbf{v}}_{t+1} & = \frac{\mathbf{v}_{t+1}}{1 - \beta_2^{t+1}} \\
\mathbf{w}_{t+1} & = \mathbf{w}_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_{t+1} + \epsilon}} \odot \hat{\mathbf{m}}_{t+1}
\end{align*} \]

**ADAM**

(Adaptive Moment Estimate)
Optimizers Recap

- **Gradient Descent**
  - Vanilla, costly, but for best convergence rate
- **Stochastic Gradient Descent**
  - Simple, lightweight
- **Mini-batch SGD**
  - balanced between SGD and GD
  - 1st choice for small, simple models
- **SGD w. Momentum**
  - Faster, capable to jump out local minimum
- **AdaGrad**
- **RMSProp**
- **ADAM**
  - JUST USE ADAM IF YOU DON’T KNOW WHAT TO USE IN DEEP LEARNING
But are they equivalent somehow?

No!

There are *many* minimizers of the training loss. The *optimizer* determines which minimizer you converge to.
Learning Rate Scheduling
OPT: Open Pre-trained Transformer Language Models

OPT is an open source LLM like GPT-4 from Meta.

For large models like OPT-175B, more engineering efforts are needed.

Figure 1: **Empirical LR schedule.** We found that lowering learning rate was helpful for avoiding instabilities.
Hyperparameters

- Learning rate
- Batch size
- Beta1 & beta2 of adam
- Regularization strength

These are all hyperparameters that affect performance!

Hyperparameter Optimization (HPO)

- Learning rate
- Batch size
- Beta1 & beta2 of adam
- Regularization strength

These are all hyperparameters that affect performance!

**Random search HPO is the efficient and simple way to start!**
Summary

- **Optimization** tries to obtain the model weights that **minimize the loss function**.
- **Adam** is often a good default optimizer in deep learning
- The learning rate usually needs to be tuned carefully
- A monotonically **decreasing learning rate scheduler** with a **warmup** is a good default choice
- **Random search** HPO is the **efficient** and **simple** way to start!