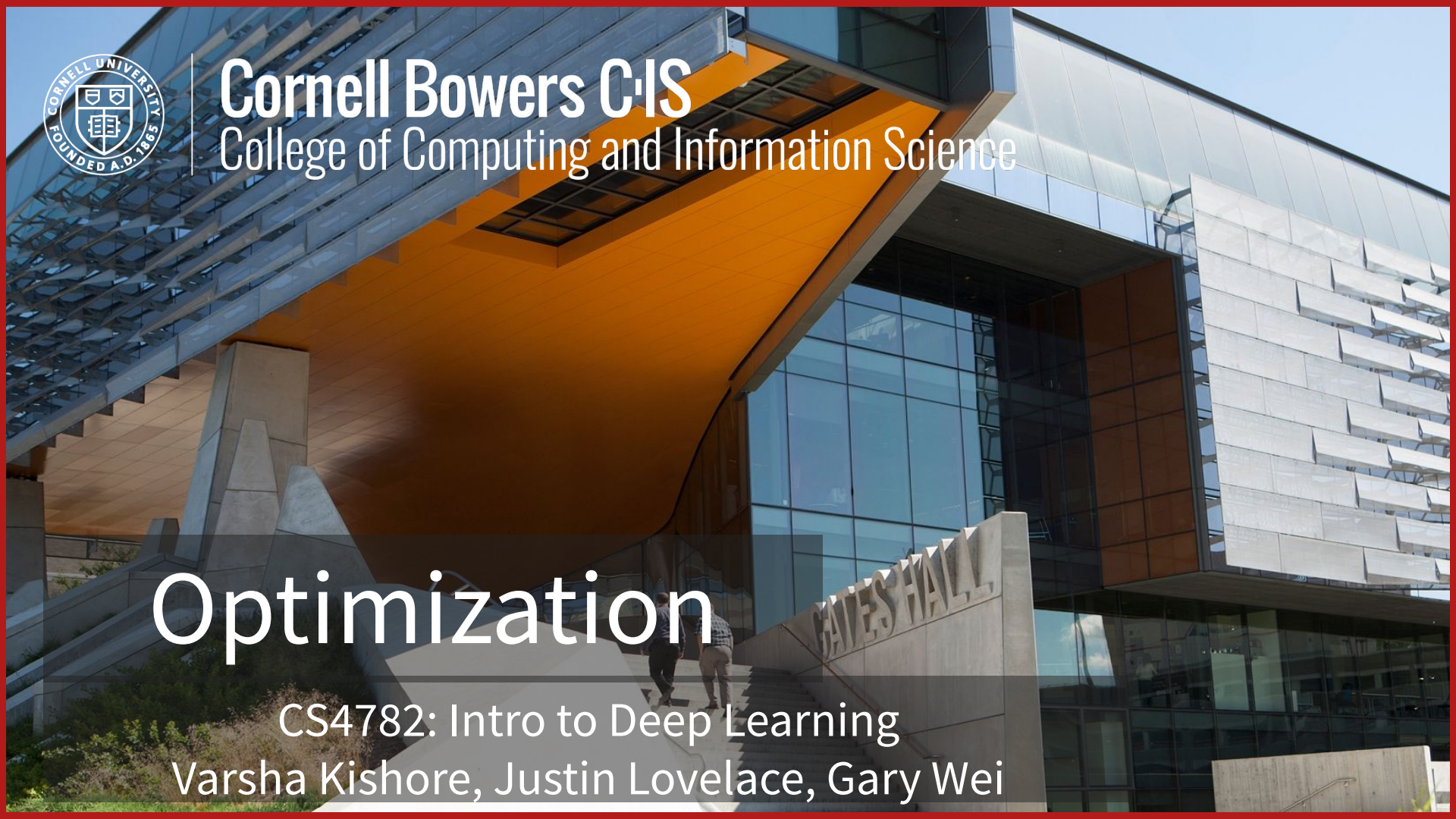


Optimization

CS4782: Intro to Deep Learning

Varsha Kishore, Justin Lovelace, Gary Wei

Agenda

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

How to learn MLP weights?

Gradient descent through backpropagation!

Calculus Review: The Chain Rule

Lagrange's Notation: $\text{If } h(x) = f(g(x)), \text{ then } h' = f'(g(x))g'(x)$

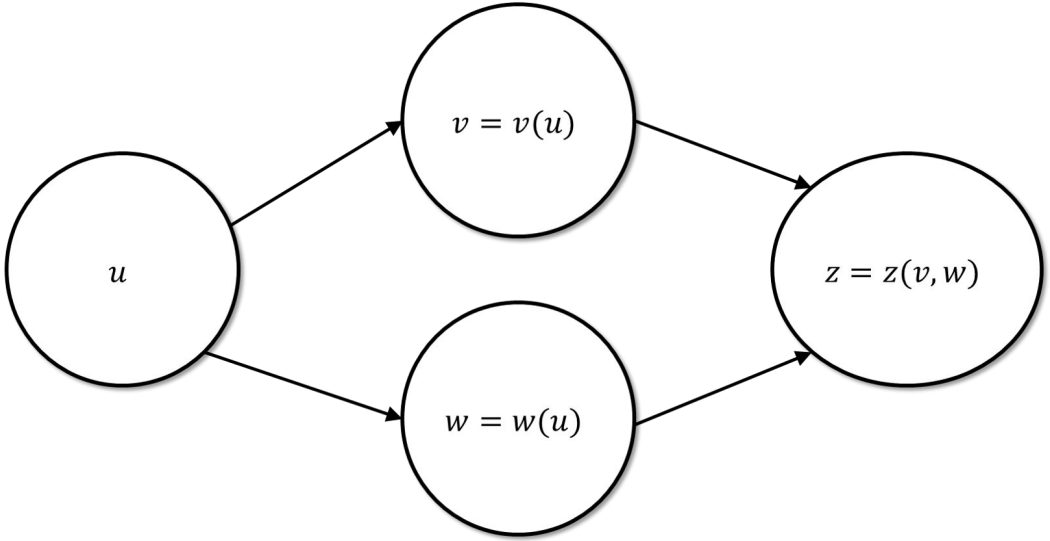
Leibniz's Notation: $\text{If } z = h(y), y = g(x), \text{ then } \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

Example: $\text{If } z = \ln(y), y = x^2, \text{ then}$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$= (\frac{1}{y})(2x) = (\frac{1}{x^2})(2x) = \frac{2}{x}$$

Multivariate Chain Rule

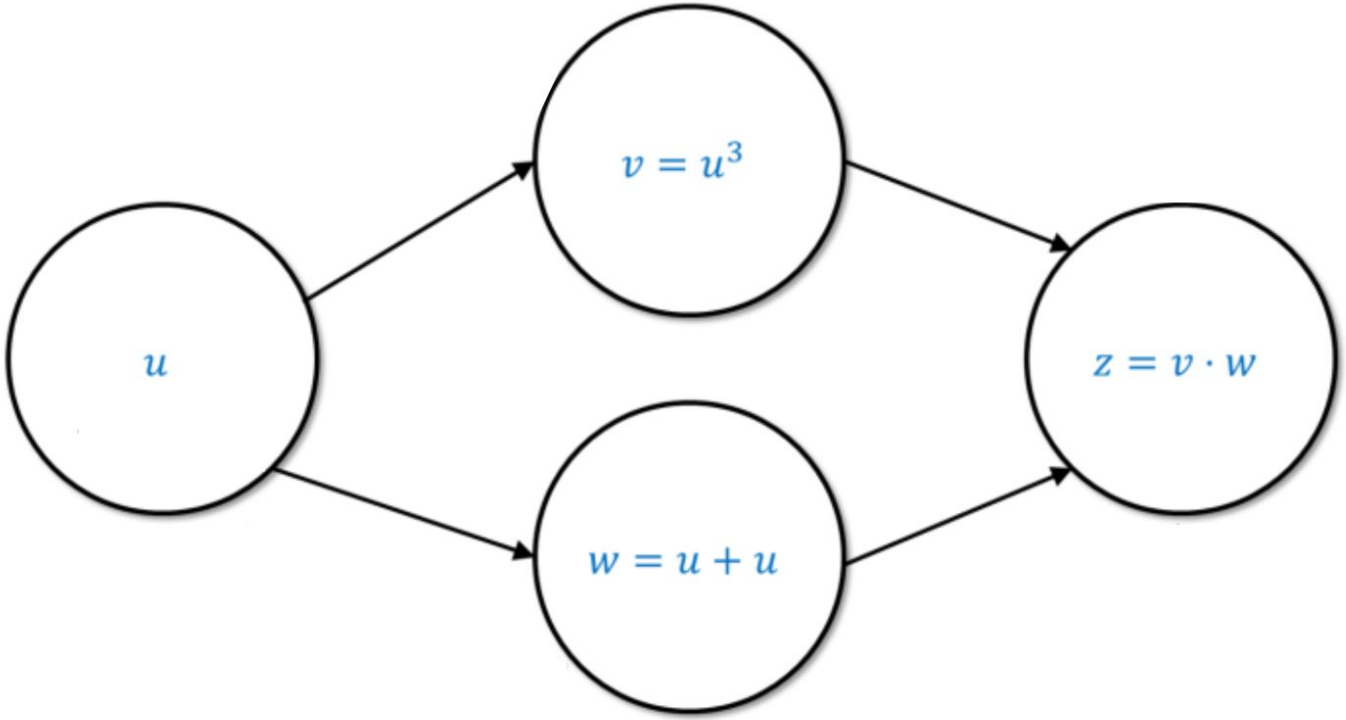


If $f(u)$ is $z = f(v(u), w(u))$, then

$$\frac{\partial f}{\partial u} = \left(\frac{\partial v}{\partial u}\frac{\partial z}{\partial v} + \frac{\partial w}{\partial u}\frac{\partial z}{\partial w}\right)$$

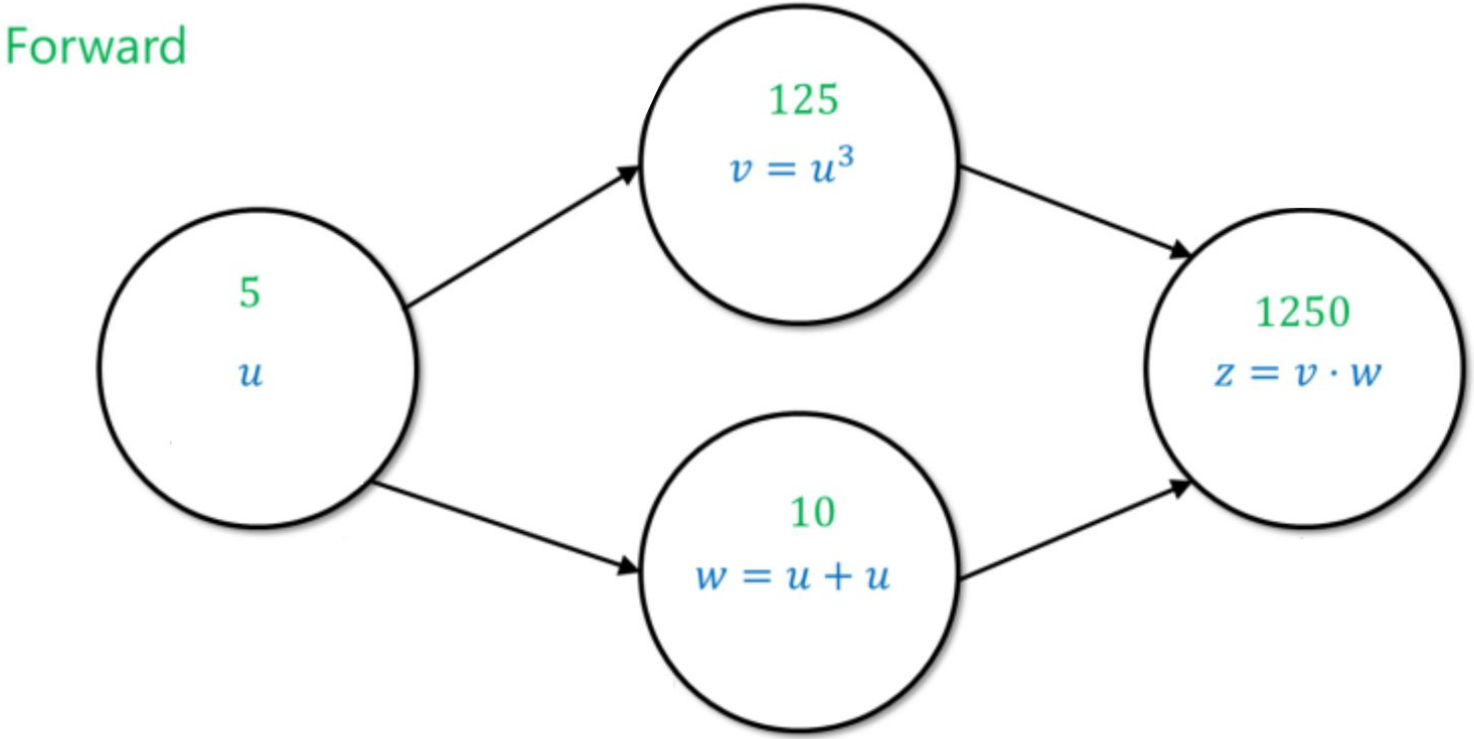
https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Backpropagation- An Example



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

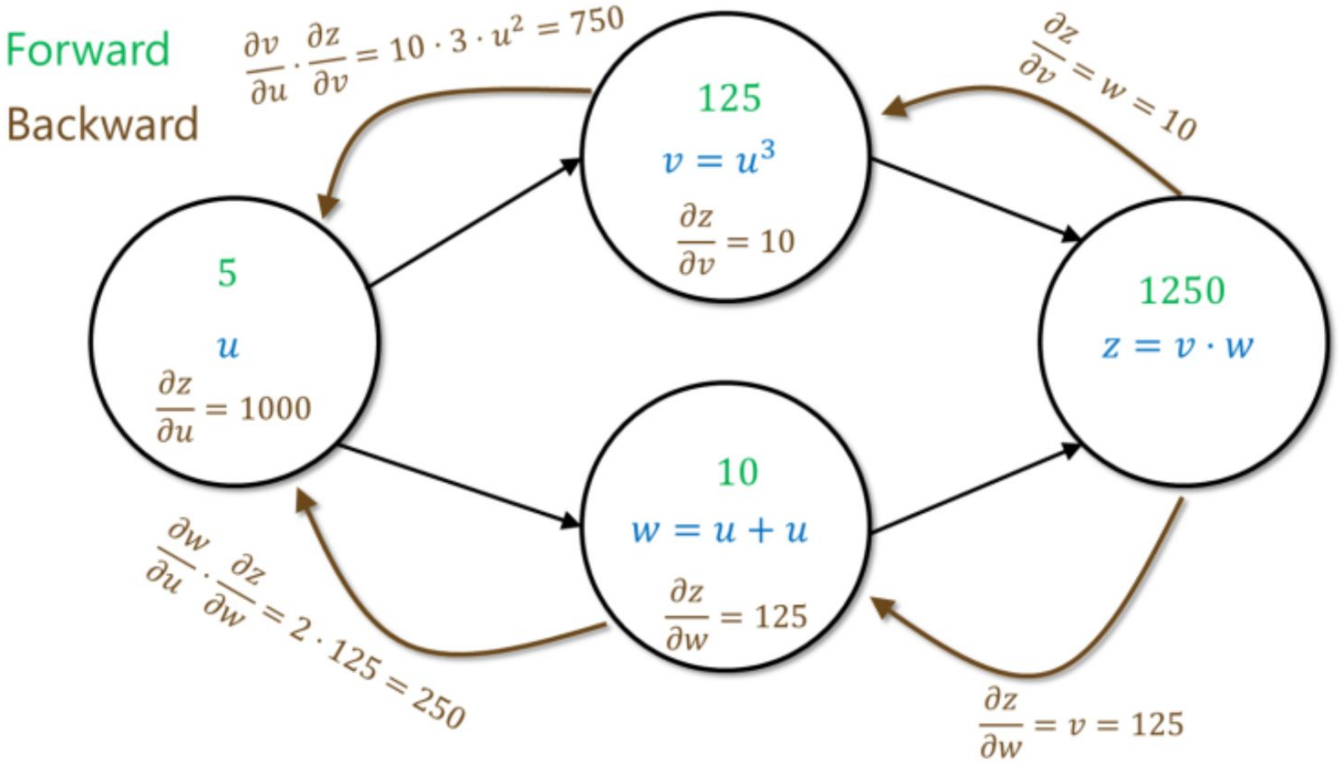
Backpropagation- An Example



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

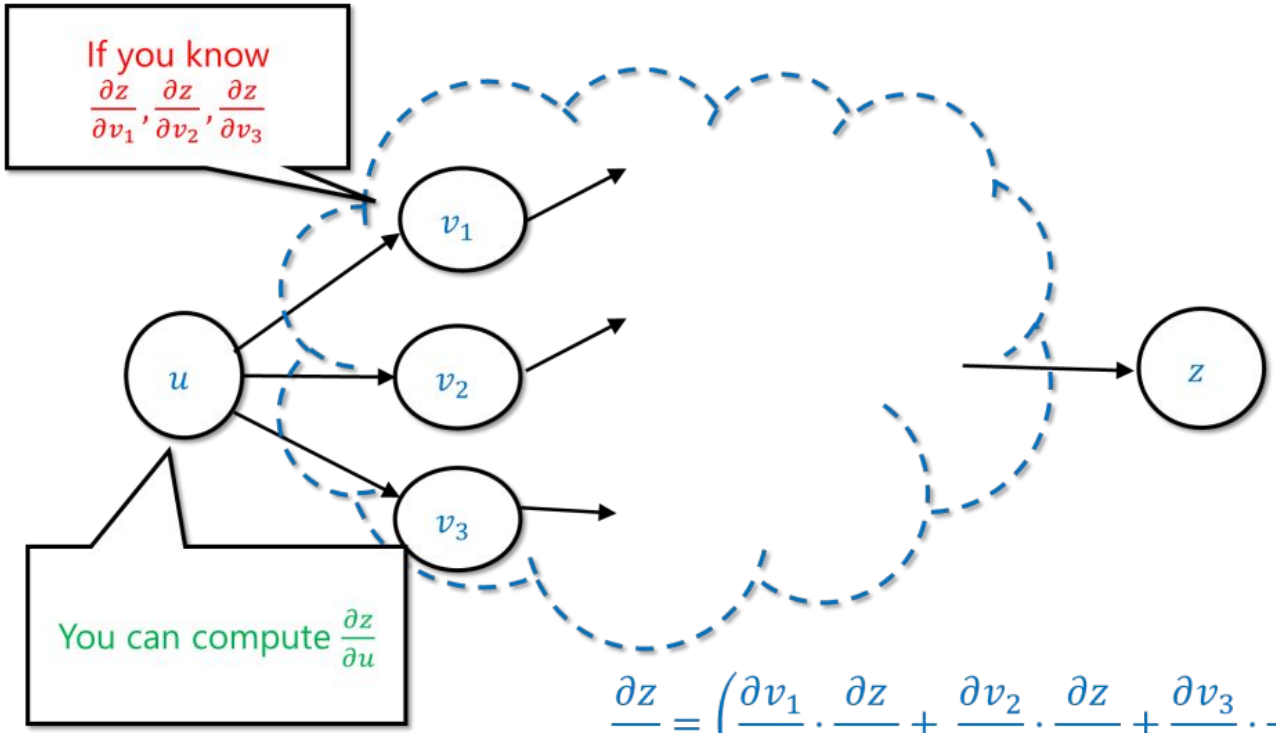
Backpropagation- An Example

$$\frac{\partial z}{\partial u} = \left(\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w}\right)$$



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Backpropagation- Key Idea



$$\frac{\partial z}{\partial u} = \left(\frac{\partial v_1}{\partial u} \cdot \frac{\partial z}{\partial v_1} + \frac{\partial v_2}{\partial u} \cdot \frac{\partial z}{\partial v_2} + \frac{\partial v_3}{\partial u} \cdot \frac{\partial z}{\partial v_3}\right)$$

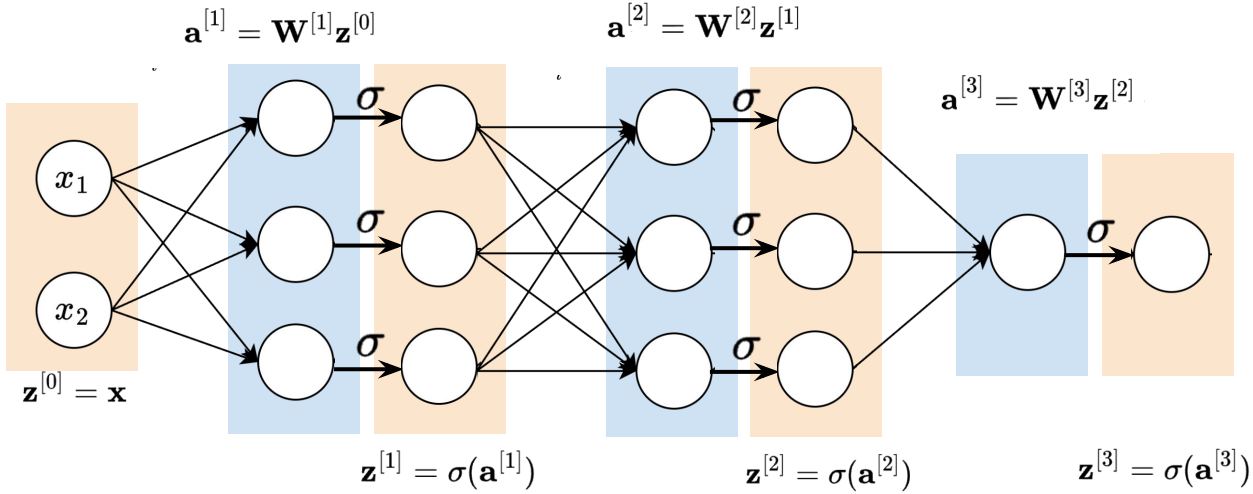
https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Cornell Bowers CIS

Forward Pass - MLP

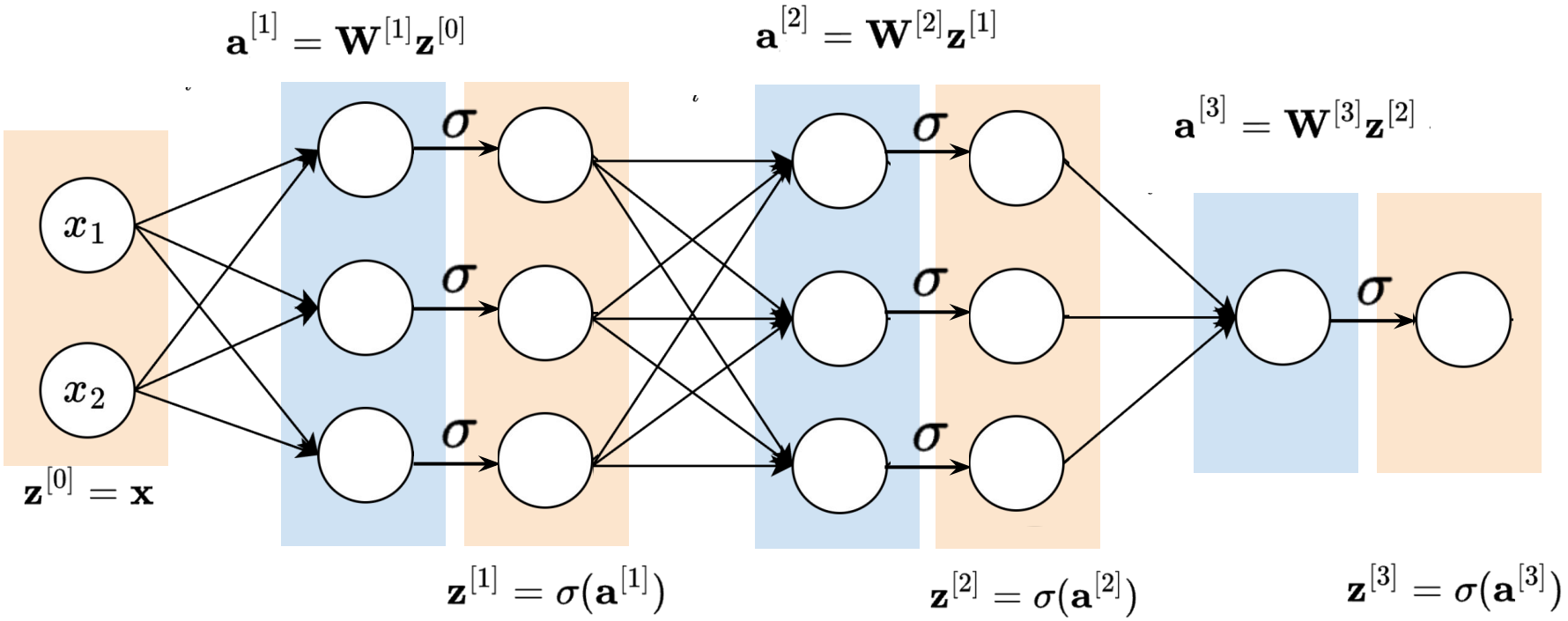
Algorithm Forward Pass through MLP

Input: input $\mathbf{x}$, weight matrices $\mathbf{W}^{[1]}, \ldots, \mathbf{W}^{[L]}$, bias vectors $\mathbf{b}^{[1]}, \ldots, \mathbf{b}^{[L]}$
$\mathbf{z}^{[0]} = \mathbf{x}$ ▷ Initialize input
for $l = 1$ **to** L **do**
  $\mathbf{a}^{[l]} = \mathbf{W}^{[l]}\mathbf{z}^{[l-1]} + \mathbf{b}^{[l]}$ ▷ Linear transformation
  $\mathbf{z}^{[l]} = \sigma^{[l]}(\mathbf{a}^{[l]})$ ▷ Nonlinear activation
end for
Output: $\mathbf{z}^{[L]}$



Backprop

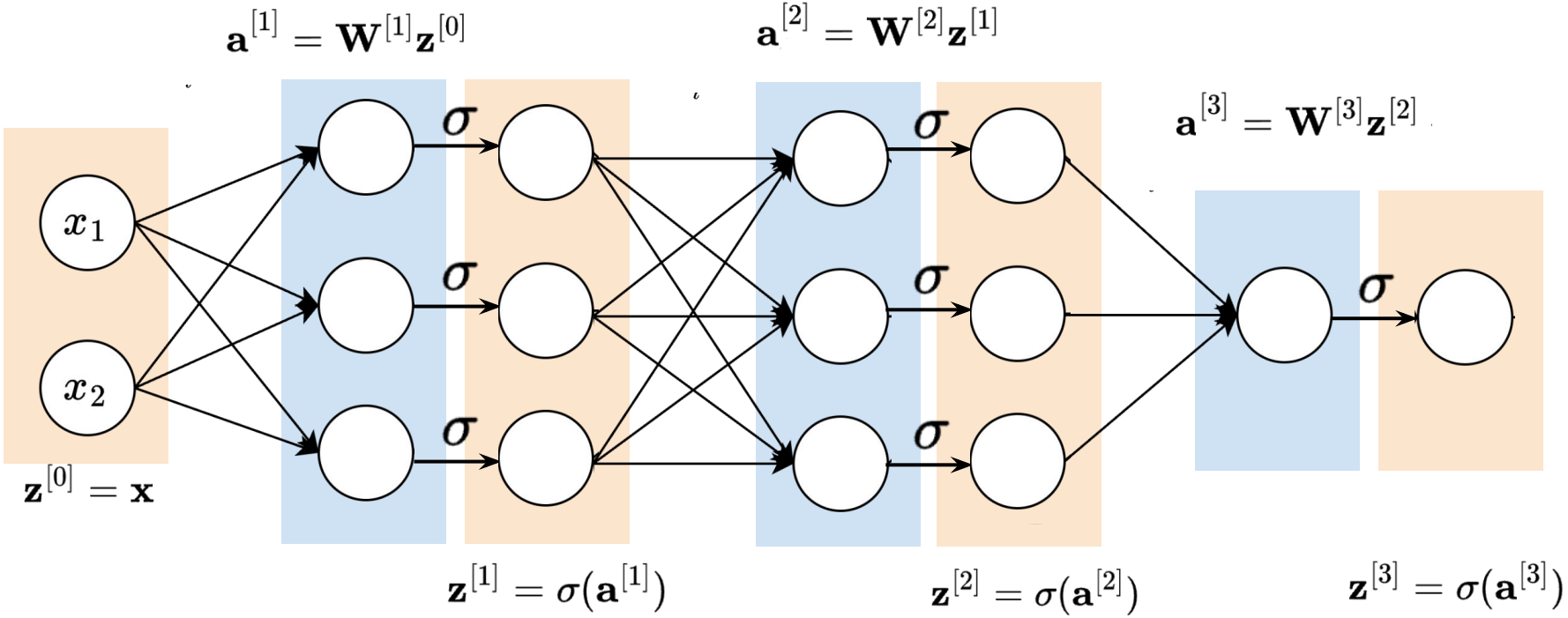
$$\text{Loss} = \mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$



Backprop

$$\text{Loss} = \mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

We can directly compute $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$!



Backprop

$$\text{Loss} = \mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

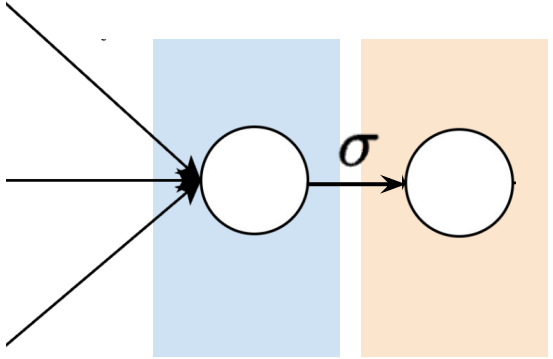
For propagation to next layer:

$$\boxed{\delta^{[3]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[3]}}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}} \odot \sigma^{[3]\prime}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$$

$$\mathbf{a}^{[3]} = \mathbf{W}^{[3]}\mathbf{z}^{[2]}$$



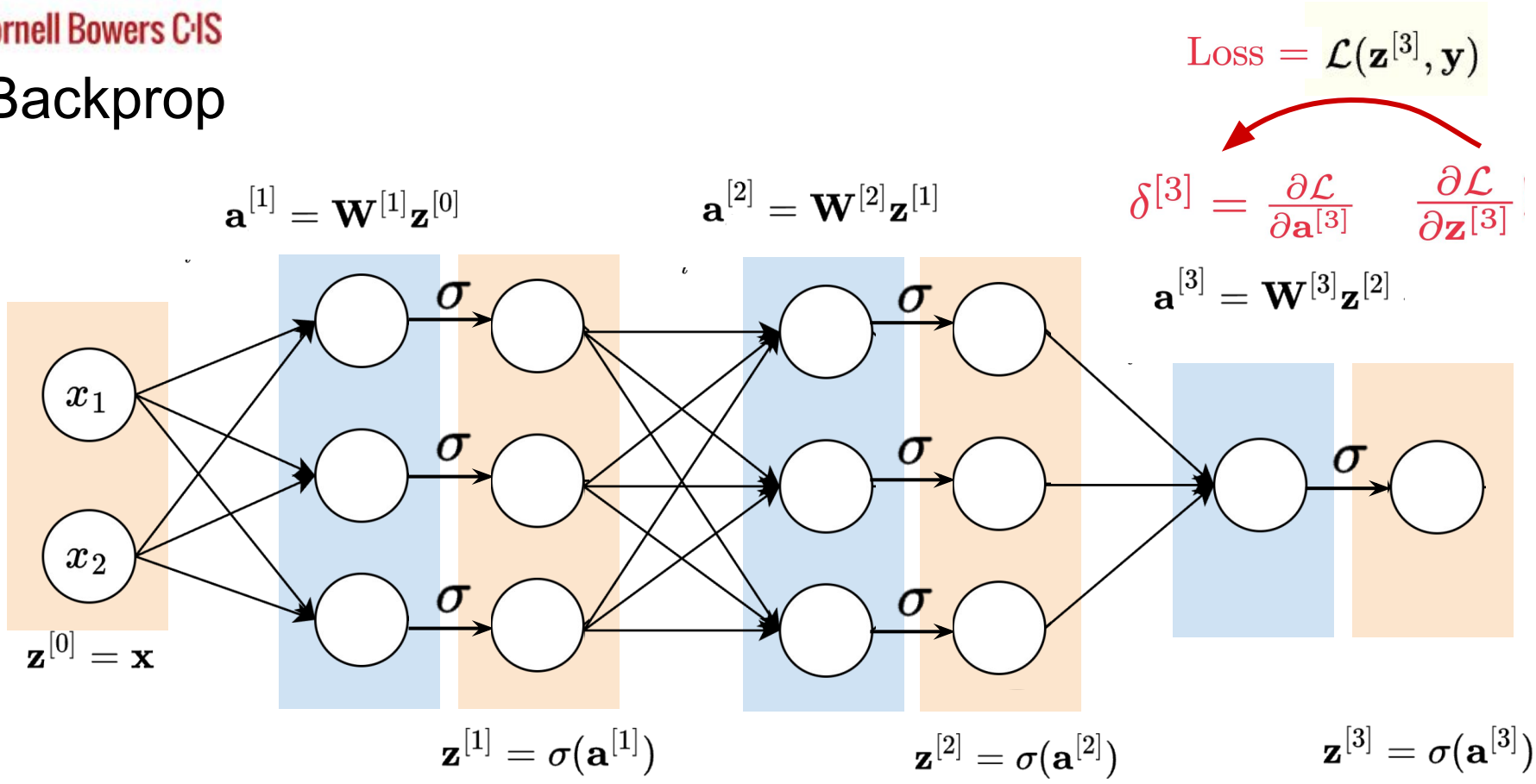
For weight updates:

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

$$= \delta^{[3]} (\mathbf{z}^{[2]})^T$$

$$\mathbf{z}^{[3]} = \sigma(\mathbf{a}^{[3]})$$

Backprop


$\text{Loss} = \mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$
$\delta^{[3]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$
$\mathbf{a}^{[1]} = \mathbf{W}^{[1]}\mathbf{z}^{[0]}$
$\mathbf{a}^{[2]} = \mathbf{W}^{[2]}\mathbf{z}^{[1]}$
$\mathbf{a}^{[3]} = \mathbf{W}^{[3]}\mathbf{z}^{[2]}$
x_1
x_2
σ
$\mathbf{z}^{[0]} = \mathbf{x}$
$\mathbf{z}^{[1]} = \sigma(\mathbf{a}^{[1]})$
$\mathbf{z}^{[2]} = \sigma(\mathbf{a}^{[2]})$
$\mathbf{z}^{[3]} = \sigma(\mathbf{a}^{[3]})$

Cornell Bowers CIS

$$\text{Loss} = \mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

$$\delta^{[1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{a}^{[1]}}$$

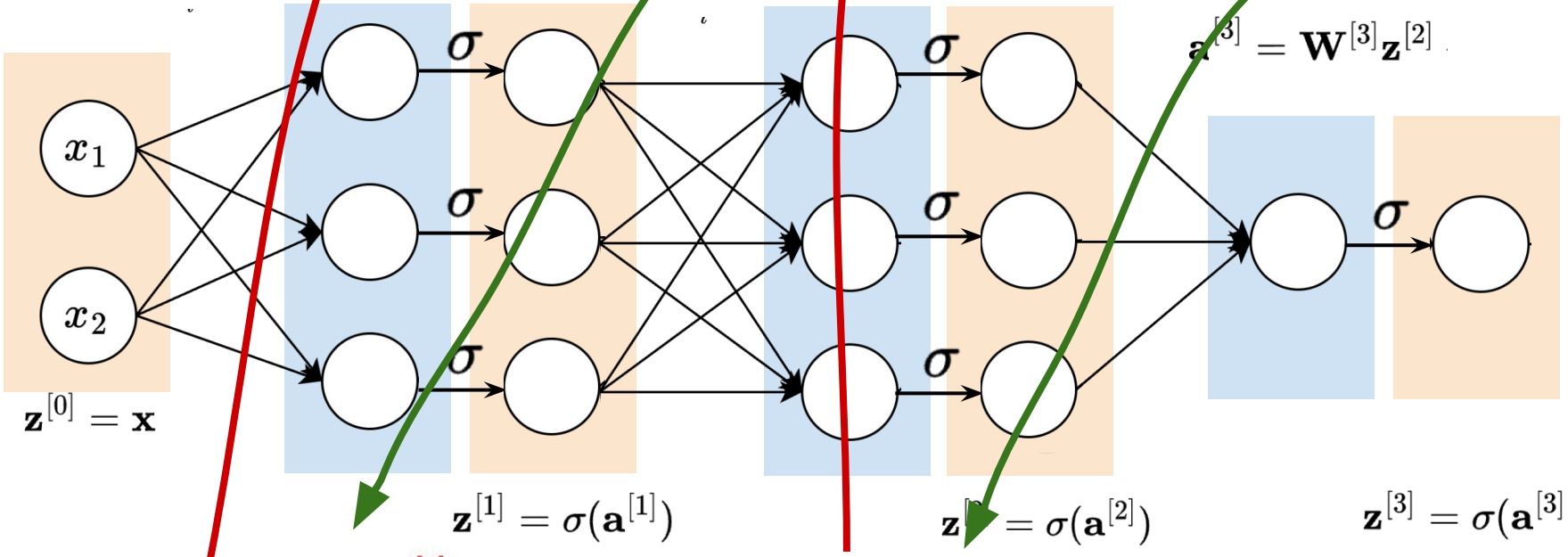
$$\delta^{[2]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[2]}}$$

$$\delta^{[3]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$$

$$\mathbf{a}^{[1]} = \mathbf{W}^{[1]}\mathbf{z}^{[0]}$$

$$\mathbf{a}^{[2]} = \mathbf{W}^{[2]}\mathbf{z}^{[1]}$$

$$\mathbf{a}^{[3]} = \mathbf{W}^{[3]}\mathbf{z}^{[2]}$$



$$\mathbf{z}^{[0]} = \mathbf{x}$$

$$\mathbf{z}^{[1]} = \sigma(\mathbf{a}^{[1]})$$

$$\mathbf{z}^{[2]} = \sigma(\mathbf{a}^{[2]})$$

$$\mathbf{z}^{[3]} = \sigma(\mathbf{a}^{[3]})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[1]}} = (W^{[2]})^T \delta^{[2]}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[2]}} = (W^{[3]})^T \delta^{[3]}$$

Cornell Bowers CIS

Backpropagation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{W}^{[3]}} = \delta^{[3]} (\mathbf{z}^{[2]})^T$$

Algorithm Backward Pass through MLP (Detailed)

1: **Input:** $\{\mathbf{z}^{[1]}, \ldots, \mathbf{z}^{[L]}\}$, $\{\mathbf{a}^{[1]}, \ldots, \mathbf{a}^{[L]}\}$, loss gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}}$

2: $\delta^{[L]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \frac{\partial \mathbf{z}^{[L]}}{\partial \mathbf{a}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \odot \sigma^{[L]'}(\mathbf{a}^{[L]})$ ▷ Error term

3: **for** $l = L$ **to** 1 **do**

4: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{W}^{[l]}} = \delta^{[l]} (\mathbf{z}^{[l-1]})^T$ ▷ Gradient of weights

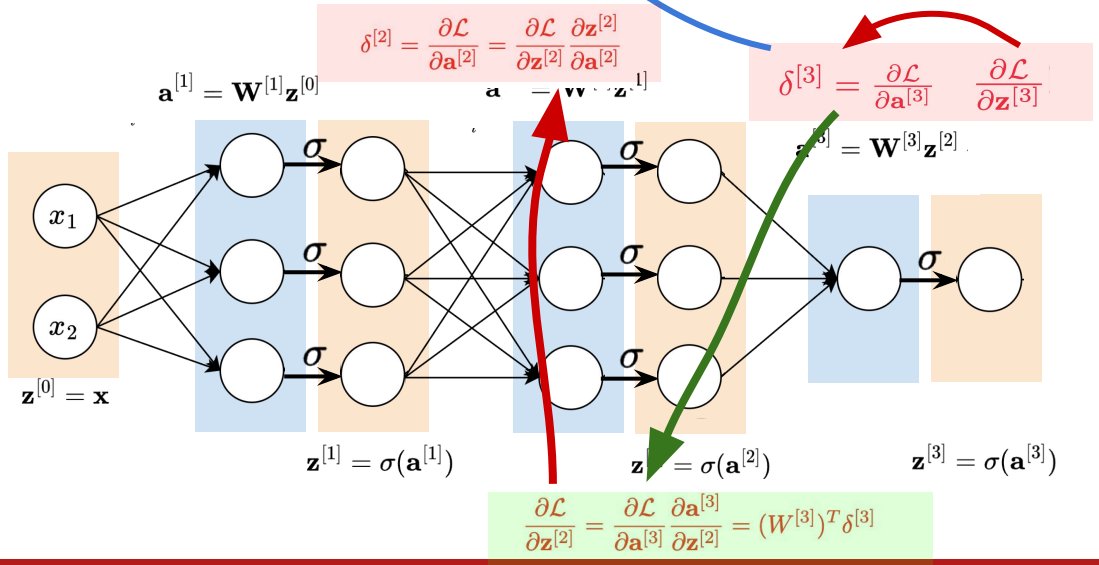
5: $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$ ▷ Gradient of biases

6: $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{z}^{[l-1]}} = (\mathbf{W}^{[l]})^T \delta^{[l]}$

7: $\delta^{[l-1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} \frac{\partial \mathbf{z}^{[l-1]}}{\partial \mathbf{a}^{[l-1]}} = ((\mathbf{W}^{[l]})^T \delta^{[l]}) \odot \sigma^{[l-1]'}(\mathbf{a}^{[l-1]})$

8: **end for**

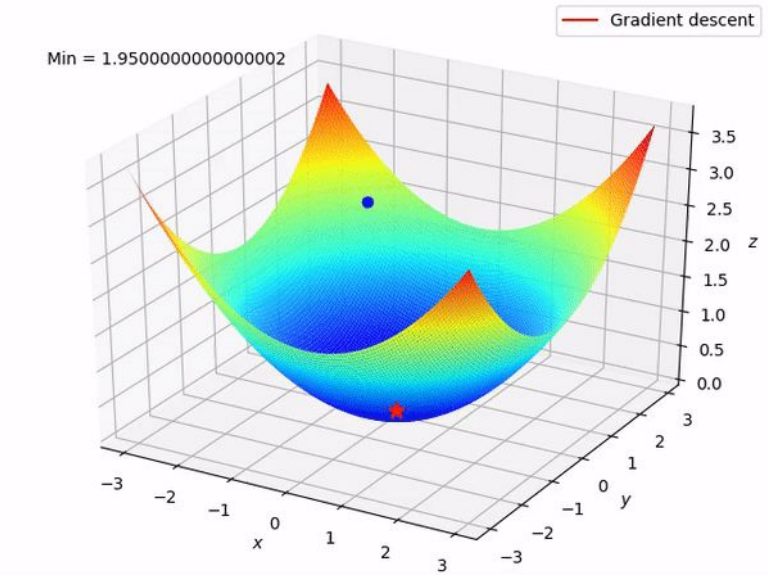
9: **Output:** $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$



$$\mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

We can directly compute $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$!

What is Optimization?



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

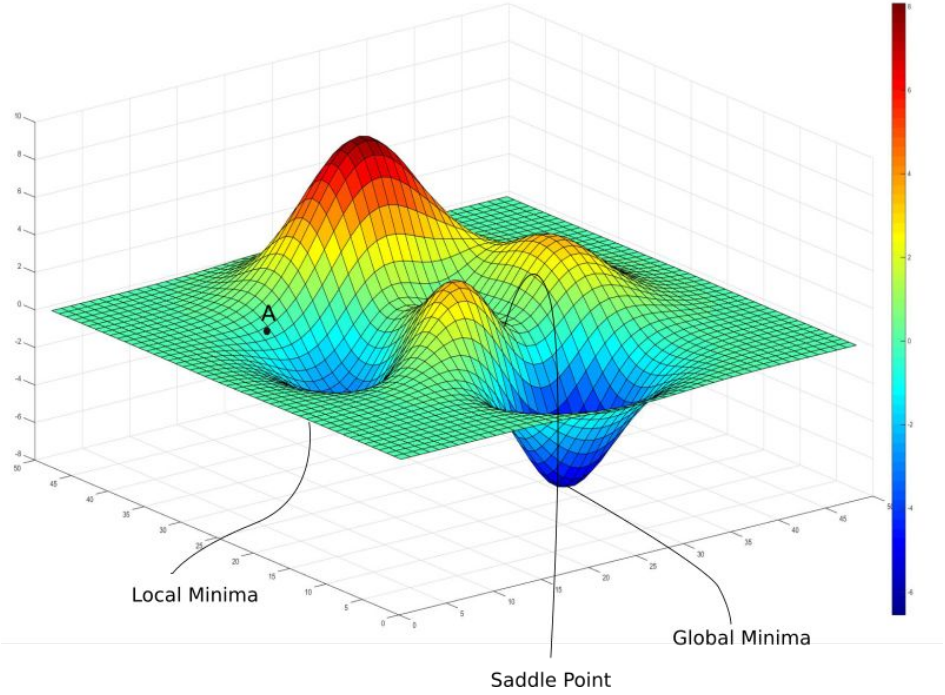
Loss function

Empirical Risk:

$$\mathcal{L}(\mathbf{w}_t) = \frac{1}{n} \sum_{i=1,\ldots,n} \ell(\mathbf{w}_t, \mathbf{x}_i)$$

t : at time step t
$\mathbf{w}_t$: Model weights (parameters) at time t
$\mathbf{x}_i$: The i-th input training data

$\mathcal{L}$: the Loss function (optimization target)
ℓ : per-sample loss

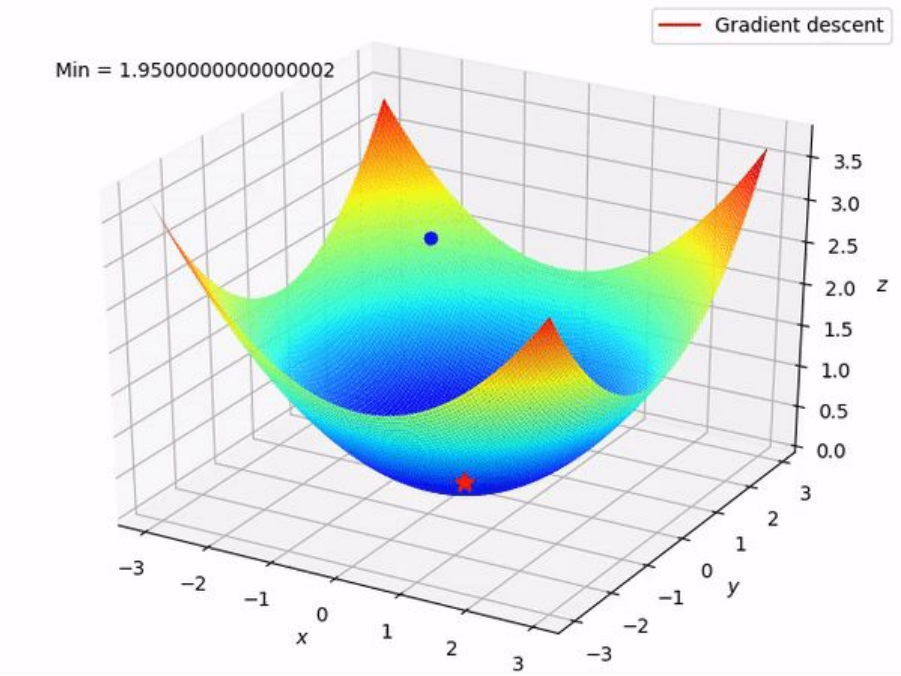


Gradient Descent (GD)

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

α: the learning rate

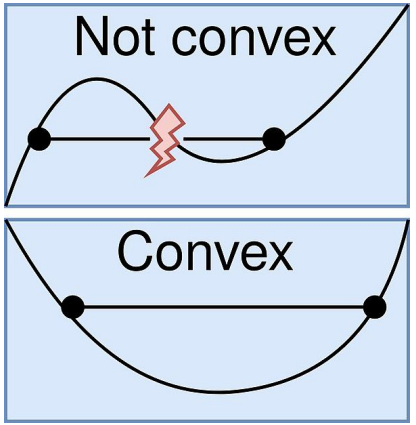
$\nabla \mathcal{L}(\mathbf{w}_t)$: the gradient of Loss w.r.t. $\mathbf{w}_t$

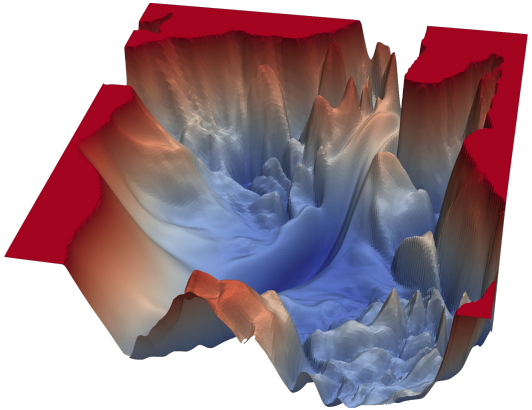


What are some potential problems with gradient descent?

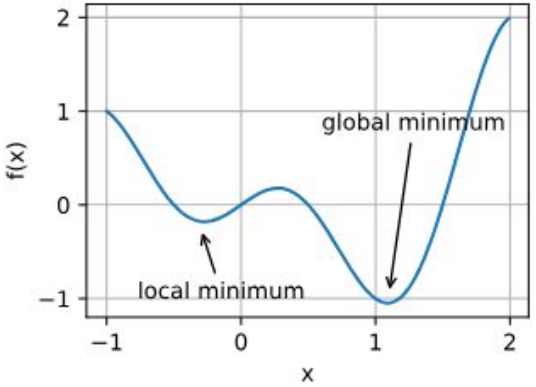
Convexity

- 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!

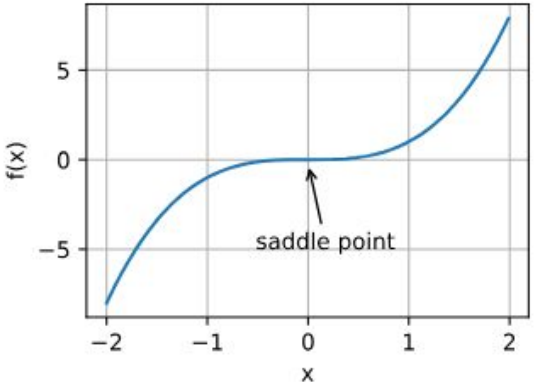




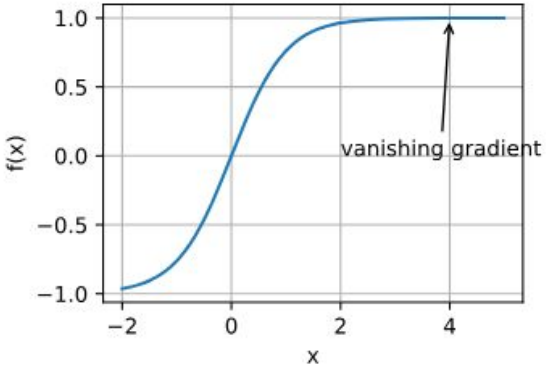
Challenges in Non-Convex Optimization



Local Minima vs. Global Minima



Saddle Points



Vanishing gradient

Gradient Descent (GD)

$$\mathcal{L}(\mathbf{w}_t) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}_t, \mathbf{x}_i)$$

$$\nabla\mathcal{L}(\mathbf{w}_t) = \frac{1}{n}\sum_{i=1}^{n} \nabla\ell(\mathbf{w}_t, \mathbf{x}_i)$$

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

Full gradient: $\mathcal{O}(n)$ time => ***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)

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

Stochastic Gradient Descent (SGD)

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

Select **1** example randomly each time

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

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)

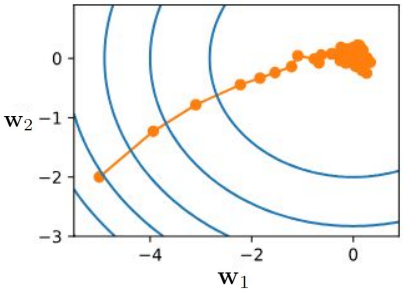
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

Select **1** example randomly each time

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

Trade off convergence!

Per-sample gradients not necessarily points to the local minimum, introducing a ***noise ball****...*



Minibatch SGD

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

Select **1** example randomly each time

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

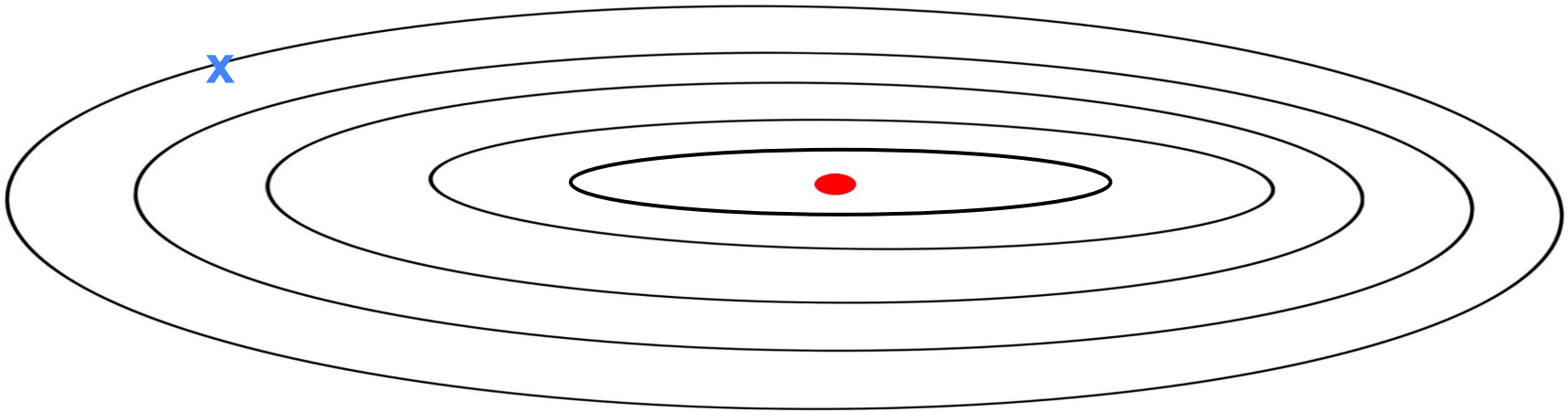
Select a batch $\mathcal{B}_t$ of examples randomly each time, with *batch size* b

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

Local Minimum

Draw the gradients:

- Smaller learning rate
- Larger learning rate



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 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

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

$$\longrightarrow$$

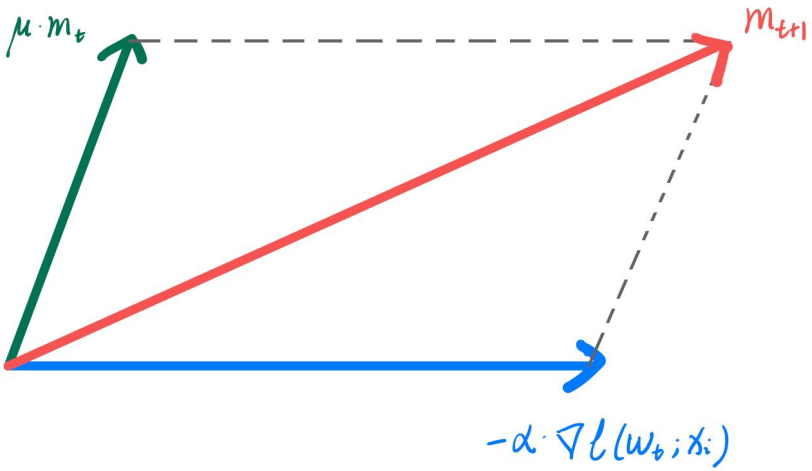
$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

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.*

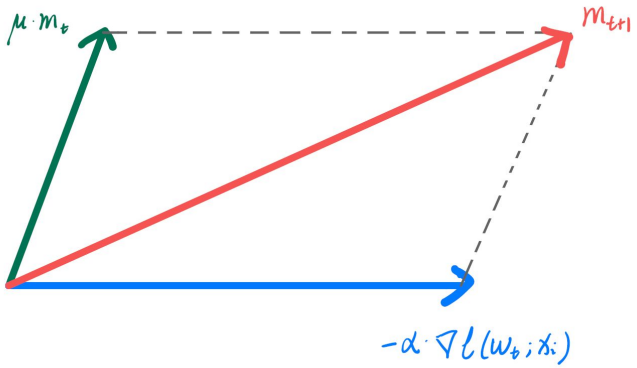


$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

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.*



$$
\begin{aligned}
\mathbf{g}_t &= \nabla l(\mathbf{w}_t; \mathbf{x}_i) \\
\mathbf{m}_{t+1} &= \mu \mathbf{m}_t - \alpha \mathbf{g}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \mu \mathbf{m}_t - \alpha \mathbf{g}_t \\
&= \mathbf{w}_t + \mu(\mu \mathbf{m}_{t-1} - \alpha \mathbf{g}_{t-1}) - \alpha \mathbf{g}_t \\
&= \mathbf{w}_t + \mu(\mu(\mu \mathbf{m}_{t-2} - \alpha \mathbf{g}_{t-2}) - \alpha \mathbf{g}_{t-1}) - \alpha \mathbf{g}_t \\
&= \mathbf{w}_t - \alpha \mathbf{g}_t - \mu \alpha \mathbf{g}_{t-1} - \mu^2 \alpha \mathbf{g}_{t-2} - \mu^3 \alpha \mathbf{g}_{t-3} - \dots \\
&= \mathbf{w}_t - \alpha \sum_{i=0}^{t} \mu^i \mathbf{g}_{t-i}
\end{aligned}
$$

Quick Recap

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

Minibatch SGD

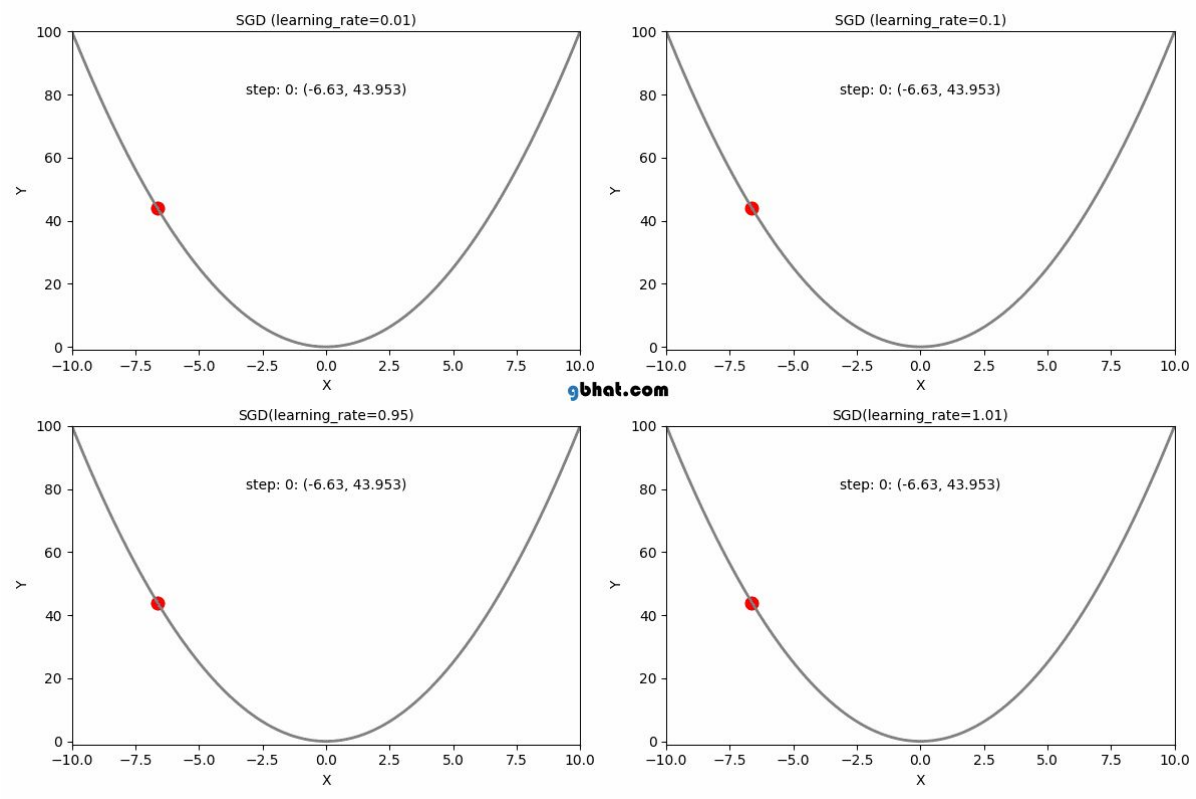
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

SGD w. Momentum

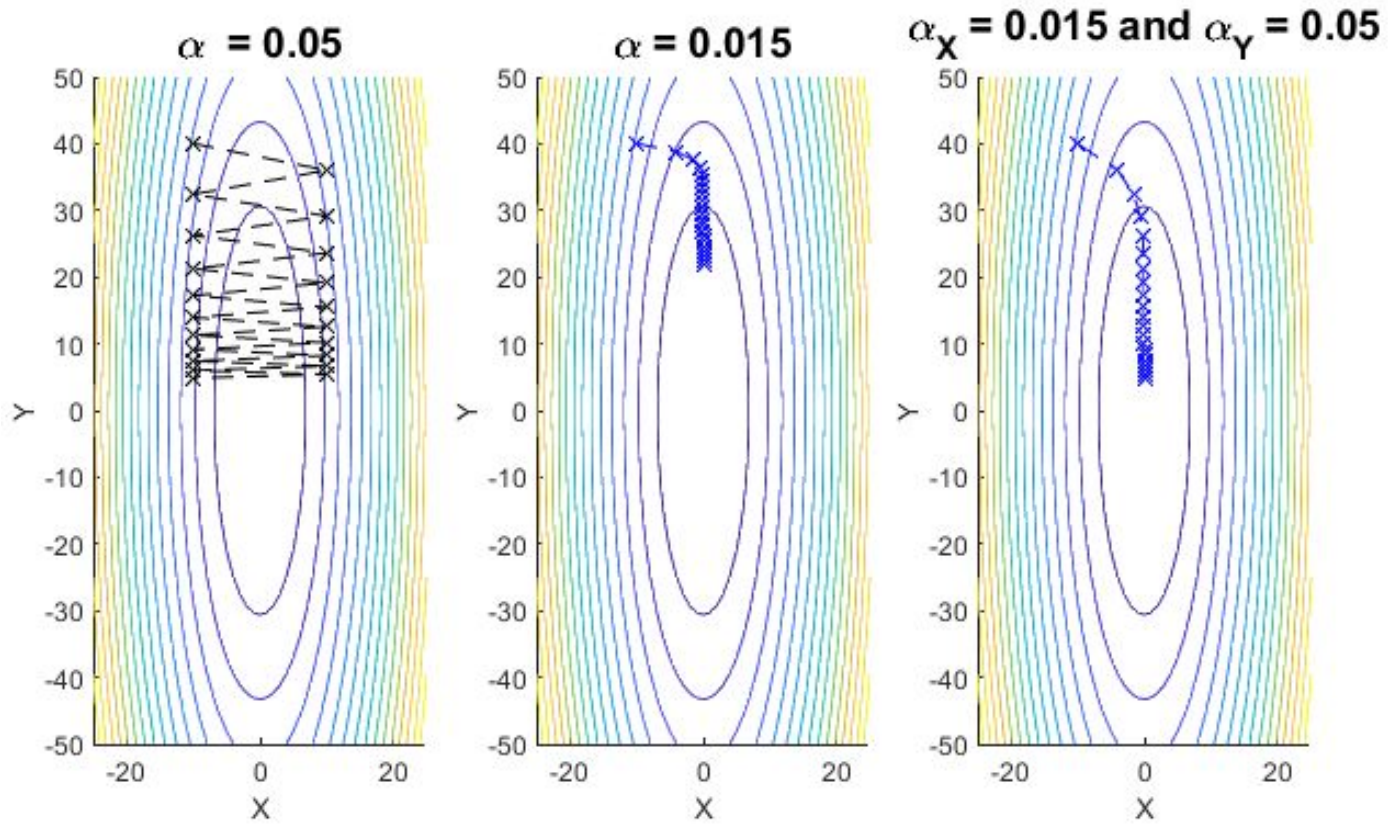
$$m_{t+1} = \mu m_t - \alpha \nabla l(w_t; x_i)$$

$$w_{t+1} = w_t + m_{t+1}$$

Importance of Learning Rate



Another example



Adaptive Optimizers

Different Learning Rate** for **each element** of the **Model Weights!

AdaGrad (Duchi et al. 2011)

More updates -> more decay

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

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

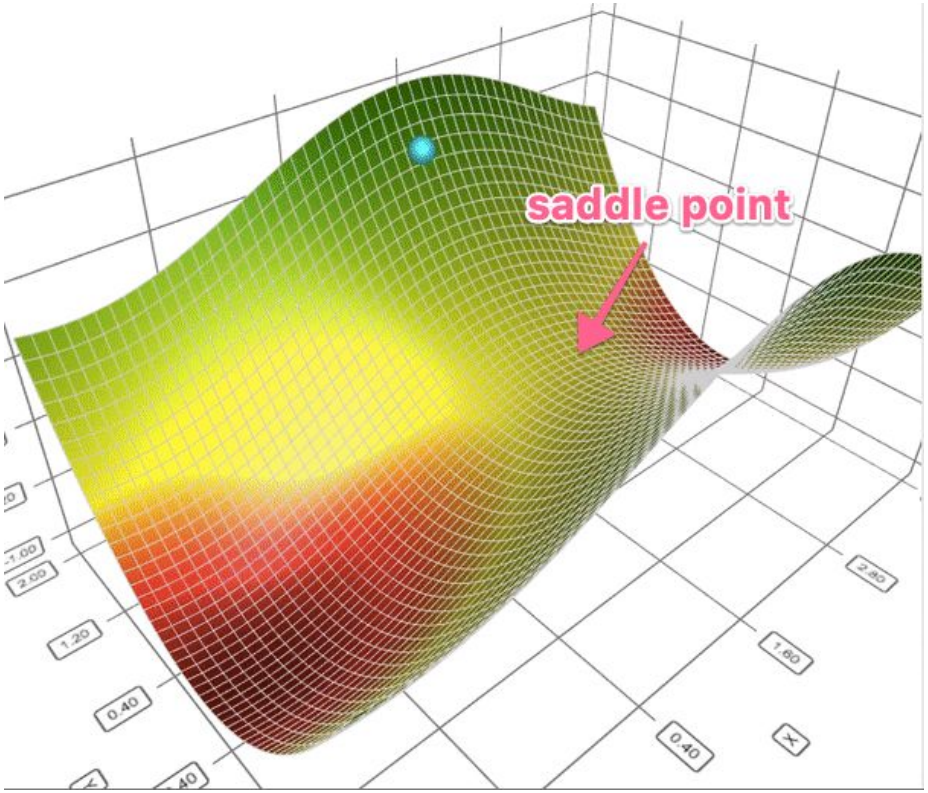
SGD

Element-wise product

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Adagrad



Gradient Descent
AdaGrad

RMSProp (Graves, 2013)

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

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Adagrad

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

RmsProp

where $\beta \in [0, 1]$ the exponential moving average constant.

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

Momentum

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

RMSProp

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

RMSProp

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\widehat{\mathbf{v}}_{t+1} + \epsilon}} \odot \widehat{\mathbf{m}}_{t+1}$$

ADAM
(**Ada**ptive **M**oment Estimate)

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

Momentum

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1-\beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

RMSProp

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1-\beta_1)\mathbf{g}_t$$
$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1-\beta_2)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_{t+1} + \epsilon}} \odot \hat{\mathbf{m}}_{t+1}$$

ADAM
(**Ada**ptive **M**oment Estimate)

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

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

$$\widehat{\mathbf{m}}_{t+1} = \frac{\mathbf{m}_{t+1}}{1-\beta_1^{t+1}}$$

$$\widehat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1-\beta_2^{t+1}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\widehat{\mathbf{v}}_{t+1}+\epsilon}} \odot \widehat{\mathbf{m}}_{t+1}$$

$\mathbb{E}[\widehat{\mathbf{m}}_{t+1}]$

$\mathbb{E}[\widehat{\mathbf{v}}_{t+1}]$

ADAM
(**Ada**ptive **M**oment 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

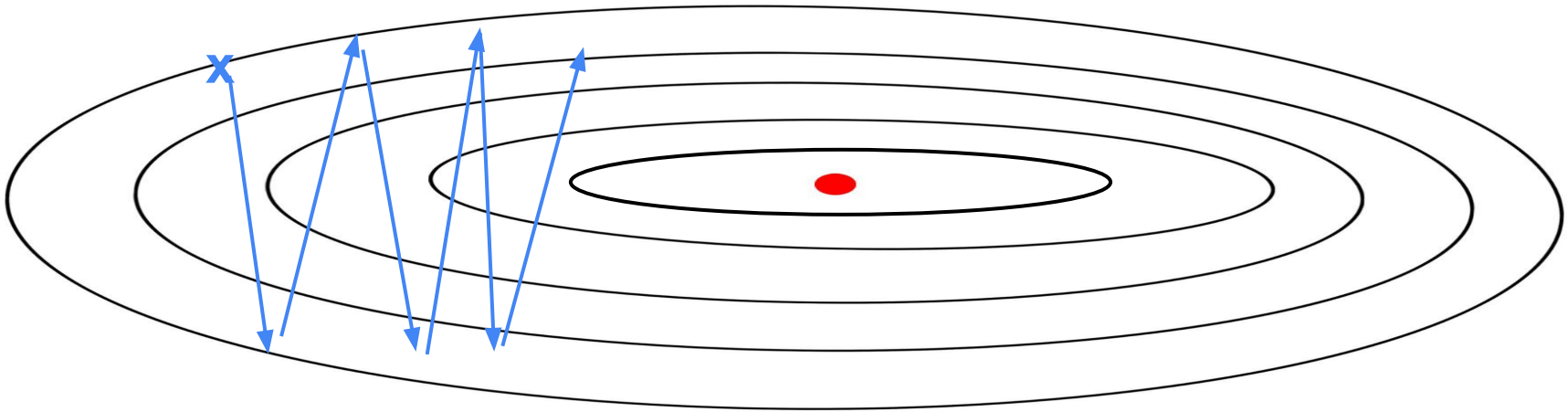
The **optimizer** determines which minimizer you converge to

Agenda

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

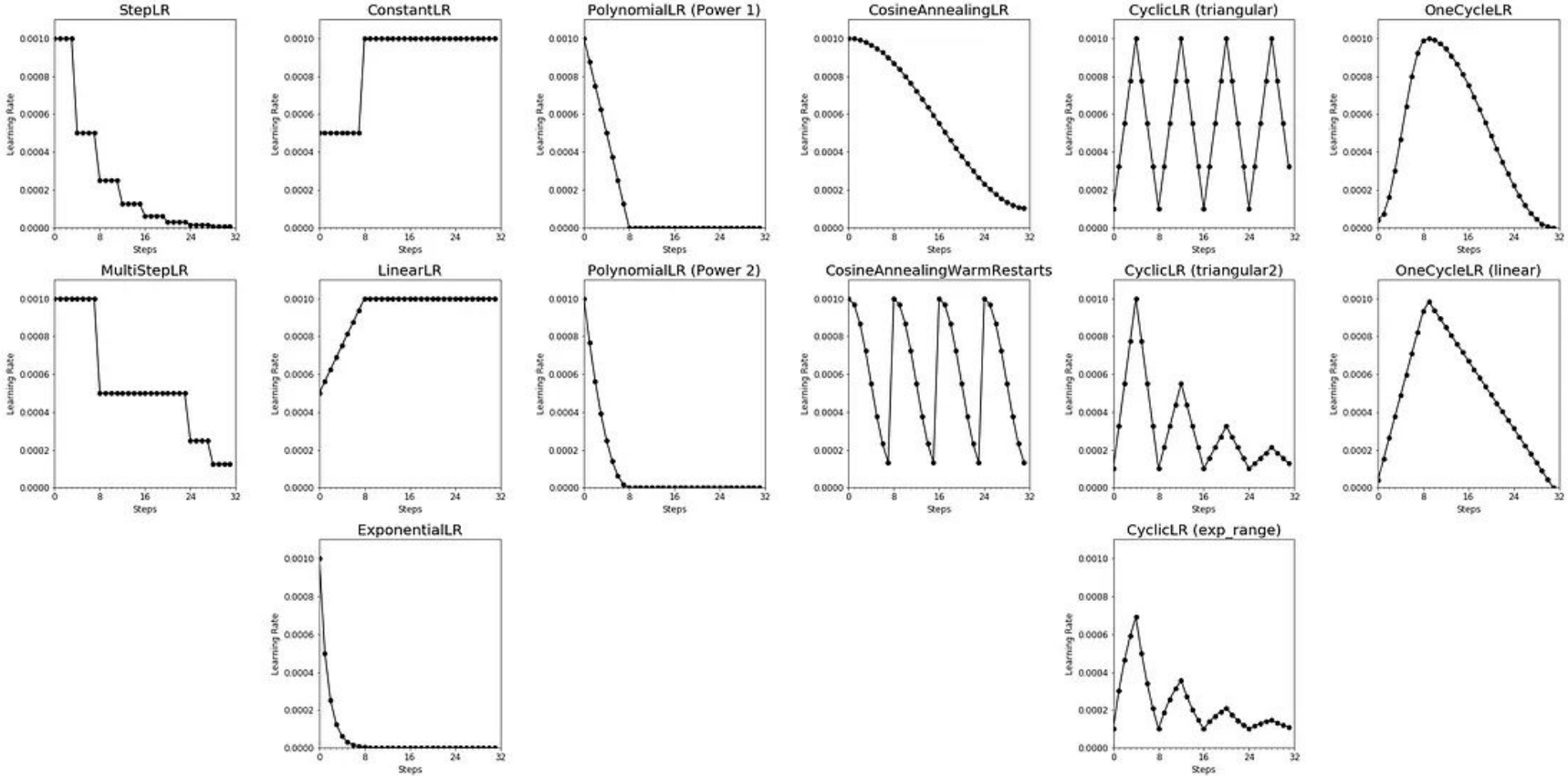
Recall: Draw the gradients

- Smaller learning rate
- Larger learning rate



Cornell Bowers CIS

Learning Rate Scheduling



Training Hyper-Parameters.

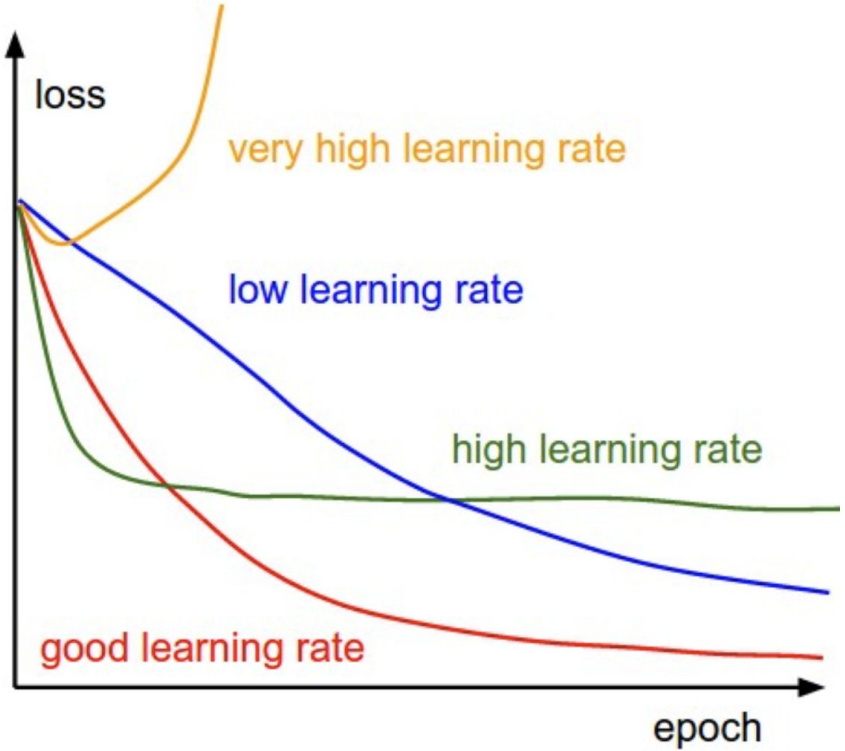
We employ the AdamW optimizer (Loshchilov and Hutter, 2017) with hyper-parameters set to $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\text{weight\_decay} = 0.1$. We set the maximum sequence length to 4K during pre-training, and pre-train DeepSeek-V3 on 14.8T tokens. As for the learning rate scheduling, we first linearly increase it from 0 to 2.2×10^{-4} during the first 2K steps. Then, we keep a constant learning rate of 2.2×10^{-4} until the model consumes 10T training tokens. Subsequently, we gradually decay the learning rate to 2.2×10^{-5} in 4.3T tokens, following a cosine decay curve. During the training of the final 500B tokens, we keep a constant learning rate of 2.2×10^{-5} in the first 333B tokens, and switch to another constant learning rate of 7.3×10^{-6} in the remaining 167B tokens. The gradient clipping norm is set to 1.0. We employ a batch size scheduling strategy, where the batch size is gradually increased from 3072 to 15360 in the training of the first 469B tokens, and then keeps 15360 in the remaining training. We leverage pipeline parallelism to deploy different layers of a model on different GPUs, and for each layer, the routed experts will be uniformly deployed on 64 GPUs belonging to 8 nodes. As for the node-limited routing, each token will be sent to at most 4 nodes (i.e., $M = 4$). For auxiliary-loss-free load balancing, we set the bias update speed γ to 0.001 for the first 14.3T tokens, and to 0.0 for the remaining 500B tokens. For the balance loss, we set α to 0.0001, just to avoid extreme imbalance within any single sequence. The MTP loss weight λ is set to 0.3 for the first 10T tokens, and to 0.1 for the remaining 4.8T tokens.

[Deep Seek V3 Technical Report, 2024]

Hyperparameters

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

These are all hyperparameters that affect performance!



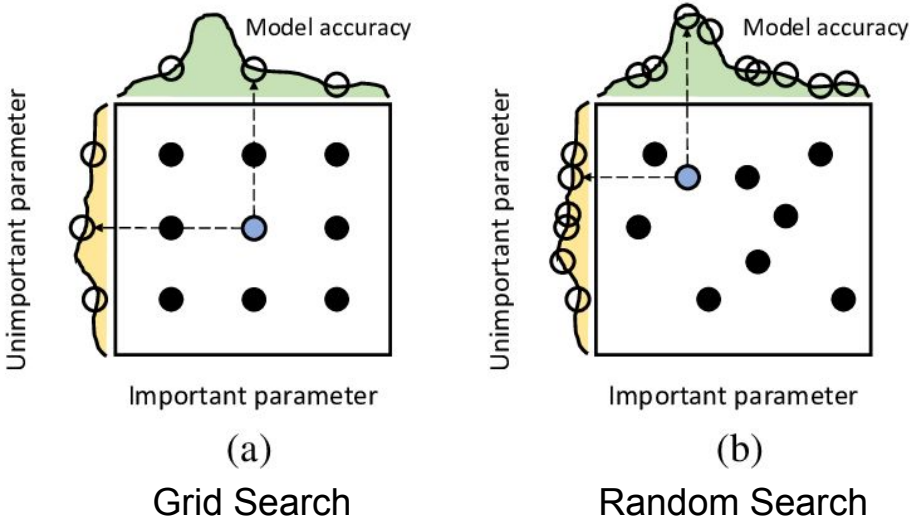
Source: https://cs231n.github.io/neural-networks-3/

Hyperparameter Optimization (HPO)

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

These are all hyperparameters that affects 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!*