



Cornell Bowers C-IS

College of Computing and Information Science

Optimization

CS4782: Intro to Deep Learning

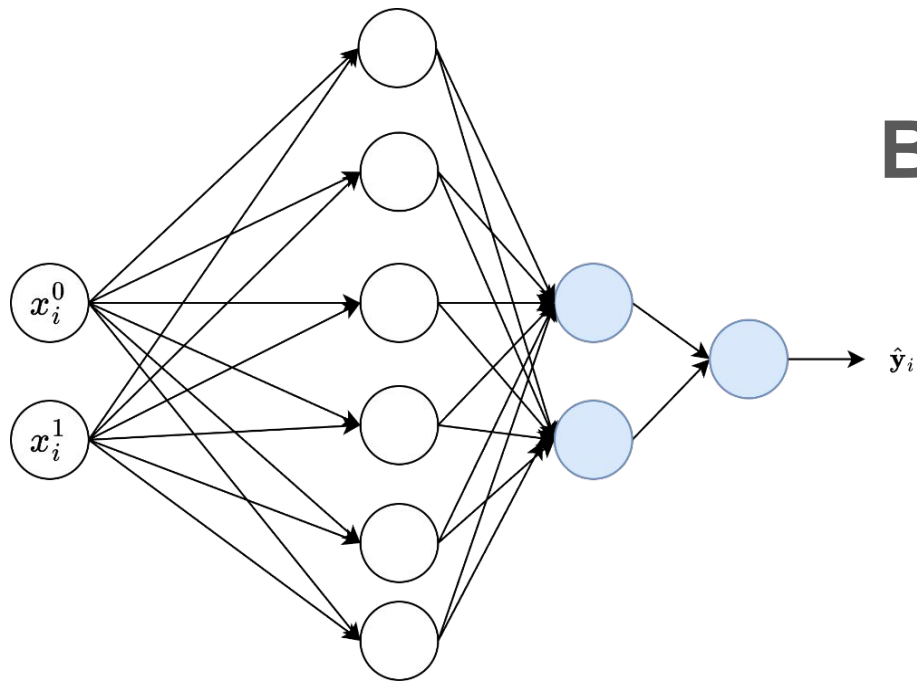
Before we start:

Look at recap quizzes 1,2,3

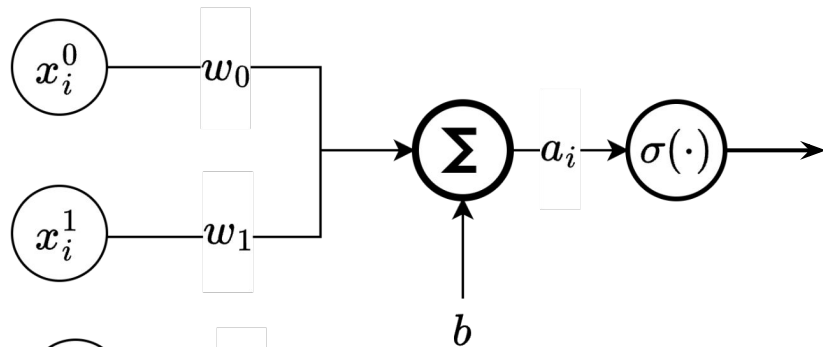
Discuss with your neighbors

Recap 1

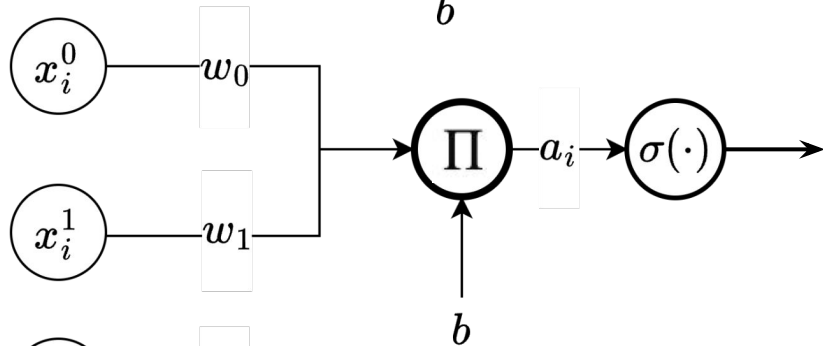
What's inside one of the circles?



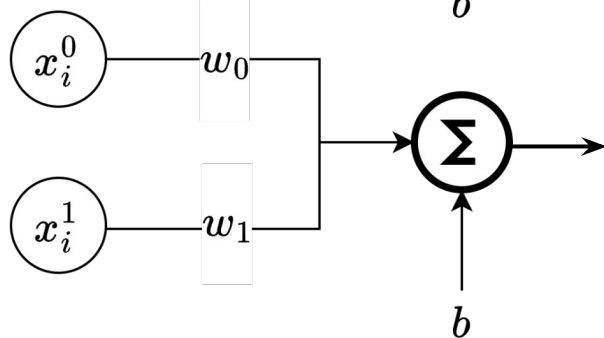
A.



B.

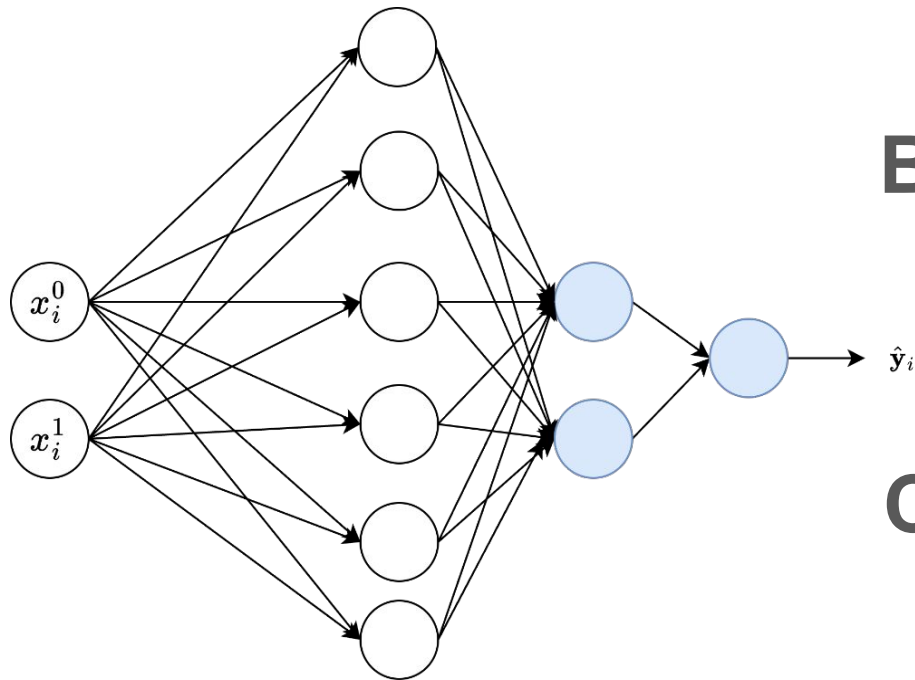


C.

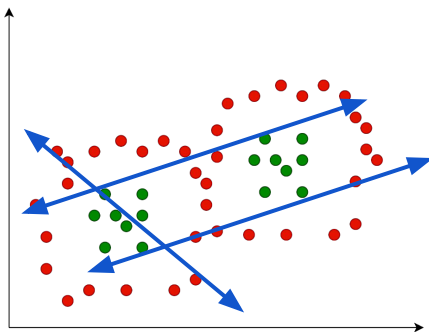


Recap 2

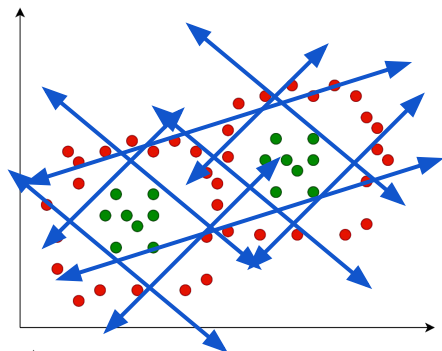
Which decision boundary cannot be learned by this network?



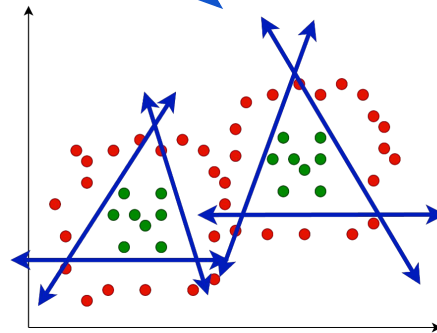
A.



B.



C.



Recap 3

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

\mathbf{x} : (2, 1)

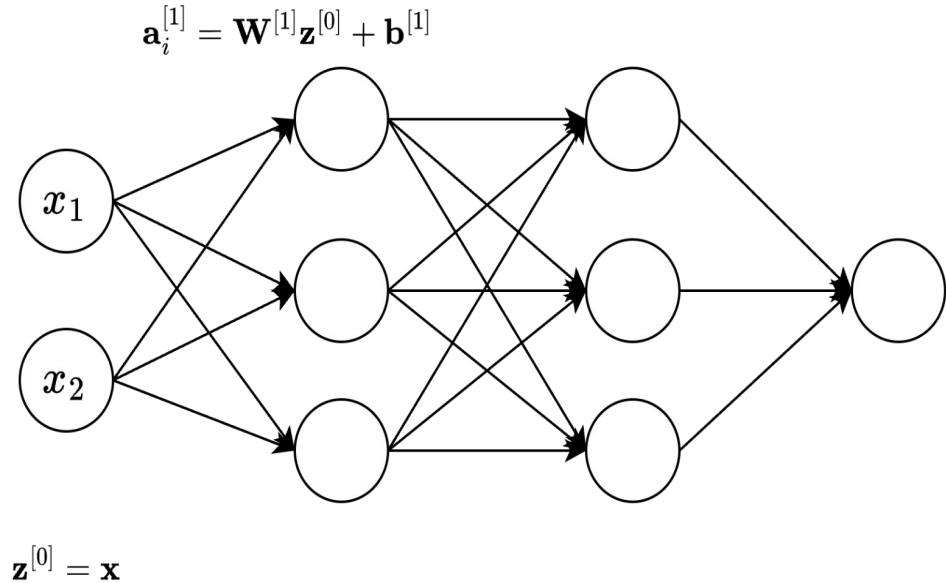
$\mathbf{W}^{[1]}$: (2,3) (3,1) (3,2)

$\mathbf{b}^{[1]}$: (3,2) (3,1) (3,1)

A.

B.

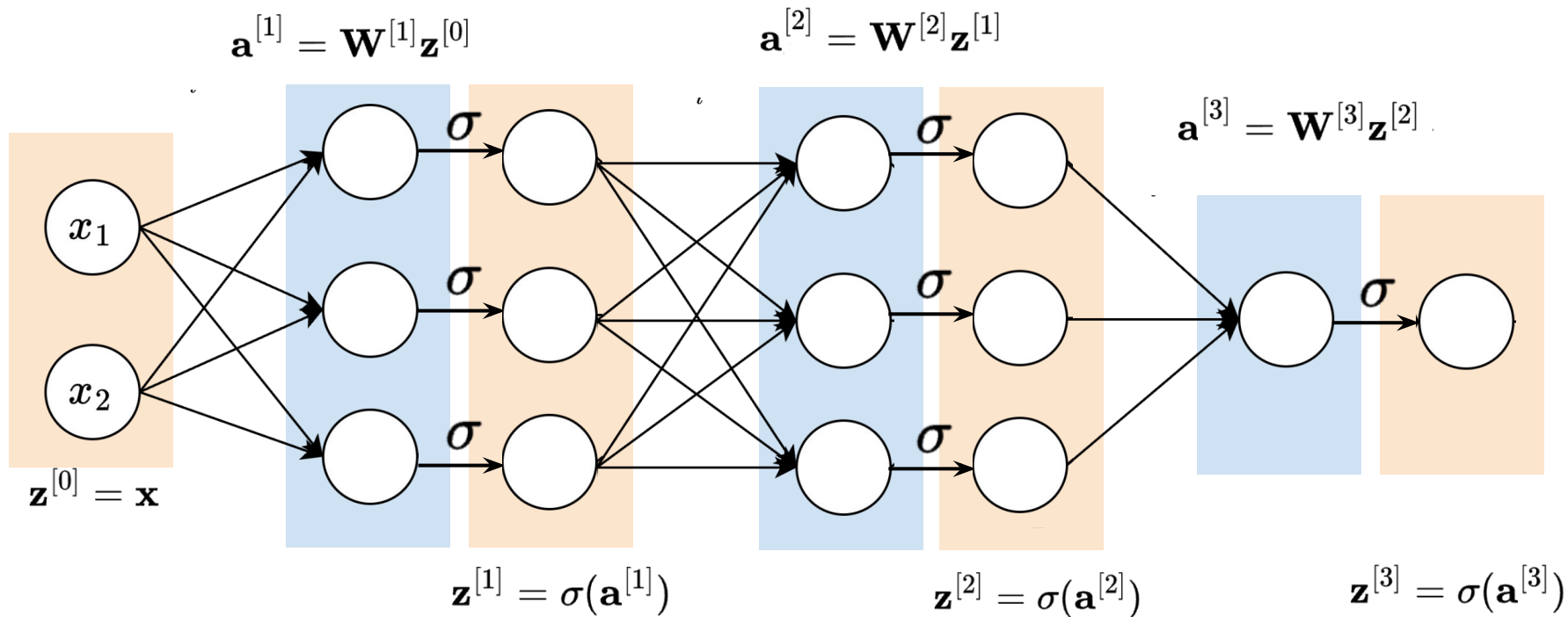
C.



Agenda

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

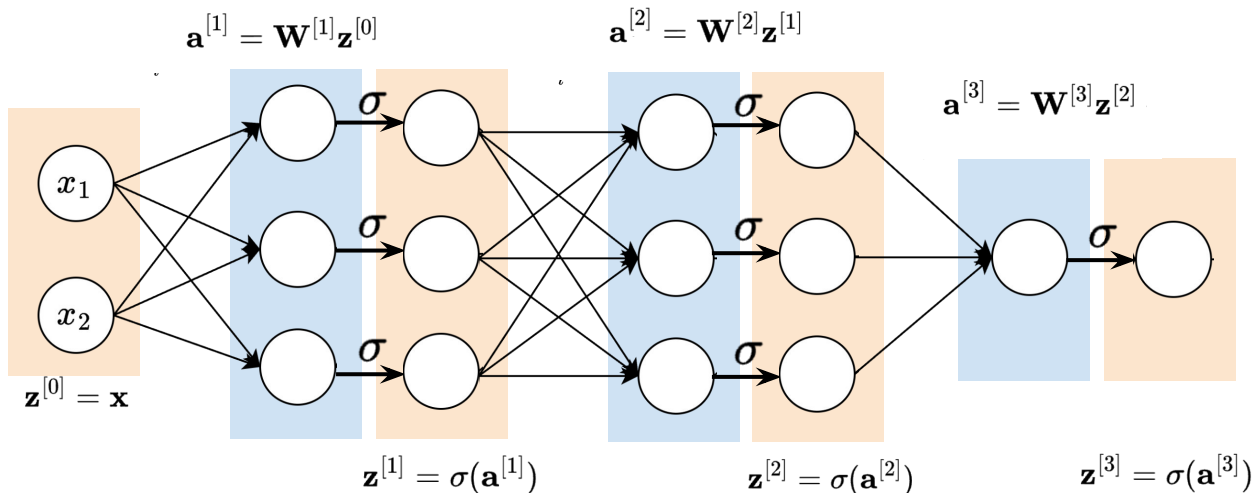
Forward Pass - MLP



Forward Pass - MLP

Algorithm Forward Pass through MLP

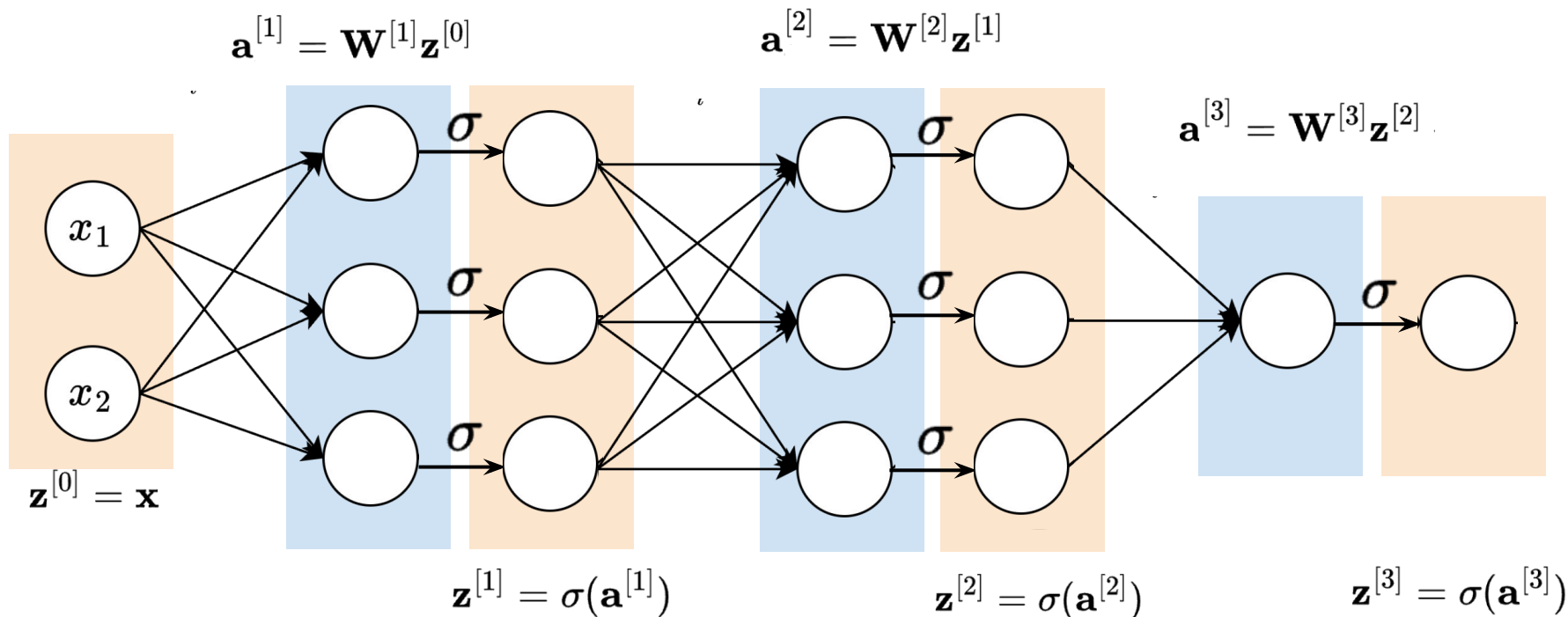
- 1: **Input:** input \mathbf{x} , weight matrices $\mathbf{W}^{[1]}, \dots, \mathbf{W}^{[L]}$, bias vectors $\mathbf{b}^{[1]}, \dots, \mathbf{b}^{[L]}$
- 2: $\mathbf{z}^{[0]} = \mathbf{x}$ ▷ Initialize input
- 3: **for** $l = 1$ **to** L **do**
- 4: $\mathbf{a}^{[l]} = \mathbf{W}^{[l]} \mathbf{z}^{[l-1]}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



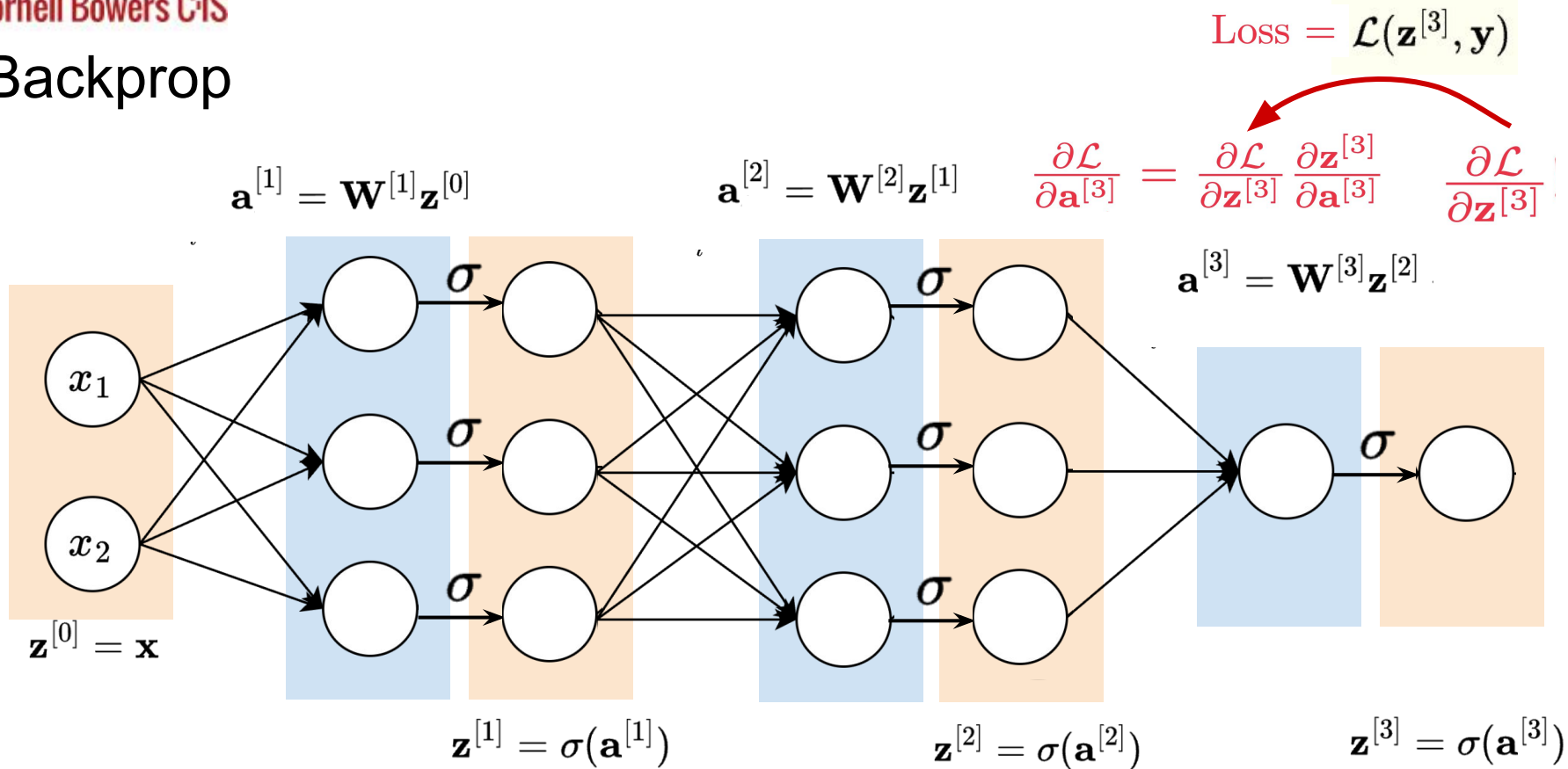
Backprop

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

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



Backprop



Backprop

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

For propagation to next layer:

$$\delta^{[3]} =$$

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

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

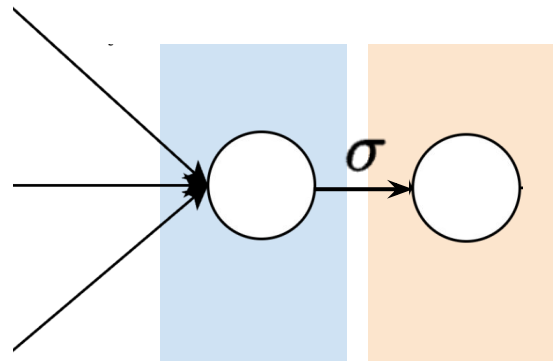
$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$$

For weight updates:

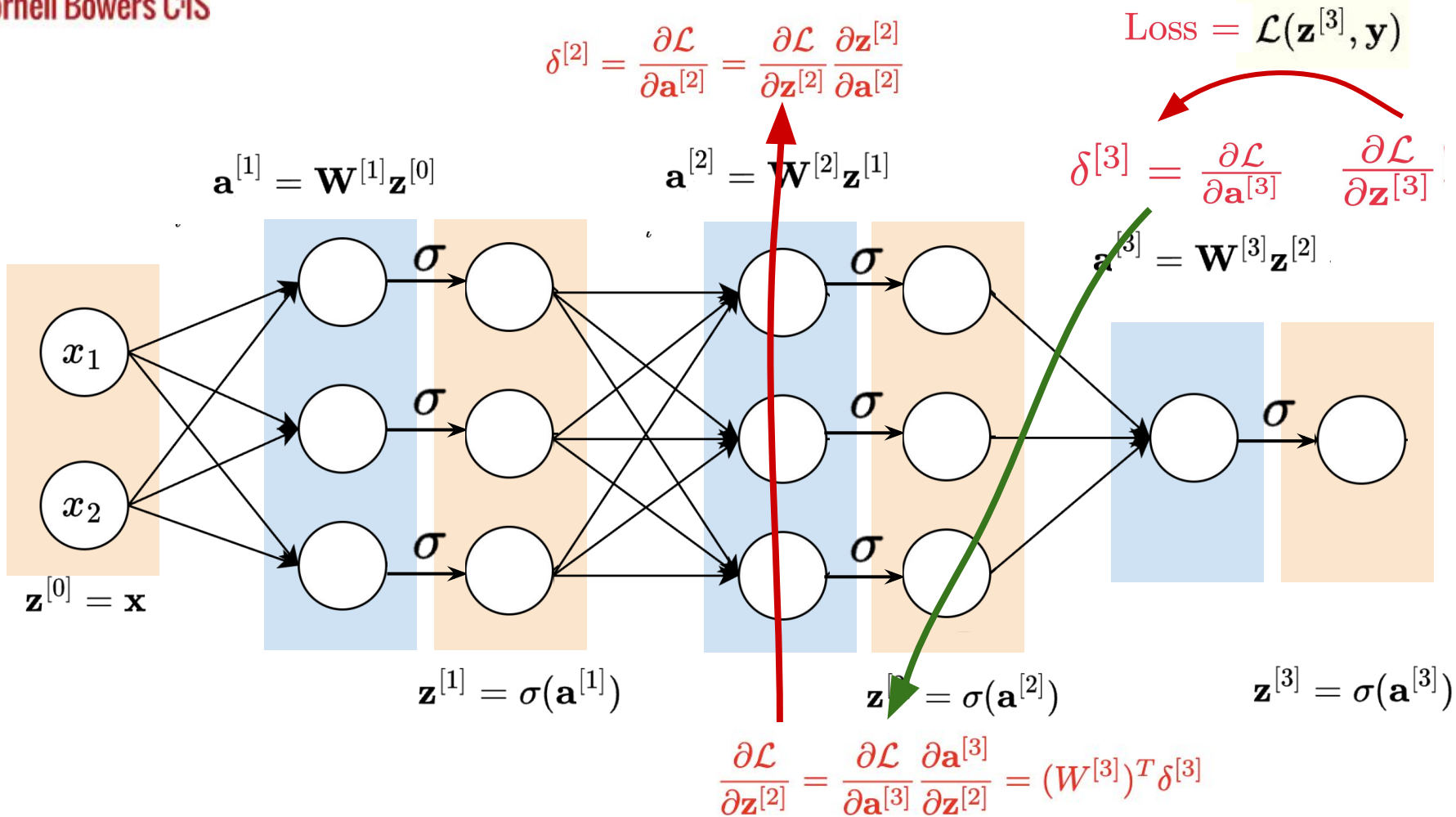
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{W}^{[3]}}$$

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

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



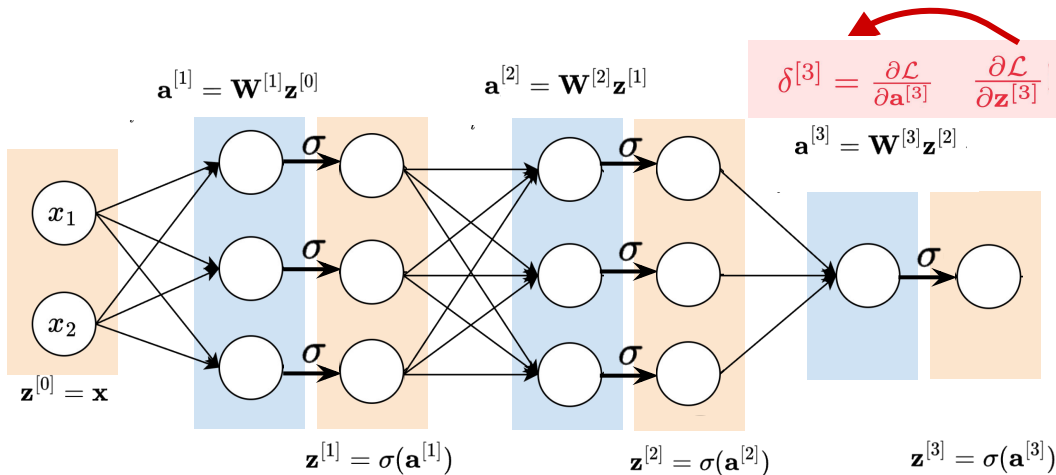
$$\mathbf{z}^{[3]} = \sigma(\mathbf{a}^{[3]})$$



Backpropagation

Algorithm Backward Pass through MLP (Detailed)

- 1: **Input:** $\{\mathbf{z}^{[1]}, \dots, \mathbf{z}^{[L]}\}, \{\mathbf{a}^{[1]}, \dots, \mathbf{a}^{[L]}\},$ 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:** $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$



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

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

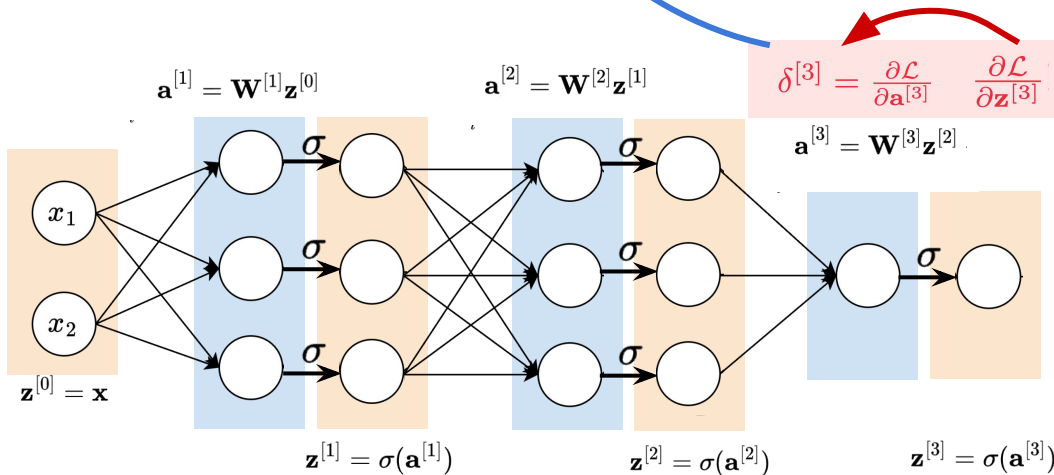
We can directly compute $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$!

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:** $\{\mathbf{z}^{[1]}, \dots, \mathbf{z}^{[L]}\}, \{\mathbf{a}^{[1]}, \dots, \mathbf{a}^{[L]}\},$ 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:** $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$



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

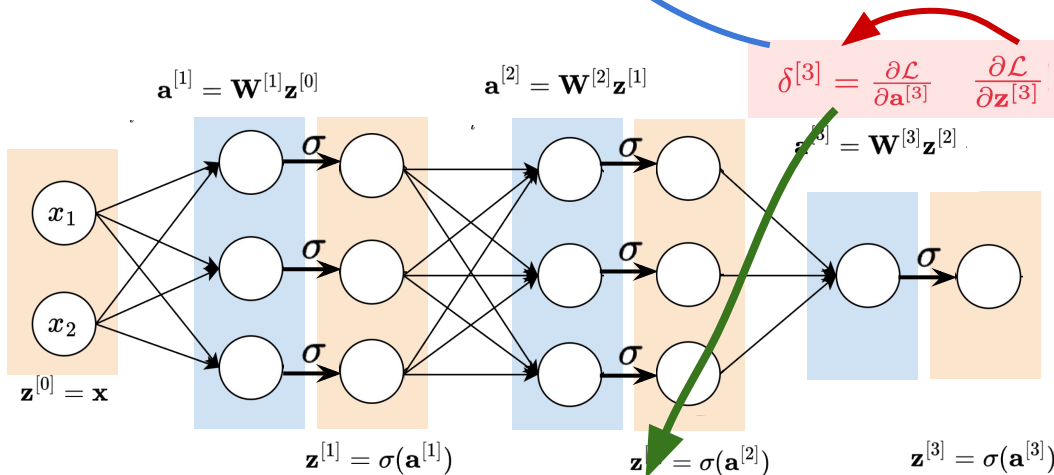
Backpropagation

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

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

Algorithm Backward Pass through MLP (Detailed)

- 1: **Input:** $\{\mathbf{z}^{[1]}, \dots, \mathbf{z}^{[L]}\}, \{\mathbf{a}^{[1]}, \dots, \mathbf{a}^{[L]}\},$ 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 $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$!

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

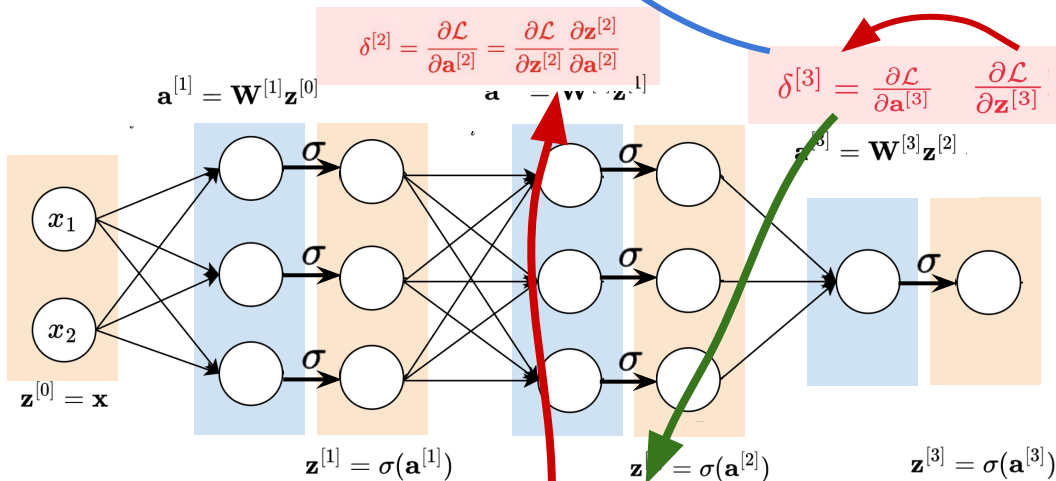
Backpropagation

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

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

Algorithm Backward Pass through MLP (Detailed)

- 1: **Input:** $\{\mathbf{z}^{[1]}, \dots, \mathbf{z}^{[L]}\}, \{\mathbf{a}^{[1]}, \dots, \mathbf{a}^{[L]}\},$ 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:** $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$

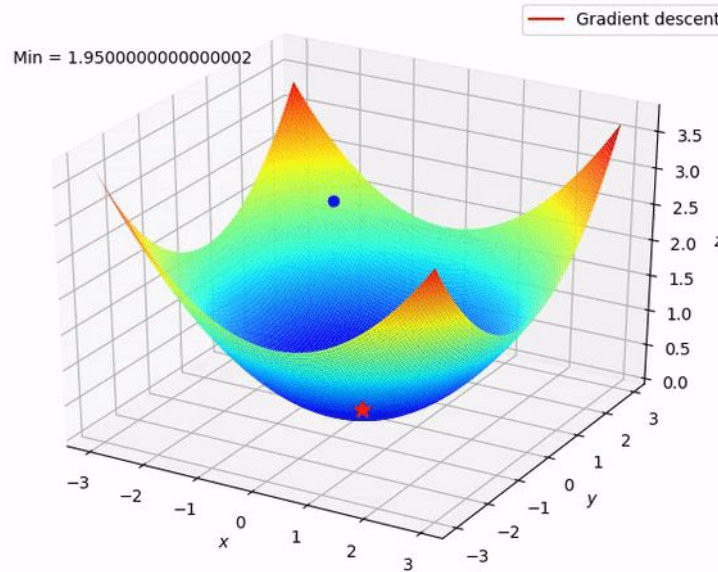


$$\mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

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

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

What is Optimization?



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

Loss function

Empirical Risk:

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

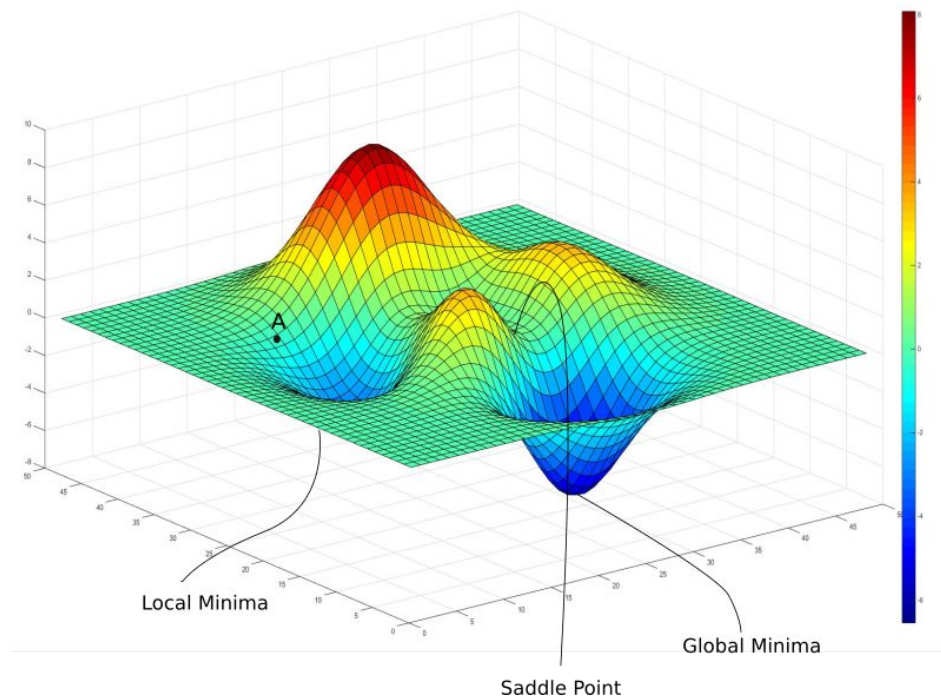
t : at time step t

\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

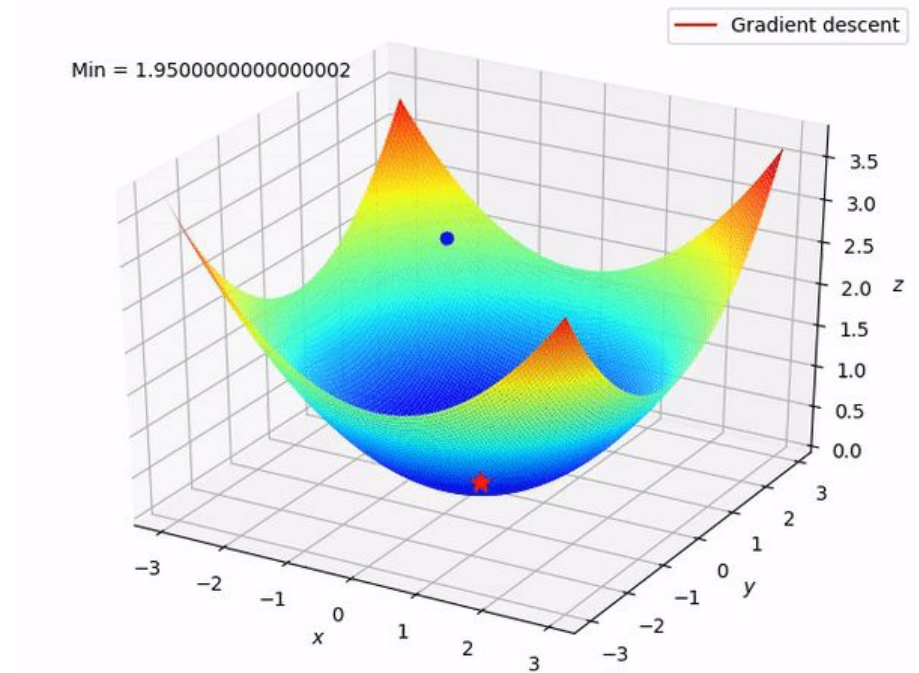


Gradient Descent (GD)

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

α : the learning rate

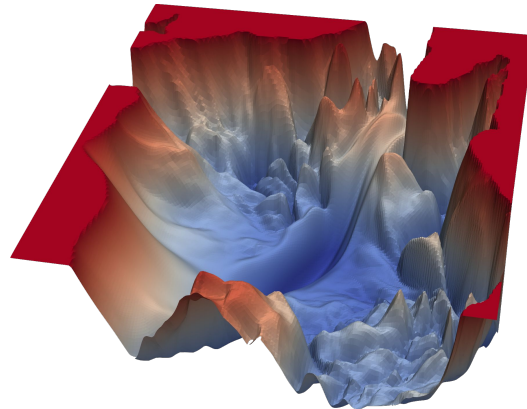
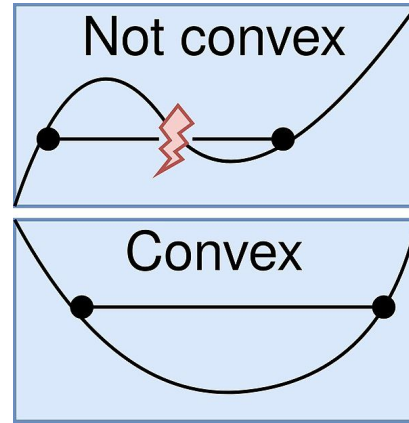
$\nabla \mathcal{L}(\mathbf{w}_t)$: the gradient of Loss w.r.t. \mathbf{w}_t



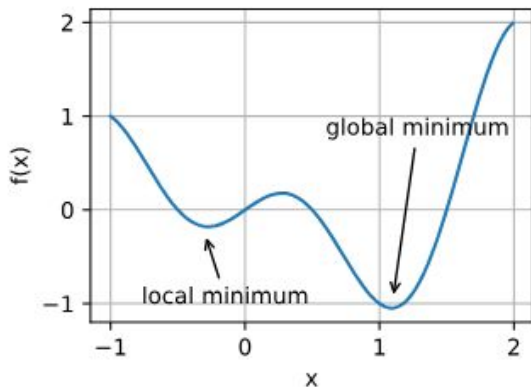
What are some potential problems with gradient descent?

Convexity

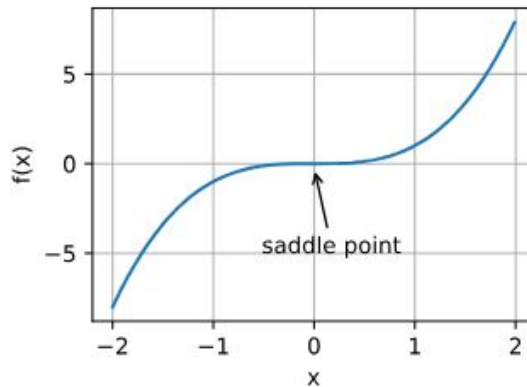
- A function on a graph is **convex** if a line segment drawn through any two points on the line of the function, then it never lies below the curved line segment
- Convexity implies that every local minimum is **global minimum**.
- Neural networks are **not** convex!



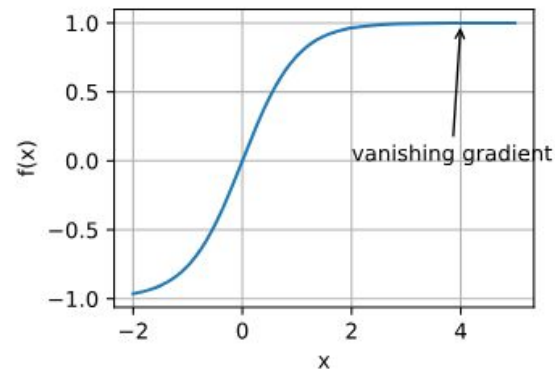
Challenges in Non-Convex Optimization



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)

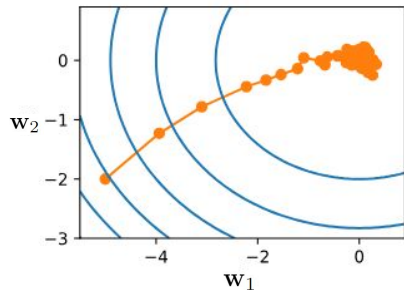
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

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

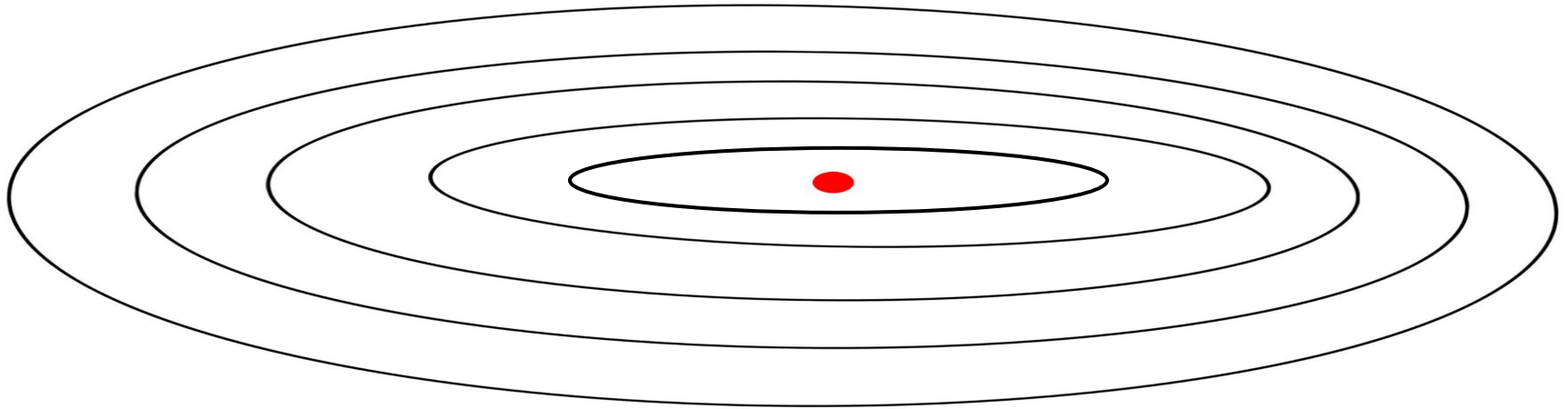
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

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

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

Let's look at an example!

Draw the gradients



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 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

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



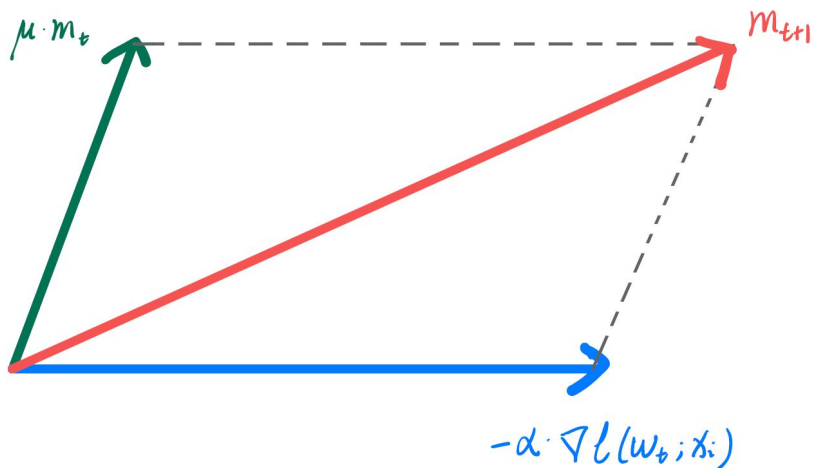
$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

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

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.

$$\mathbf{g}_t = \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \mathbf{g}_t$$

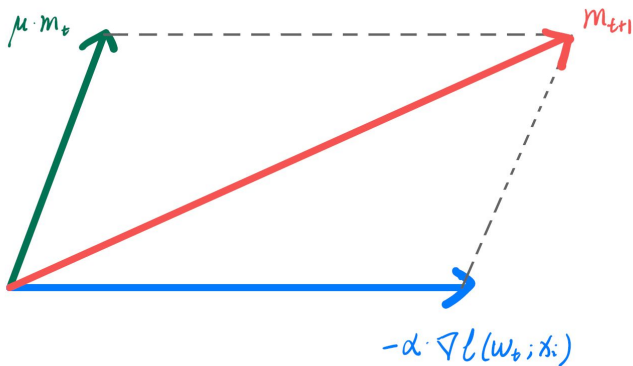
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mu \mathbf{m}_t - \alpha \mathbf{g}_t$$

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

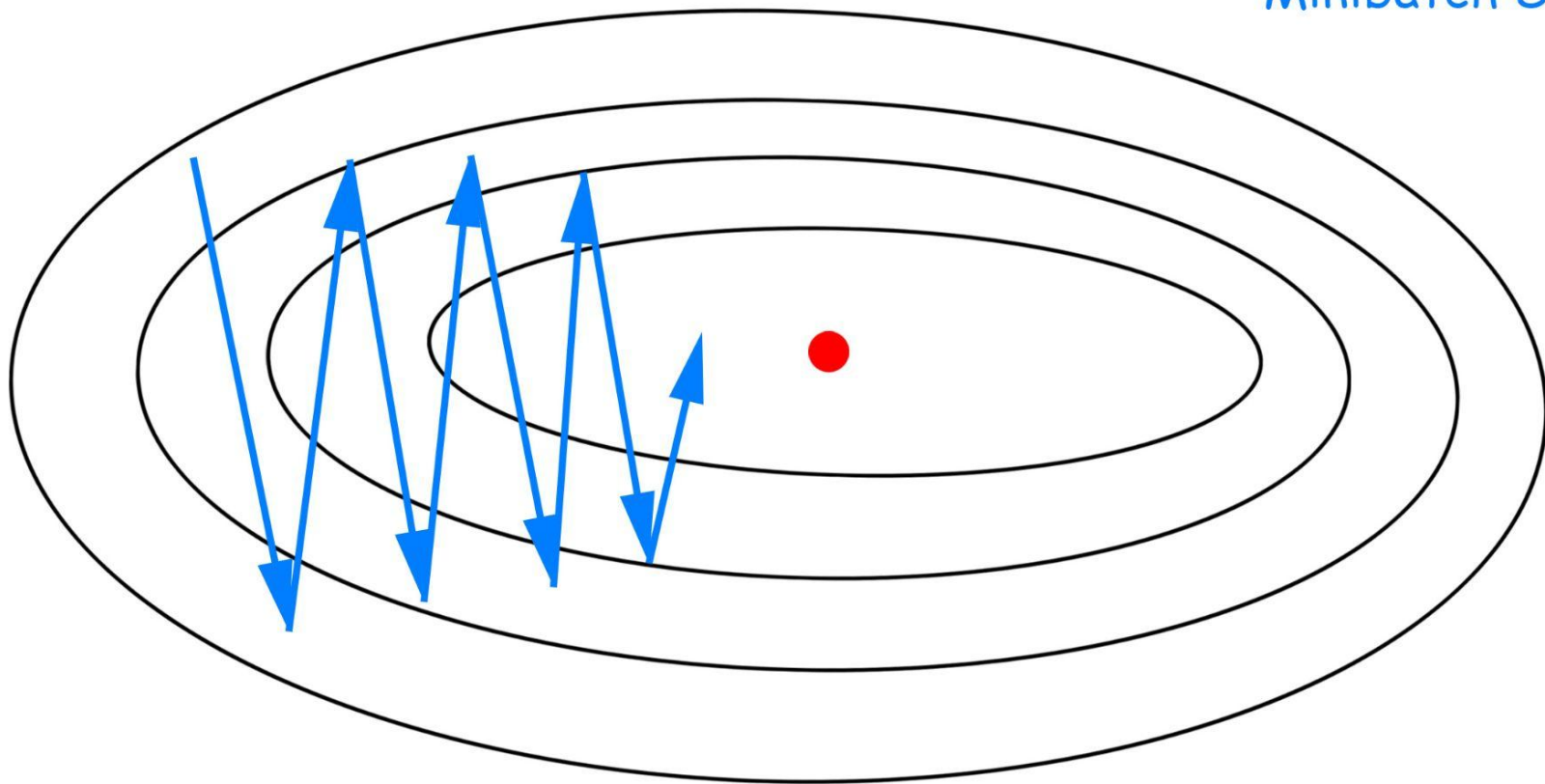
$$= \mathbf{w}_t + \mu(\mu(\mu \mathbf{m}_{t-2} - \alpha \mathbf{g}_{t-2}) - \alpha \mathbf{g}_{t-1}) - \alpha \mathbf{g}_t$$

$$= \mathbf{w}_t - \alpha \mathbf{g}_t - \mu \alpha \mathbf{g}_{t-1} - \mu^2 \alpha \mathbf{g}_{t-2} - \mu^3 \alpha \mathbf{g}_{t-3} - \dots$$

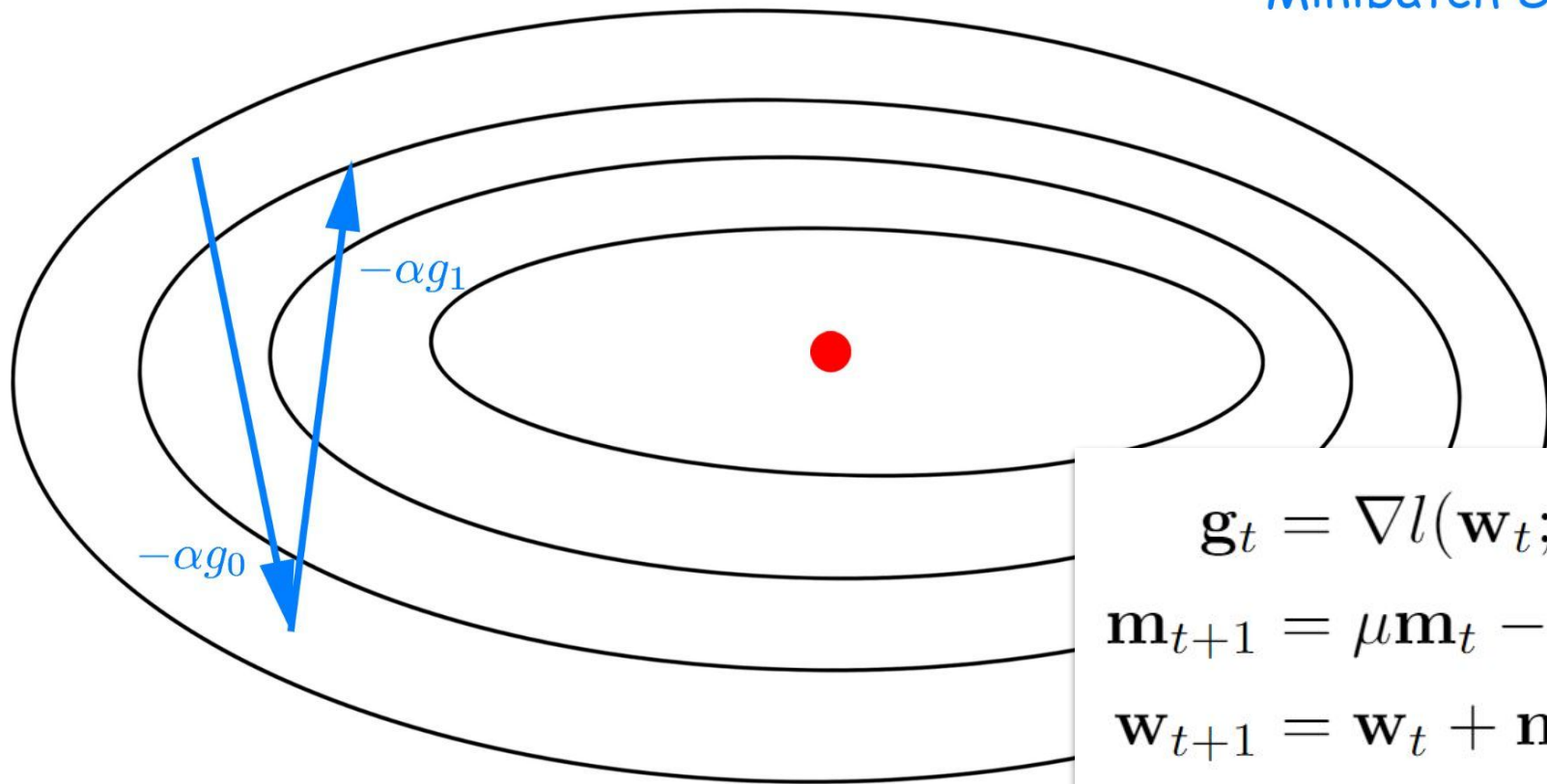
$$= \mathbf{w}_t - \alpha \sum_{i=0}^t \mu^i \mathbf{g}_{t-i}$$



Local Minimum
Minibatch SGD



Local Minimum
Minibatch SGD



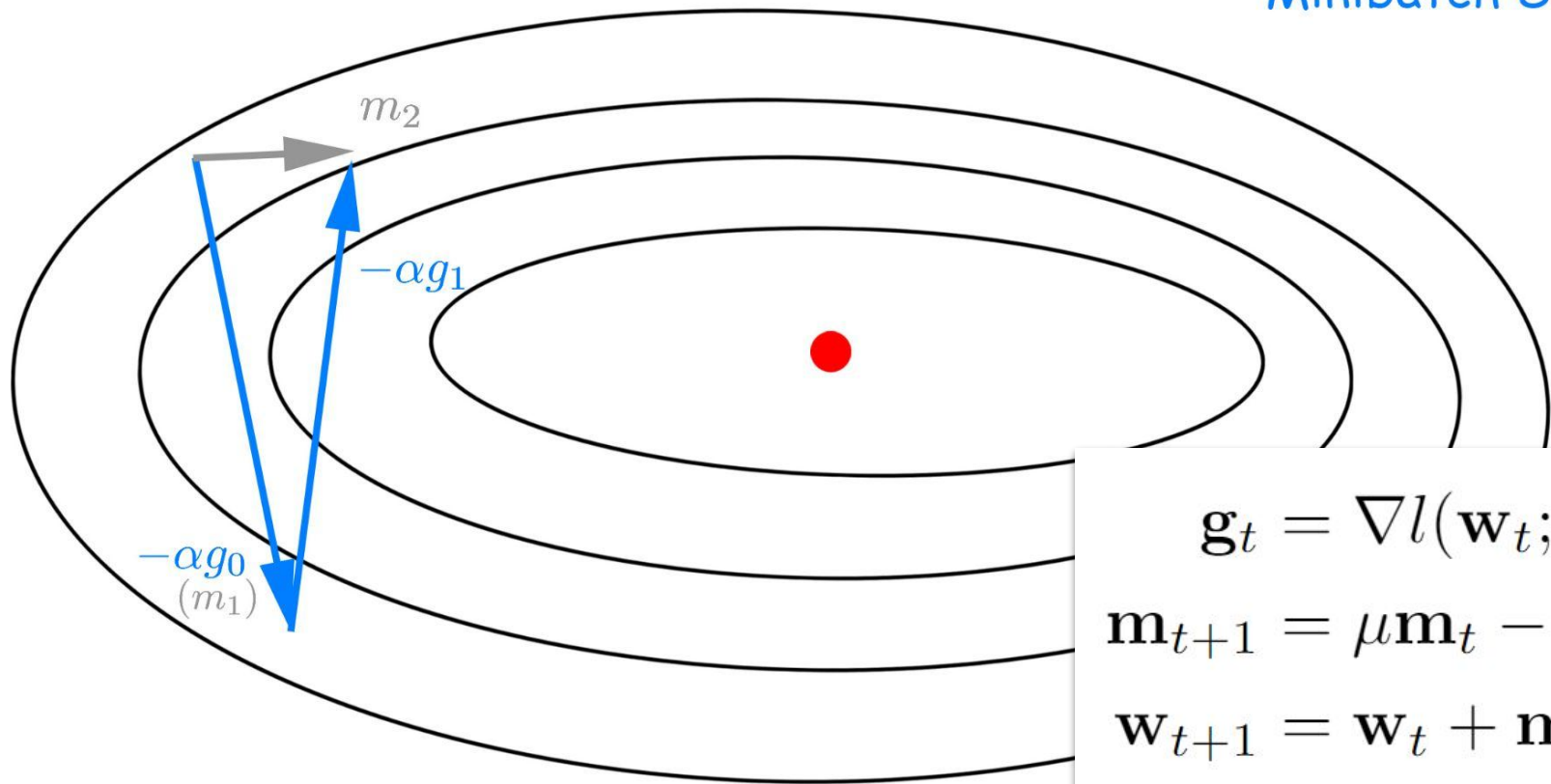
$$\mathbf{g}_t = \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \mathbf{g}_t$$

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

$$\mathbf{m}_2 = \mu \mathbf{m}_1 - \alpha \mathbf{g}_1$$

Local Minimum
Minibatch SGD



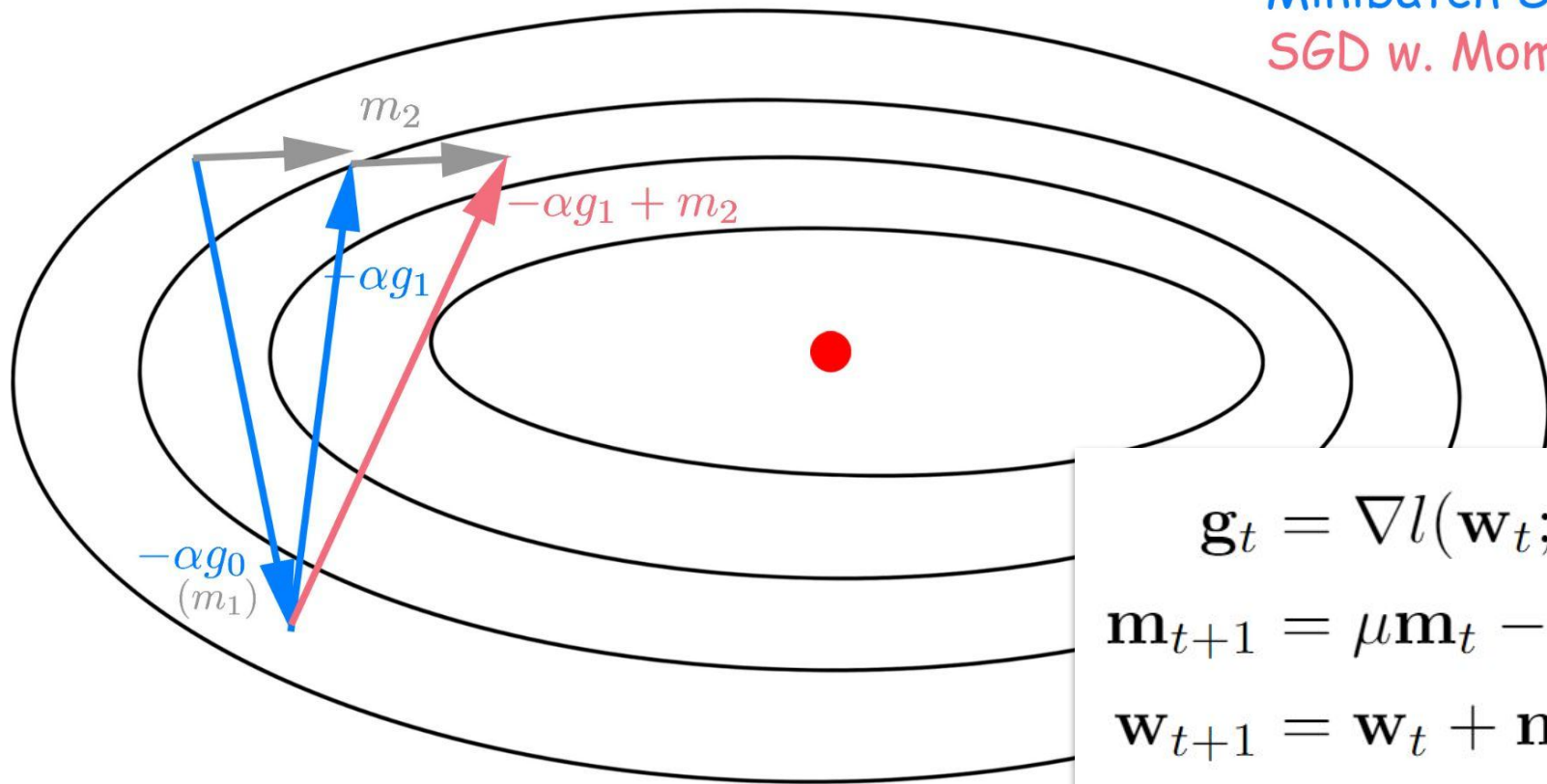
$$\mathbf{g}_t = \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \mathbf{g}_t$$

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

$$\mathbf{m}_2 = \mu\mathbf{m}_1 - \alpha\mathbf{g}_1$$

Local Minimum
Minibatch SGD
SGD w. Momentum

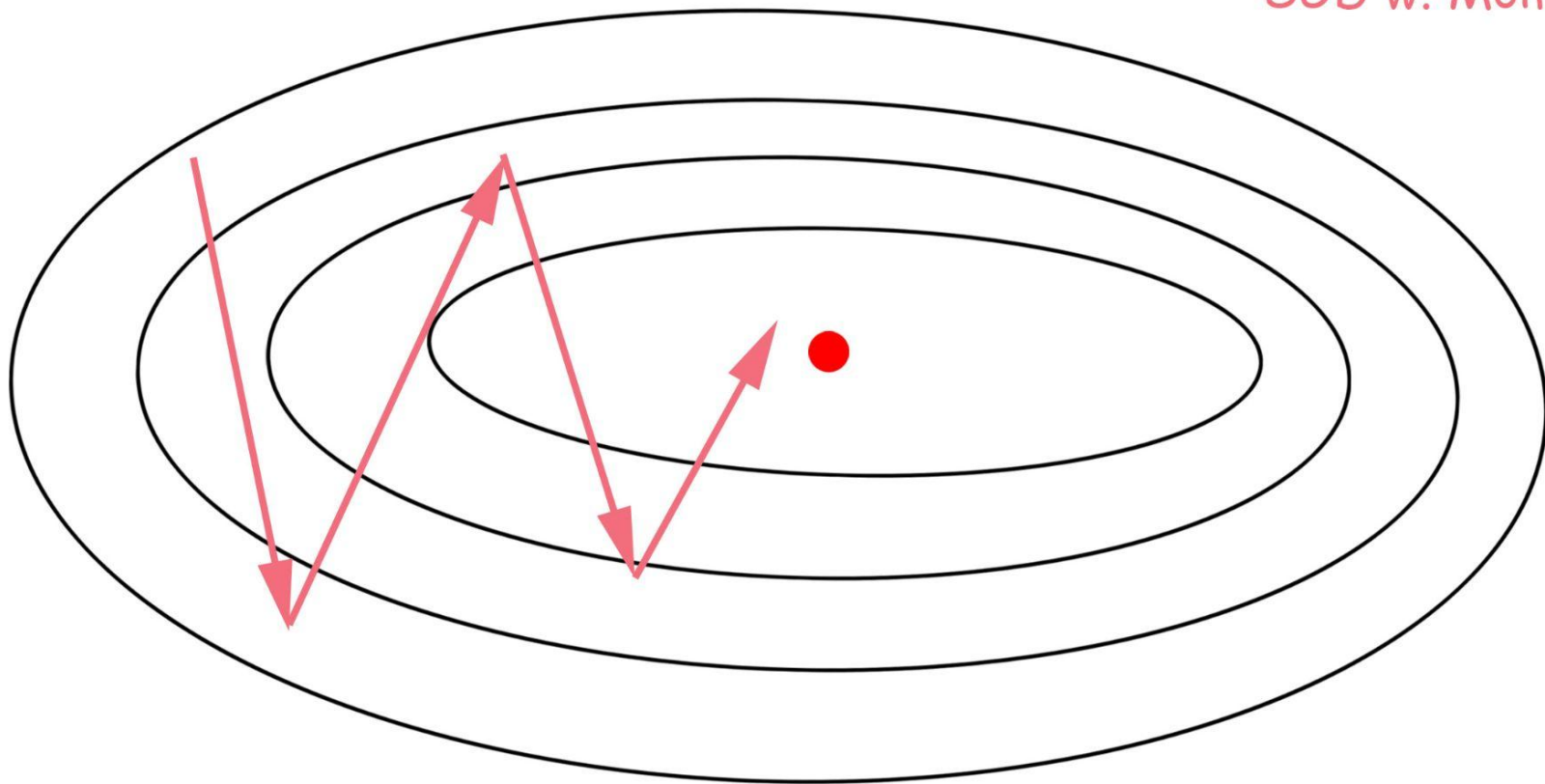


$$\mathbf{g}_t = \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

$$\mathbf{m}_{t+1} = \mu\mathbf{m}_t - \alpha\mathbf{g}_t$$

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

Local Minimum
SGD w. Momentum



Local Minimum
SGD w. Momentum



***Momentum converges almost always faster than standard
SGD!***

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

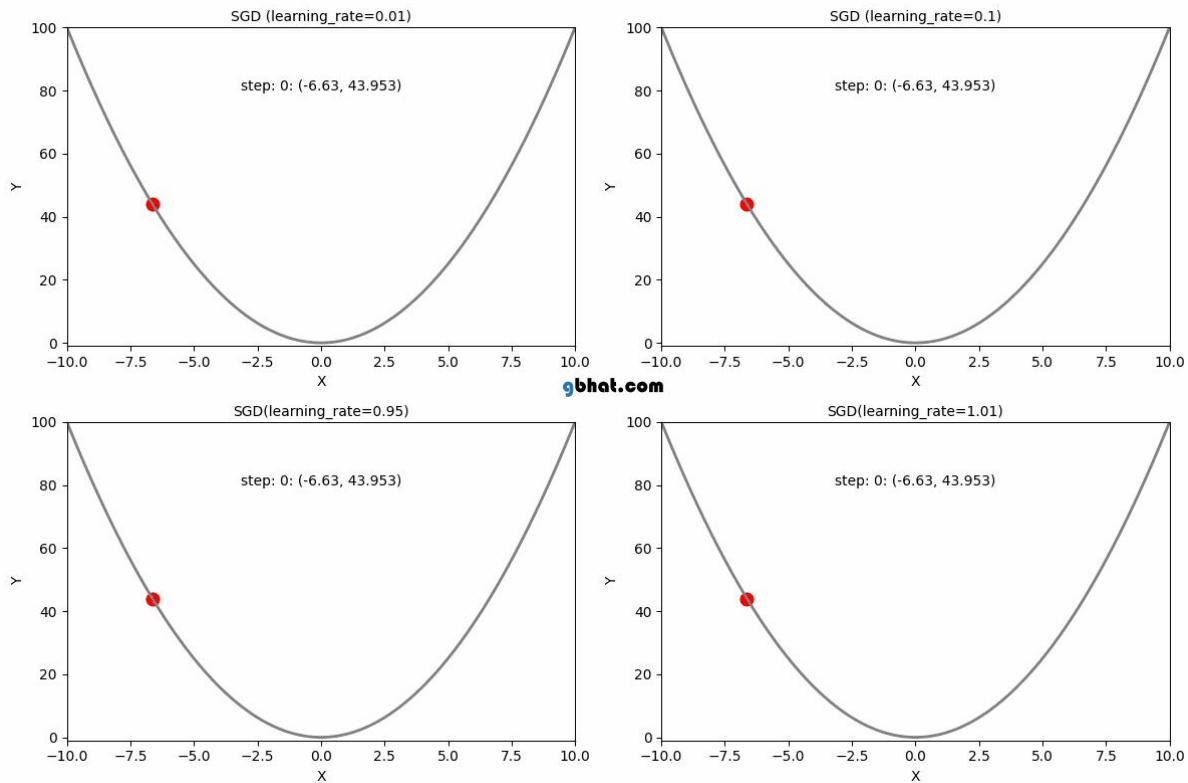
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

SGD w. Momentum

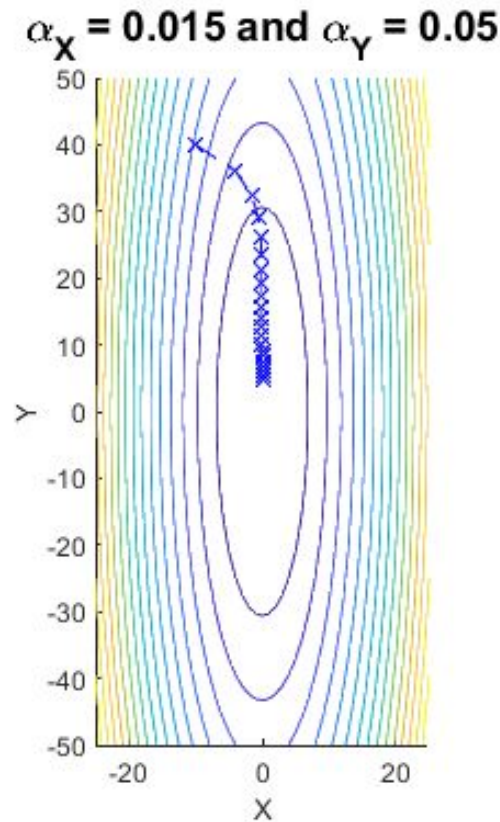
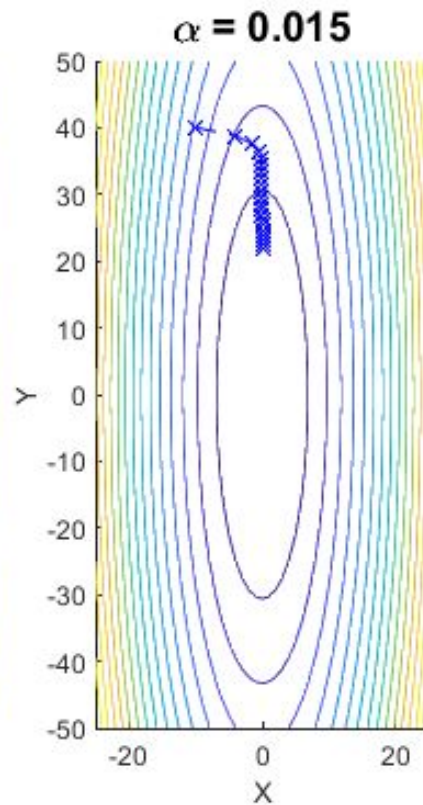
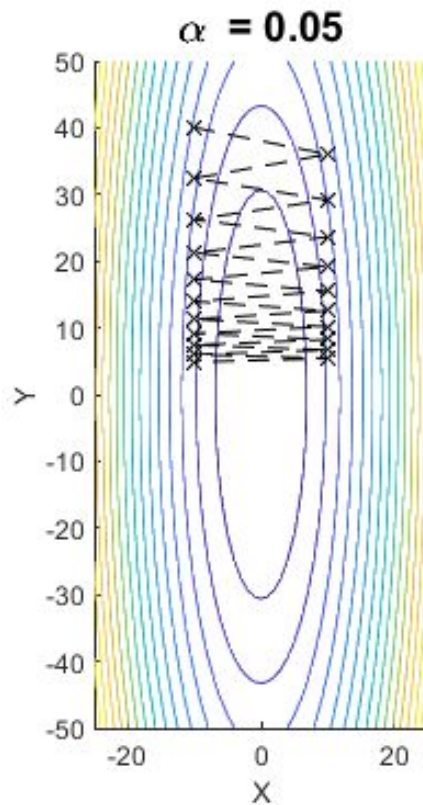
$$m_{t+1} = \mu m_t - \alpha \nabla \ell(w_t; x_i)$$

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

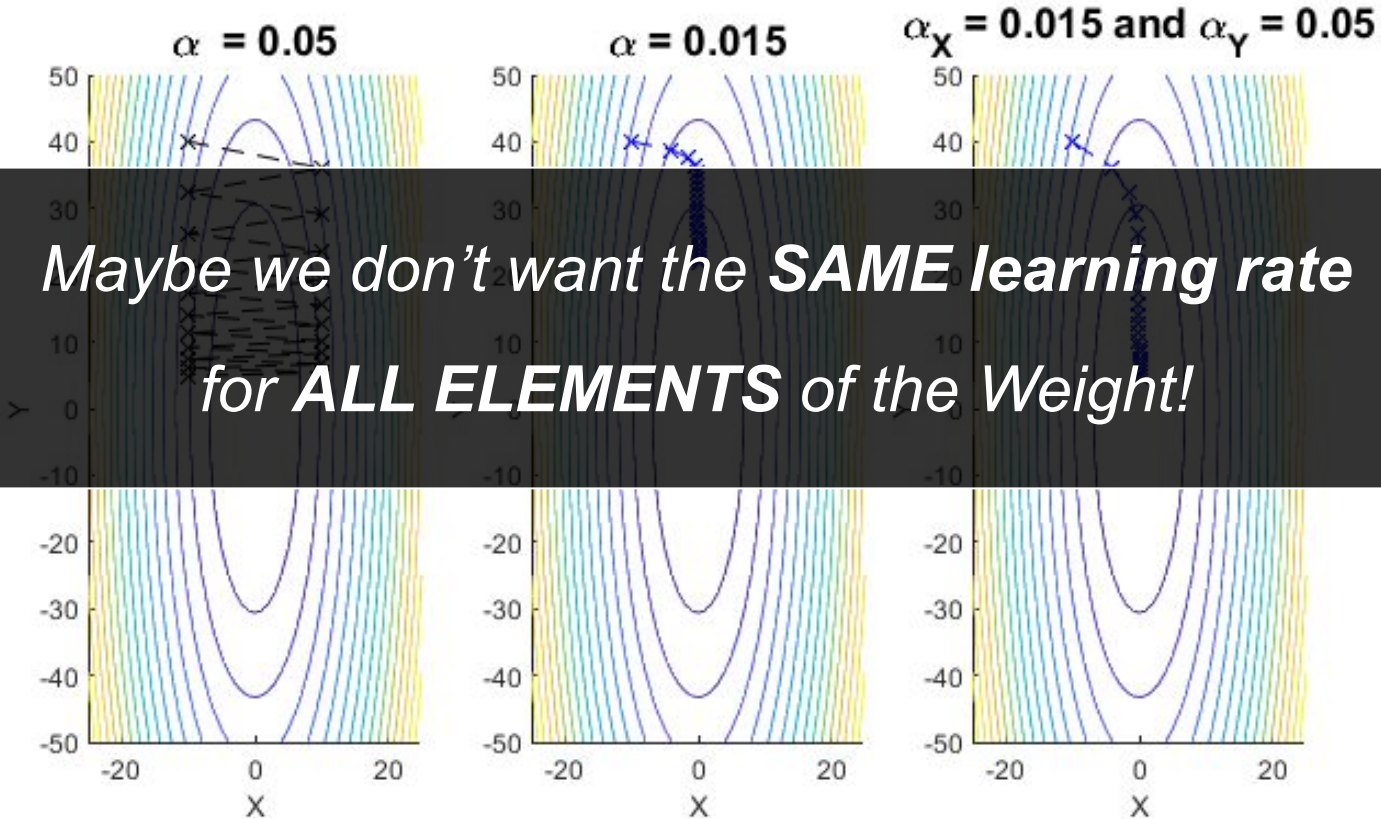
Importance of Learning Rate



Another example



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 -> more decay

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

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

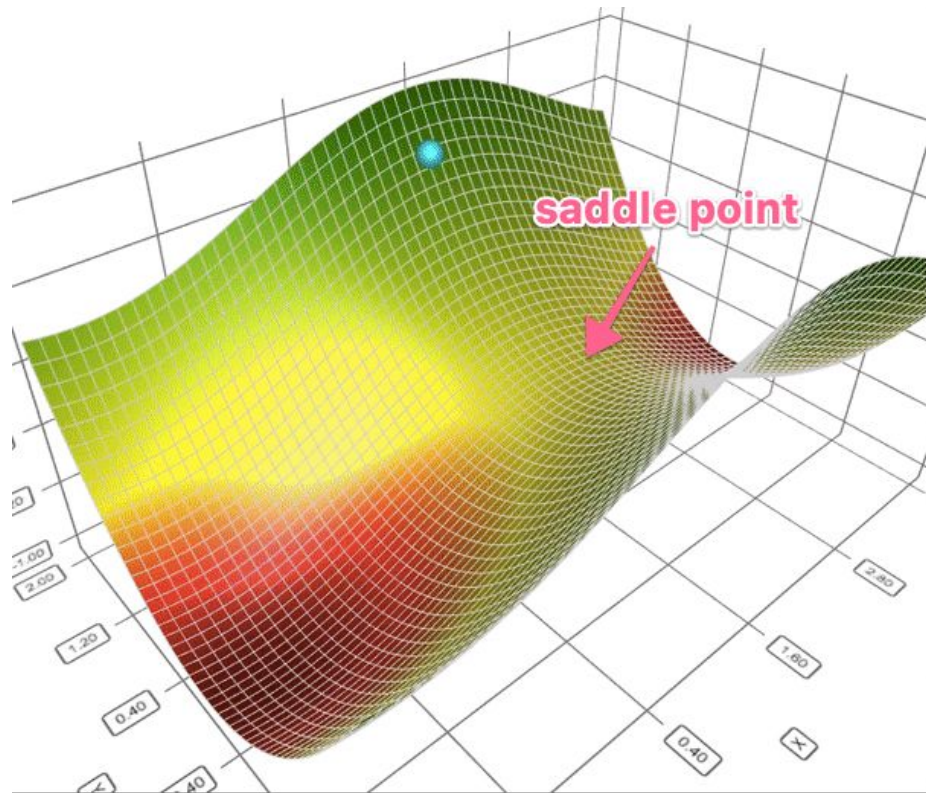
SGD

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

AdaGrad (Duchi et al. 2011)

More updates \rightarrow 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

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

Adagrad

Exercise:

*What's could be wrong with this optimizator?
(What would happen to the denominator.)*

AdaGrad (Duchi et al. 2011)

More updates \rightarrow 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

Issue: decays too aggressively!

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

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

ADAM

(**A**daptive **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

(Adaptive Moment Estimate)

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

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

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

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

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

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

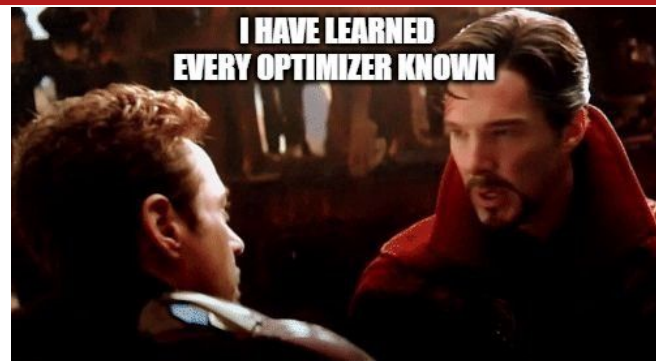
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_{t+1} + \epsilon}} \odot \hat{\mathbf{m}}_{t+1}$$

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!

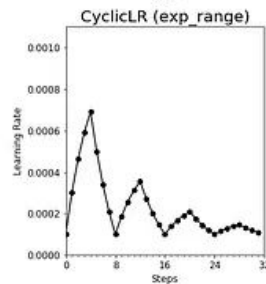
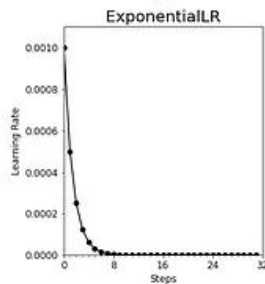
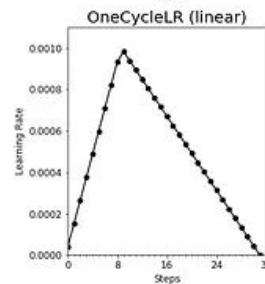
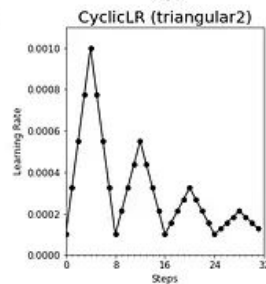
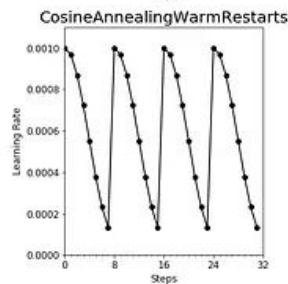
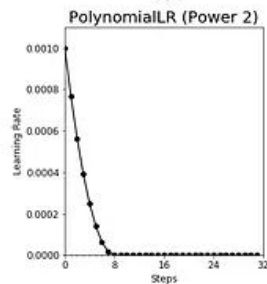
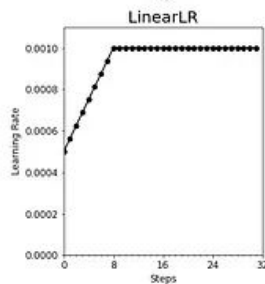
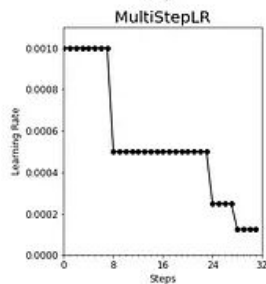
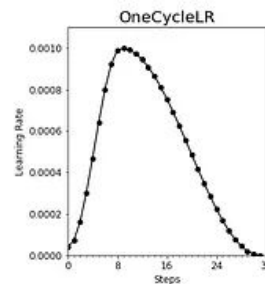
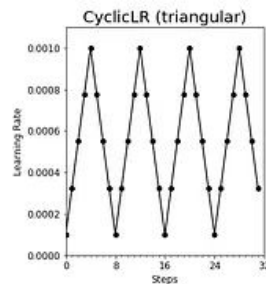
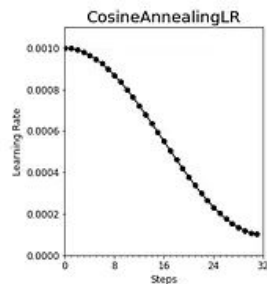
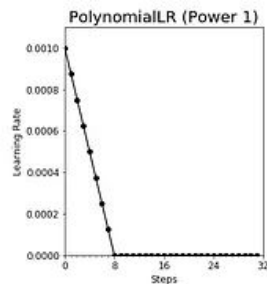
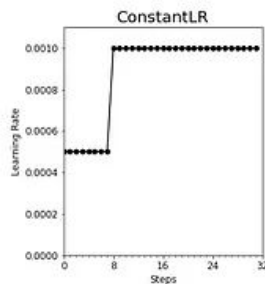
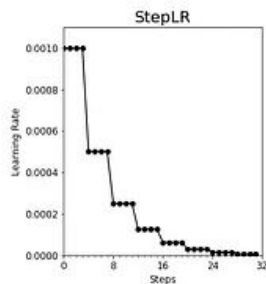
There are *many* minimizers of the training loss
The **optimizer** determines which minimizer you converge to



Agenda

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

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.

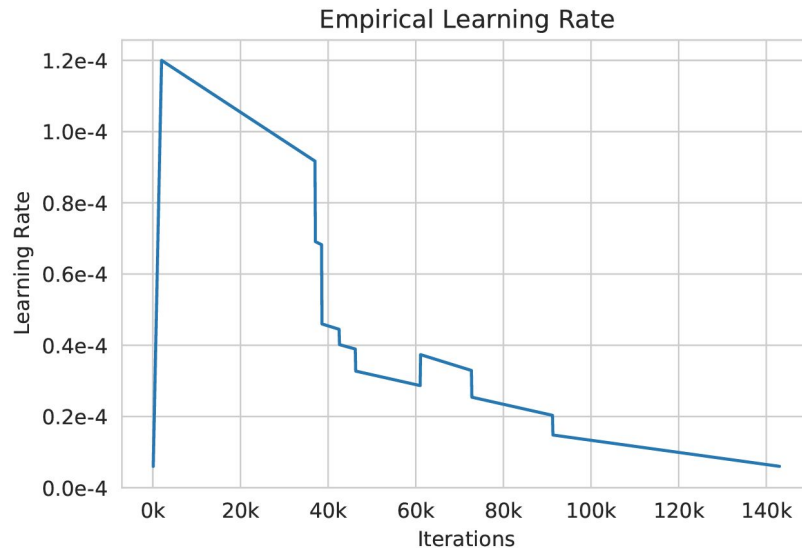
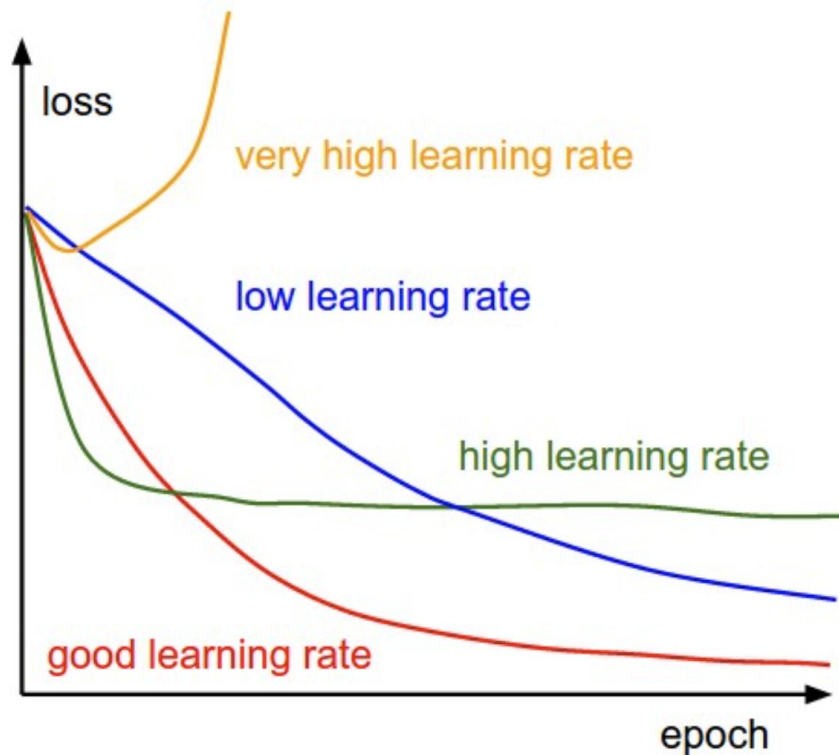


Figure 1: **Empirical LR schedule.** We found that lowering learning rate was helpful for avoiding instabilities.

Hyperparameters

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

These are all hyperparameters that affect performance!

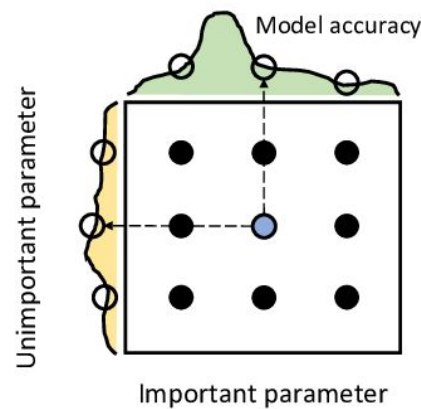


Hyperparameter Optimization (HPO)

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

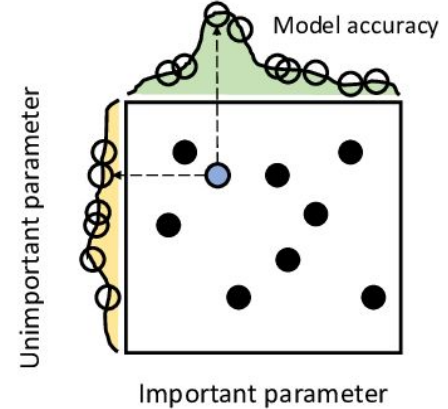
These are all hyperparameters that affects performance!

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



(a)

Grid Search



(b)

Random Search

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!*