Deep Learning

Day 4: Backprop

CSCI 1470

Eric Ewing

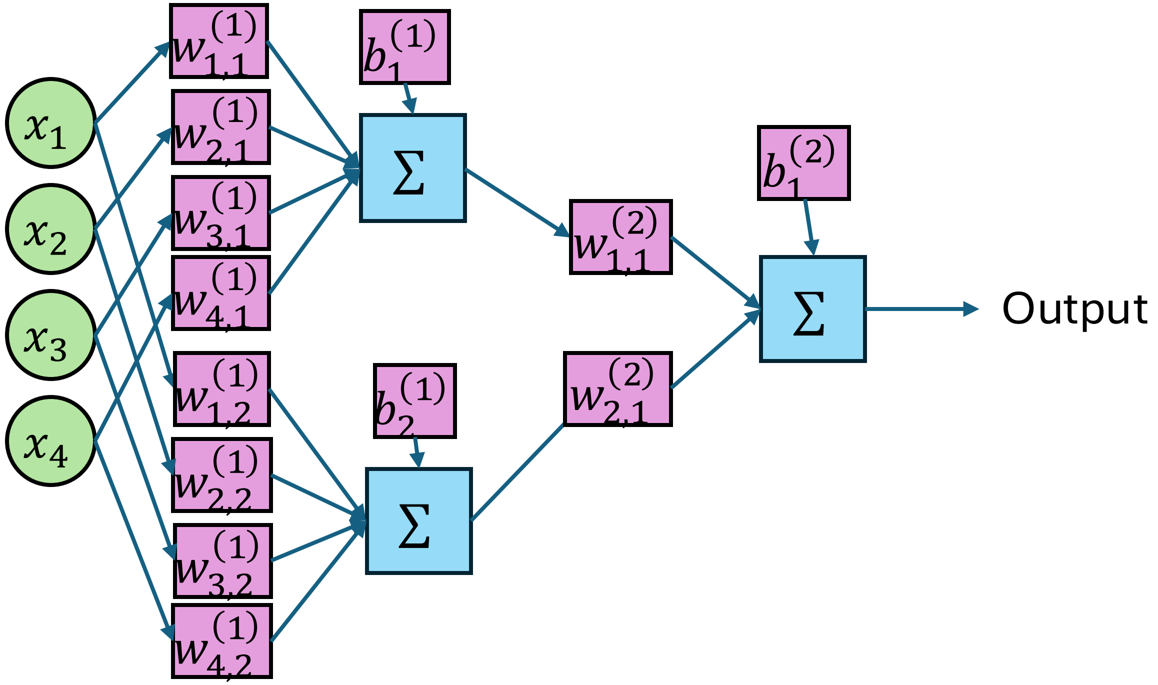
Tuesday, 9/16

The Grand Cayon

Review: Multi-Layer Perceptrons

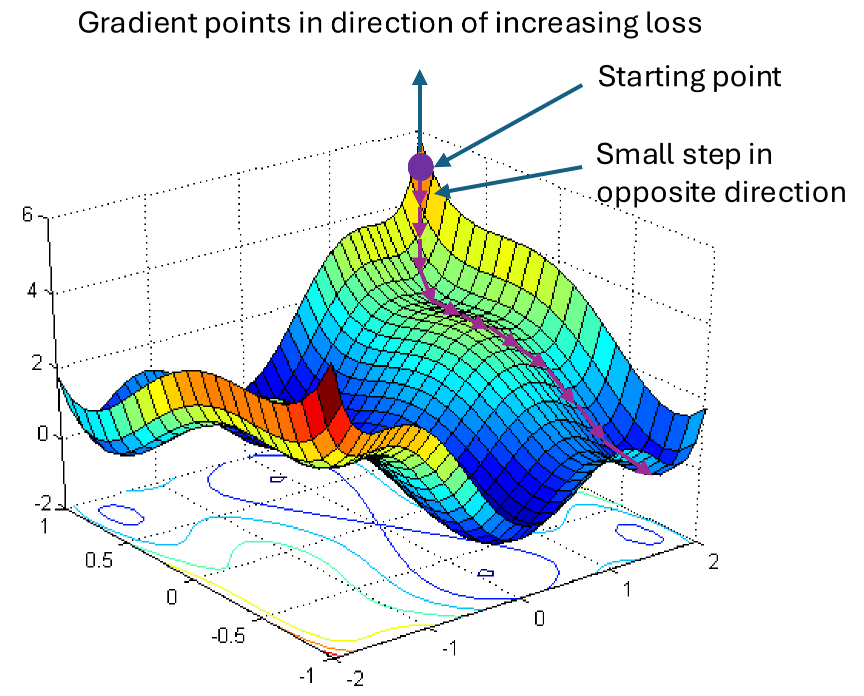
Perceptrons are linear classifiers, separating classes based on input features with a linear separator

Multi-layer perceptrons learn input features to perceptrons to represent more complex functions



Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



Vector Calculus

- Partial Derivative: the derivative of a multi-variable function with respect to one of its inputs
- Example: $f(x, w, b) = wx + b$
- The partial derivative with respect to w is $\frac{\partial f}{\partial w}$
- How to compute: Treat all other variables as constants and differentiate with respect to that variable

$$\frac{\partial f}{\partial w} = \frac{\partial}{\partial w}(wx + b) = \frac{\partial}{\partial w}(wx) + \frac{\partial}{\partial w}(b) = x$$

Gradients

Gradient: the vector of partial derivatives

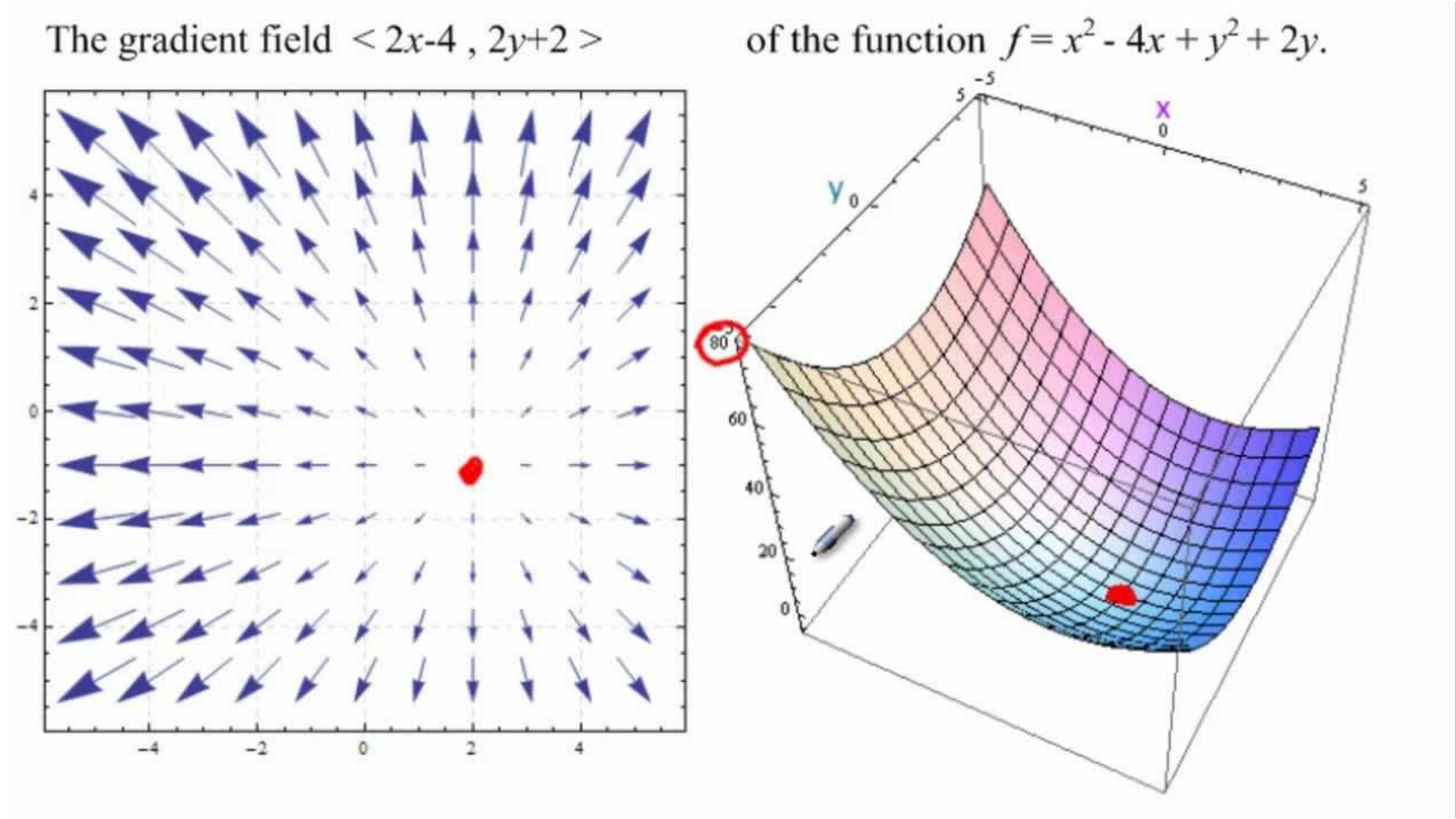
Vector "points" in direction of increasing f values.

$$\nabla f = [\frac{\partial f}{\partial w}, \frac{\partial f}{\partial b}, \dots]$$

$$f(x, w, b) = wx + b$$

$$\nabla f_\theta = [\frac{\partial f}{\partial w}, \frac{\partial f}{\partial b}, \frac{\partial f}{\partial x}]$$

Gradients



The gradient field < 2x-4 , 2y+2 >
of the function f = x² - 4x + y² + 2y.
x
y
80
60
40
20
0

Jacobians

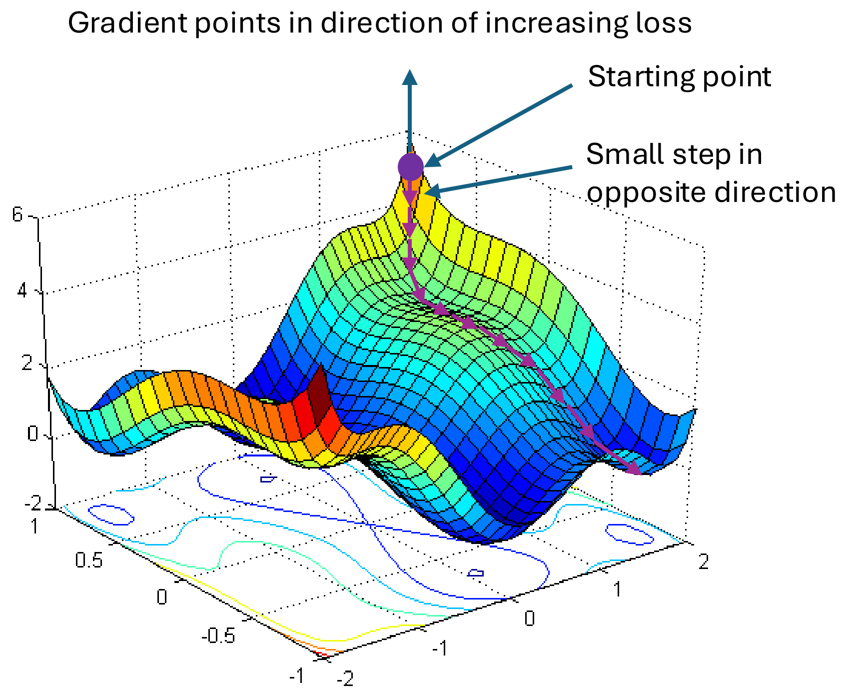
- Gradients are for functions with multiple inputs and one output
- Hidden layers in our neural networks have multiple outputs
- The Jacobian matrix is the matrix of all partial derivatives

$$f: \mathbb{R}^n \to \mathbb{R}^m$$

$$\begin{array}{cc}
 & \text{Input Variables: } [x_1, x_2, \dots, x_n] \\
\begin{array}{c}\text{Output Variables:} \\ [f_1, f_2, \dots f_m]\end{array} & \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}
\end{array}$$

Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

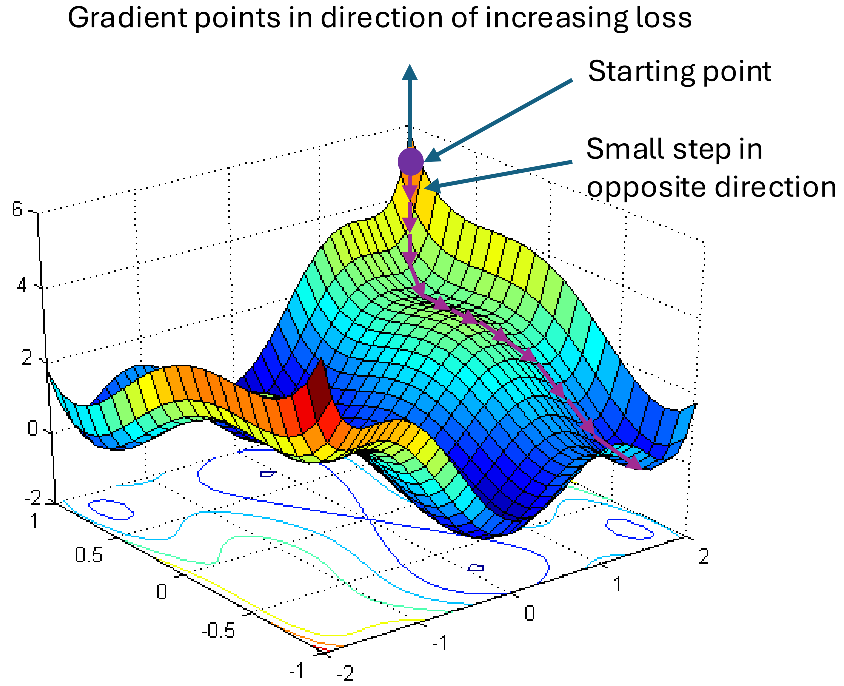
Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

For N iterations or until $\Delta\theta < \epsilon$:

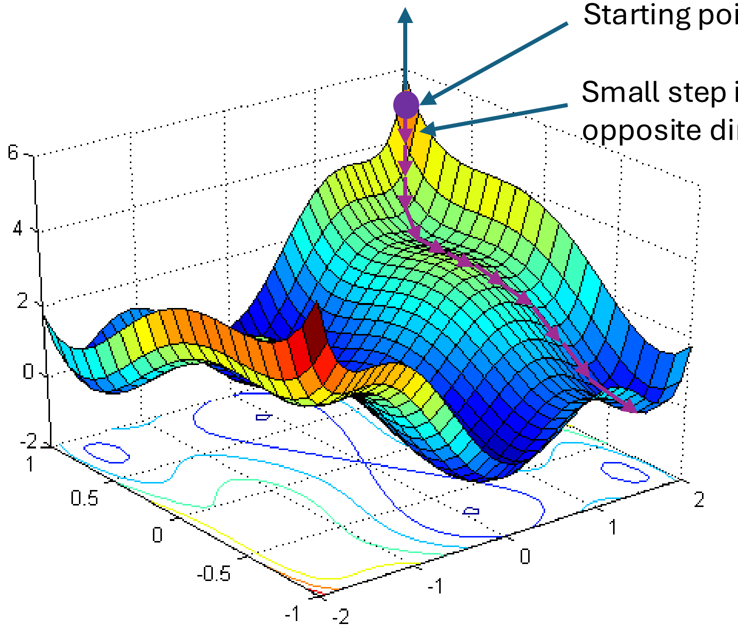
$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

Gradient of what?



Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_\theta$$

Gradient of what?

Why is this negative?

Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

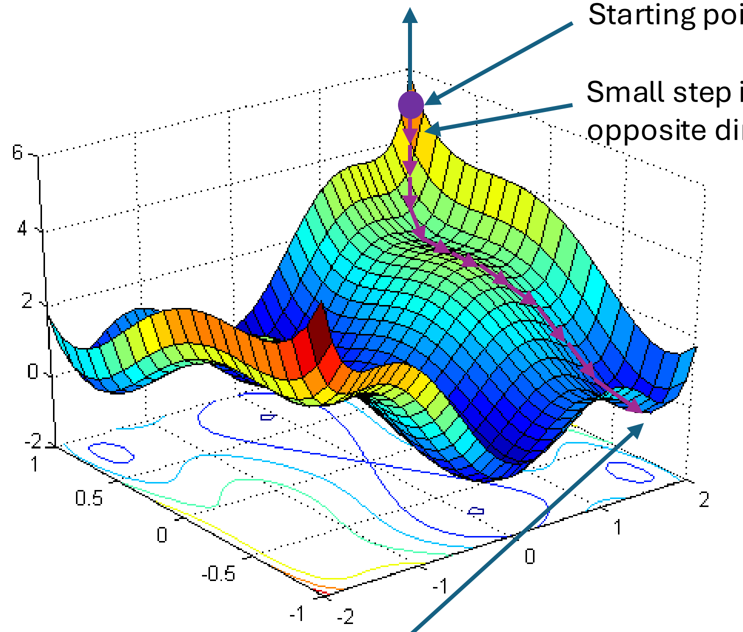
Why is this negative?

Gradient of what?

Gradient points in direction of increasing loss

Starting point

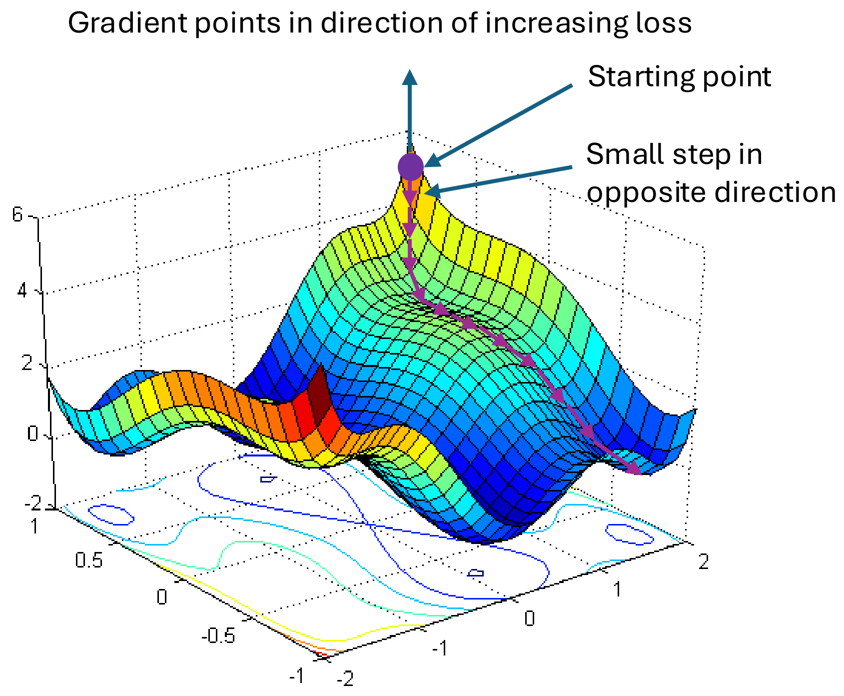
Small step in opposite direction



Wait, this isn't even the best θ

Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



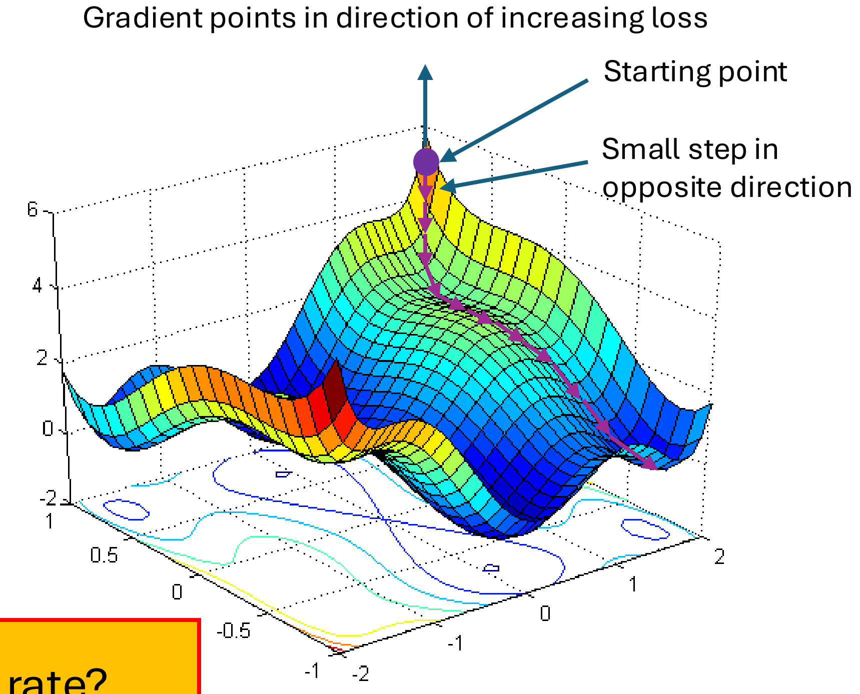
For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

Learning Rate $\alpha \in [0,1]$

Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



For N iterations or until $\Delta\theta < \epsilon$:

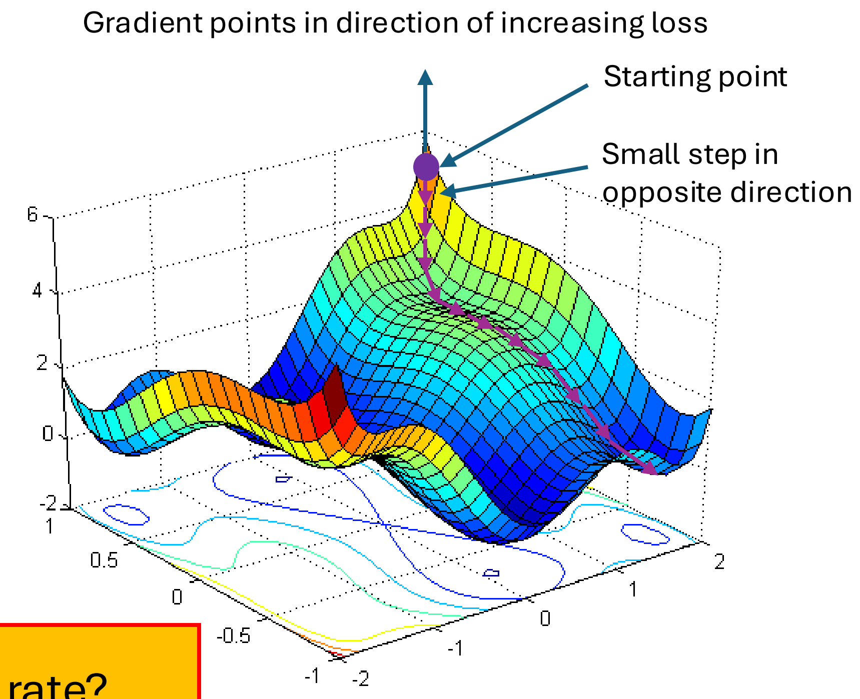
$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

Learning Rate $\alpha \in [0,1]$

Why do we need a learning rate?

Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

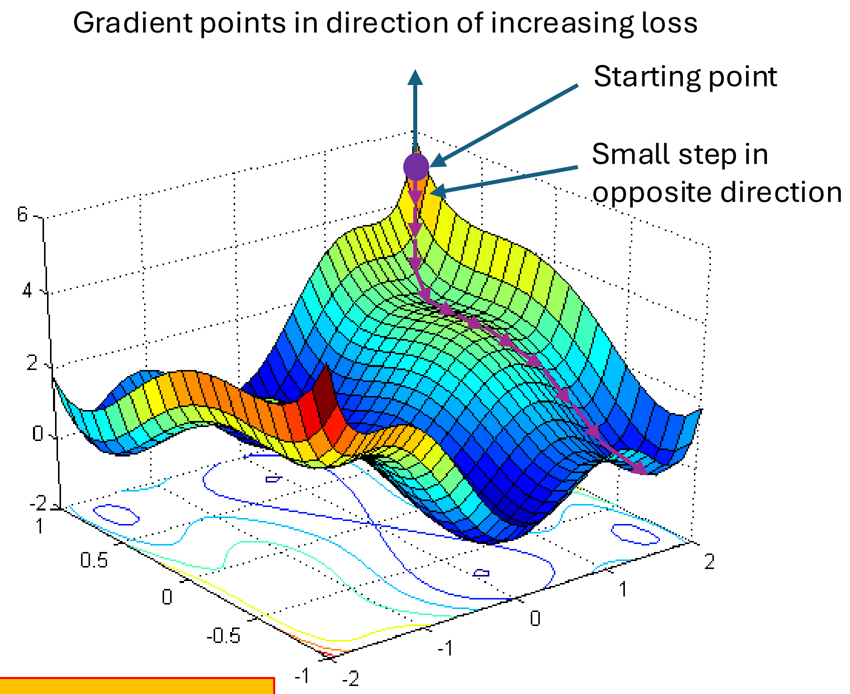
Learning Rate $\alpha \in [0,1]$

Why do we need a learning rate?

Derivatives/Gradients only hold locally

Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence



For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

Gradient Descent does not converge to the global minimum.
It can (and pretty much always does) get stuck in local minima.

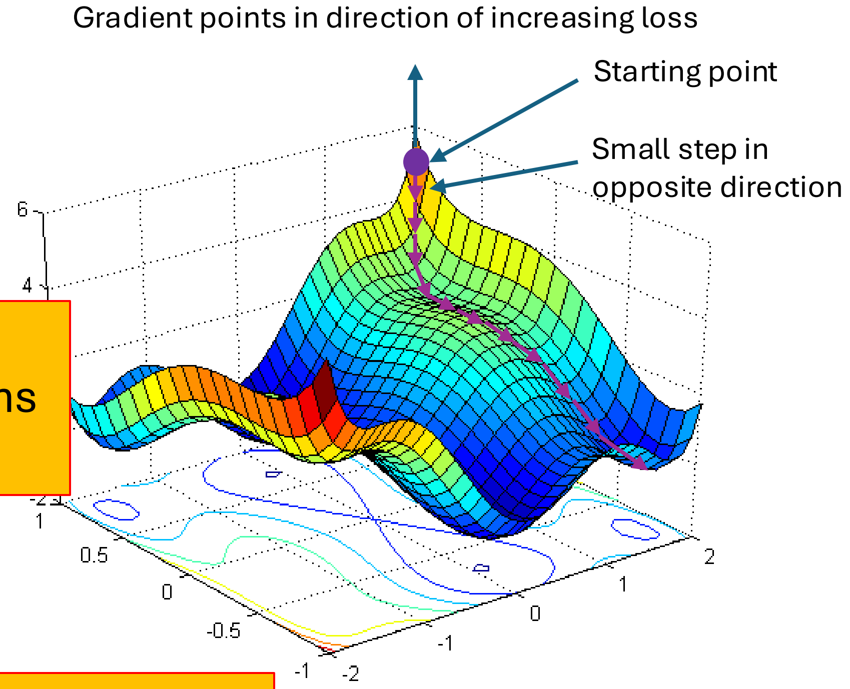
Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

Understanding gradient descent is the single most important concept in all of Deep Learning. Most decisions in DL are made for reasons related to gradients.

For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$



Gradient Descent does not converge to the global minimum.
It can (and pretty much always does) get stuck in local minima.

Gradients

Gradients are important for gradient descent…

How can we actually find them?

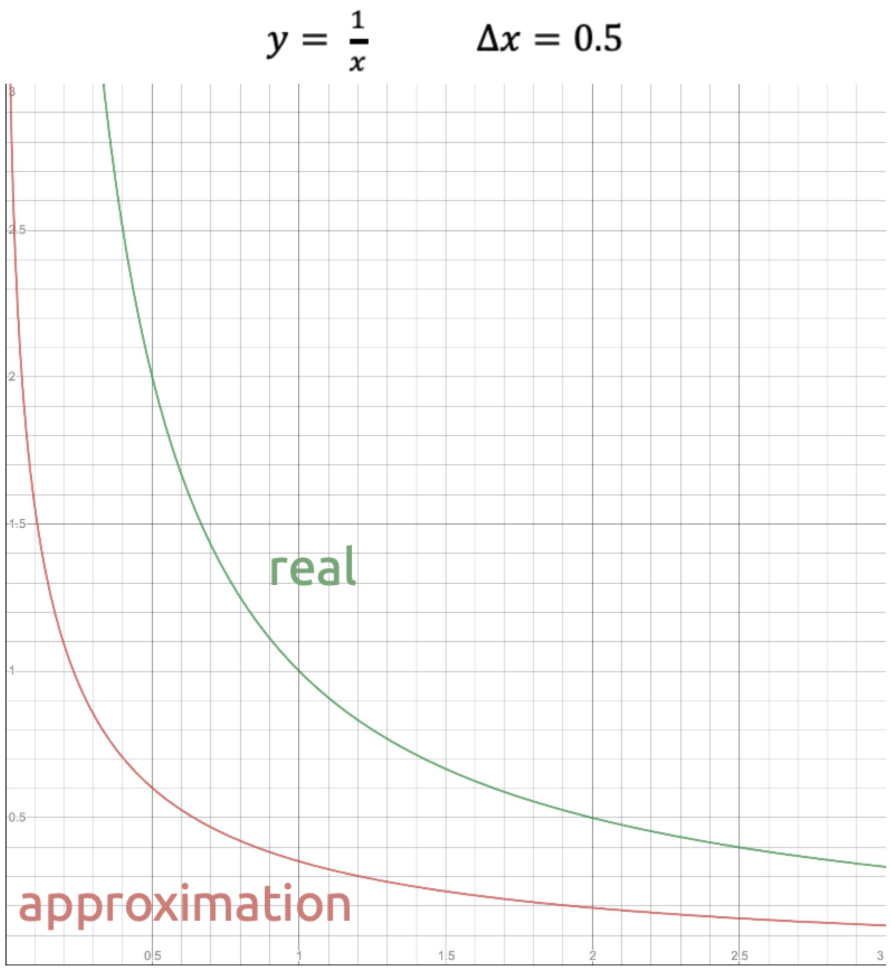
1. Numeric Differentiation
2. Symbolic Differentiation
3. Automatic Differentiation

Computer-based Derivatives

- **Numeric differentiation**
  - $\frac{df}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$
  - Pick a small step size Δx
  - Also called “finite differences”

Computer-based Derivatives

- **Numeric differentiation**
  - $\frac{df}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$
  - Pick a small step size Δx
  - Also called “finite differences”
  - Easy to implement
  - Arbitrarily inaccurate/unstable

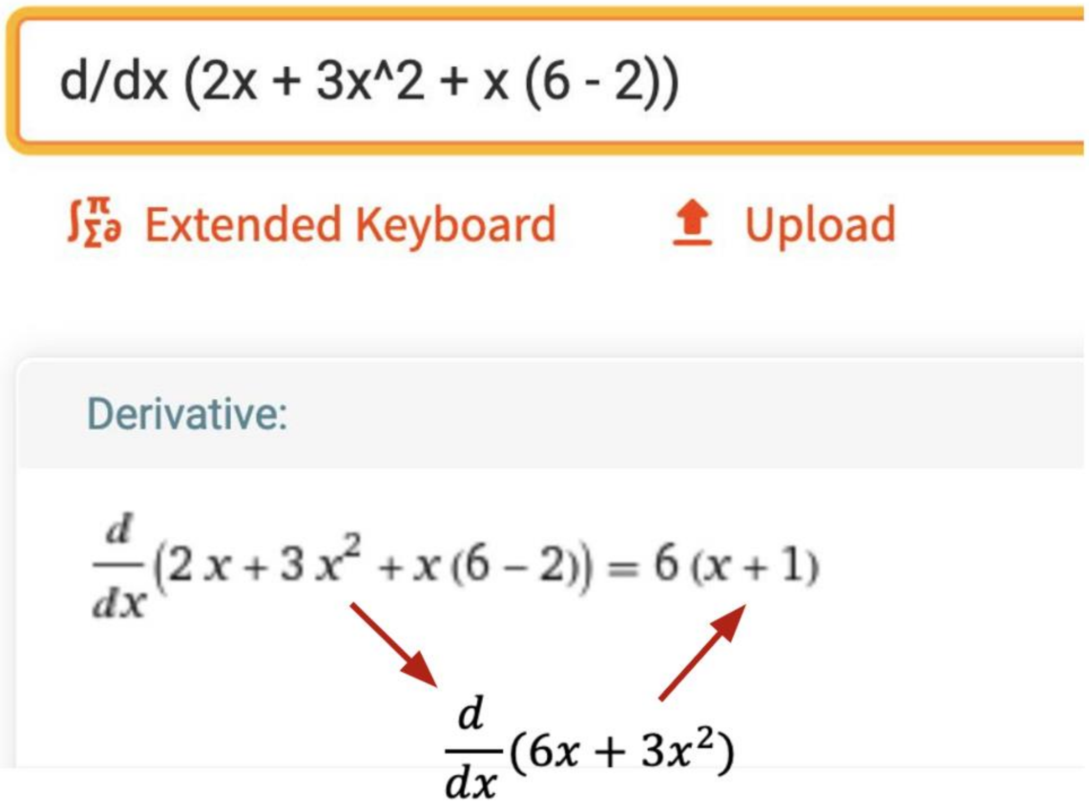


Symbolic Differentiation

Your computer performs algebra on symbols

Returns exact answers

Hard to implement, inefficient
Only handles static expressions (no loops)



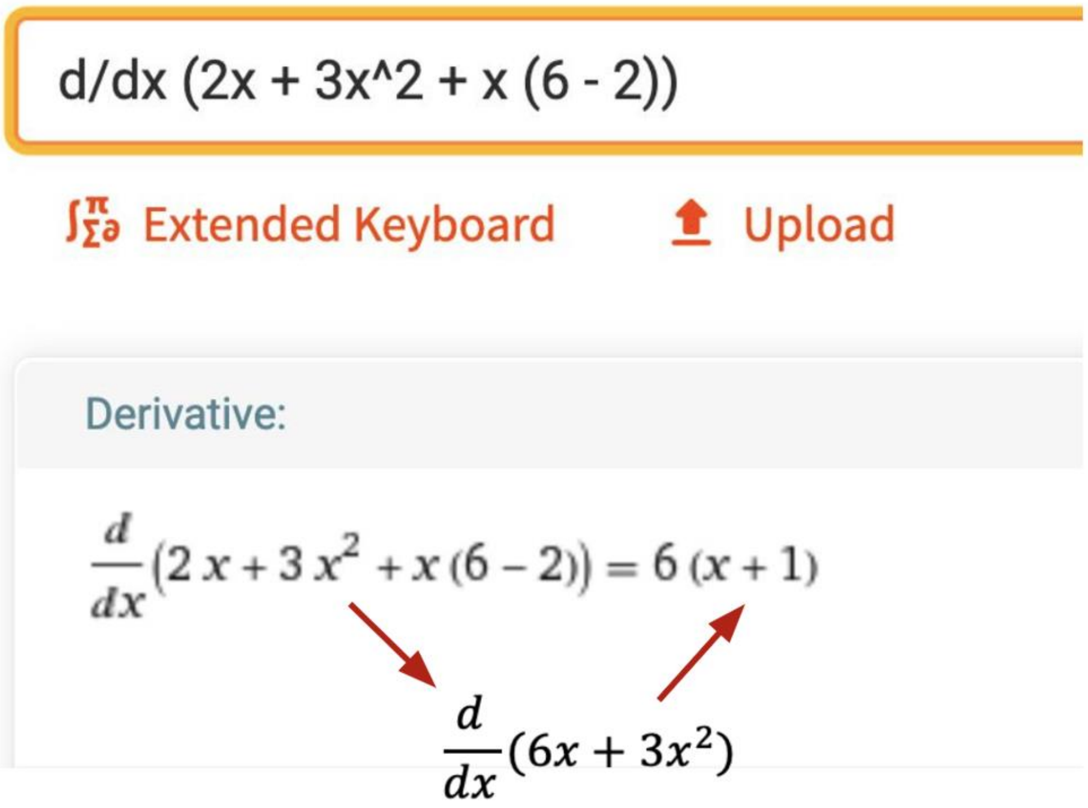
Symbolic Differentiation

Your computer performs algebra on symbols

Returns exact answers

Hard to implement, inefficient
Only handles static expressions (no loops)

```
While abs(x) >5:
  x = x / 2
```



Automatic Differentiation

Build a “compute graph” that tracks all operations during execution of a program

Automatically compute desired derivatives

Tensorflow

One of the three main neural network frameworks in Python

Gradient tape: main method of auto-differentiation

Tensorflow

One of the three main neural network frameworks in Python

Gradient tape: main method of auto-differentiation

While gradient tape is active:
1. Record all operations involving tracked tensorflow variables
2. Each operation stores the result of the operation and “parent” tensors

After loss is calculated (i.e., all operations are done), tape.gradient(loss, model.trainable_parameters) will return the gradient for all parameters in the model

Chain rule

If f and g are both differentiable and $F(x)$ is the composite function defined by $F(x) = f(g(x))$ then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x))\, g'(x)$$

Differentiate outer function

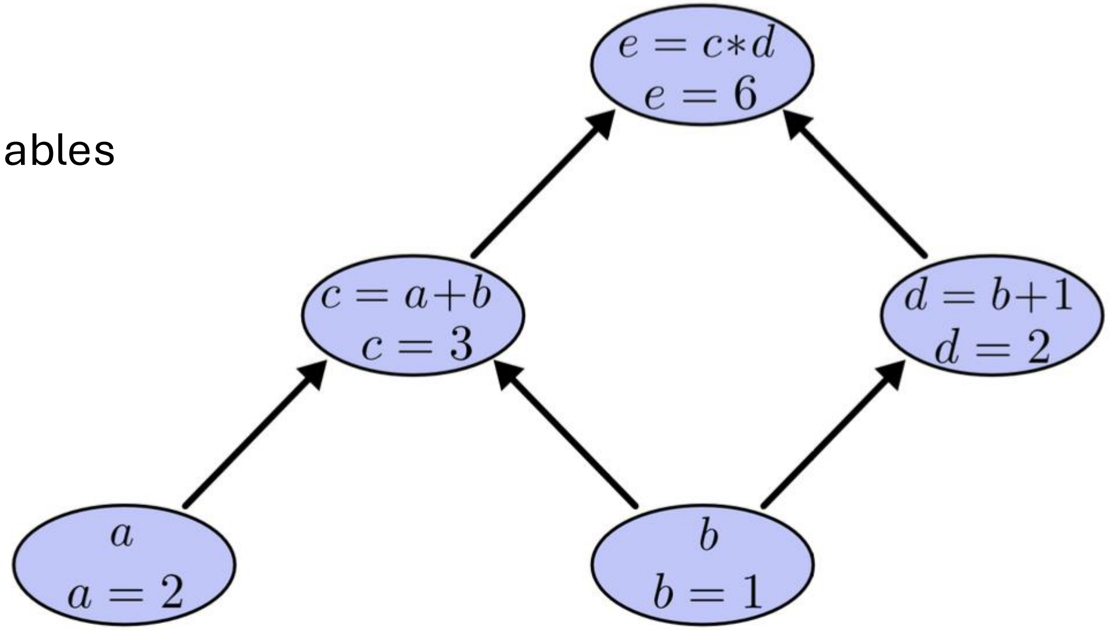
Differentiate inner function

Courtesy: https://www.onlinemathlearning.com/chain-rule.html

Computation Graph

$$e = (a+b)\cdot(b+1)$$

For each node, gradient tape tracks the operation and input variables



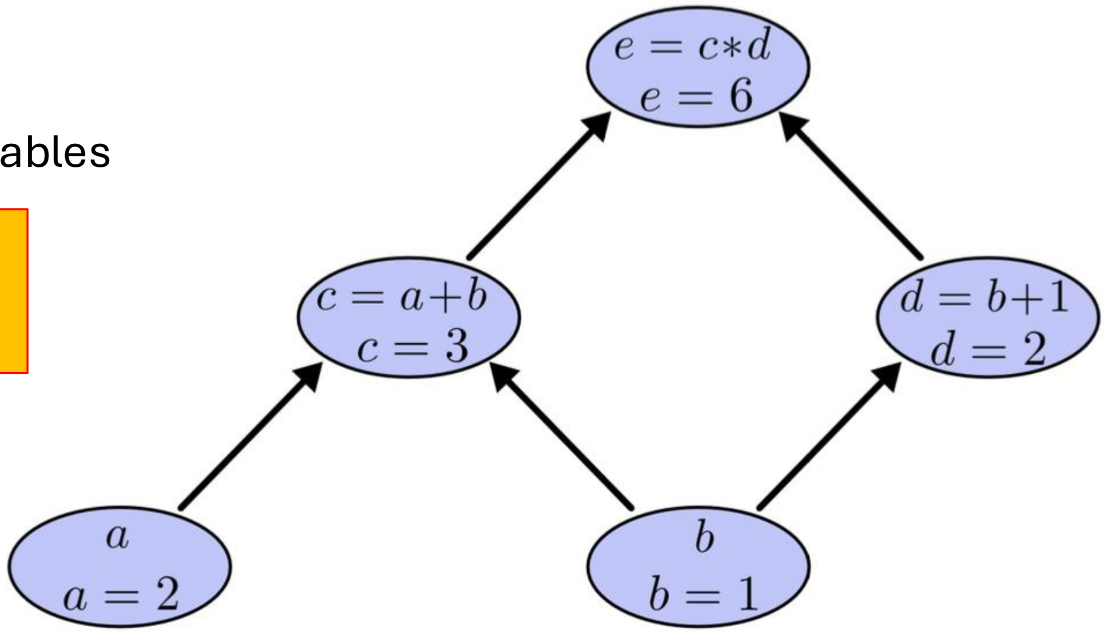
$$\frac{de}{da} = \frac{de}{dc}\frac{dc}{da} = d \qquad \frac{de}{db} = \frac{de}{dd}\frac{dd}{db} + \frac{de}{dc}\frac{dc}{db} = c + d$$

Computation Graph

$$e = (a + b) \cdot (b + 1)$$

For each node, gradient tape tracks the operation and input variables

How to compute derivatives of *e* with respect to each input?
Want $\frac{\mathrm{de}}{\mathrm{da}}, \frac{\mathrm{de}}{\mathrm{db}}$



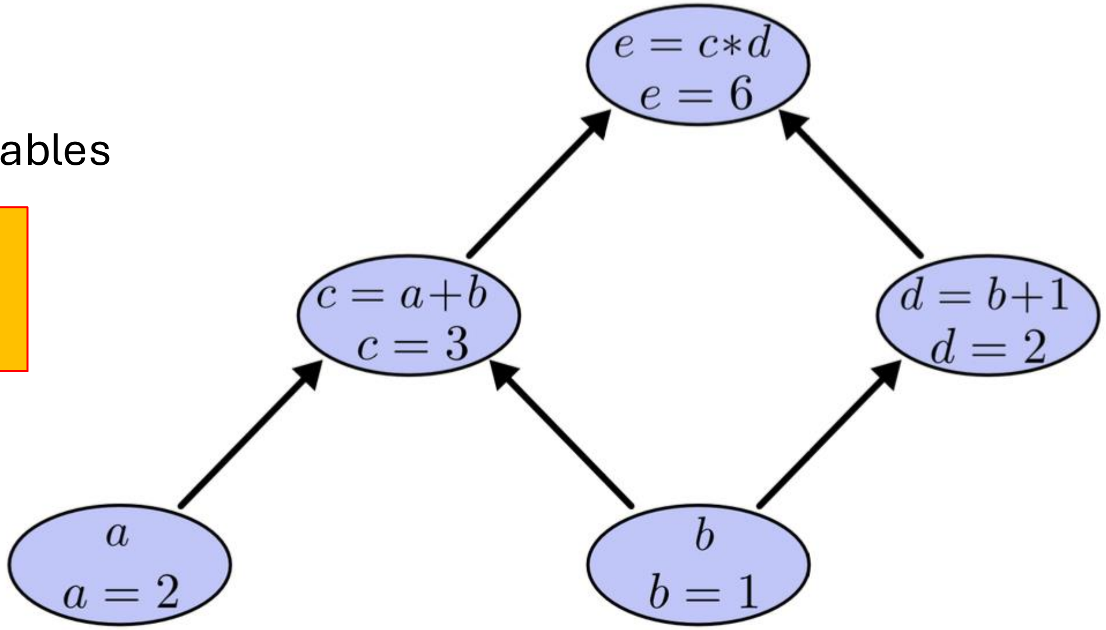
$$\frac{\mathrm{de}}{\mathrm{da}} = \frac{\mathrm{de}}{\mathrm{dc}}\frac{\mathrm{dc}}{\mathrm{da}} = \mathrm{d} \qquad \frac{\mathrm{de}}{\mathrm{db}} = \frac{\mathrm{de}}{\mathrm{dd}}\frac{\mathrm{dd}}{\mathrm{db}} + \frac{\mathrm{de}}{\mathrm{dc}}\frac{\mathrm{dc}}{\mathrm{db}} = \mathrm{c} + \mathrm{d}$$

Computation Graph

$$e = (a + b) \cdot (b + 1)$$

For each node, gradient tape tracks the operation and input variables

How to compute derivatives of *e* with respect to each input?
Want $\frac{\mathrm{de}}{\mathrm{da}}, \frac{\mathrm{de}}{\mathrm{db}}$



1. Run compute graph in "forward direction"
   Compute *e* by executing each operation
2. Run compute graph in "reverse direction"
   Compute derivatives at each node

$$\frac{\mathrm{de}}{\mathrm{da}} = \frac{\mathrm{de}}{\mathrm{dc}}\frac{\mathrm{dc}}{\mathrm{da}} = \mathrm{d}$$

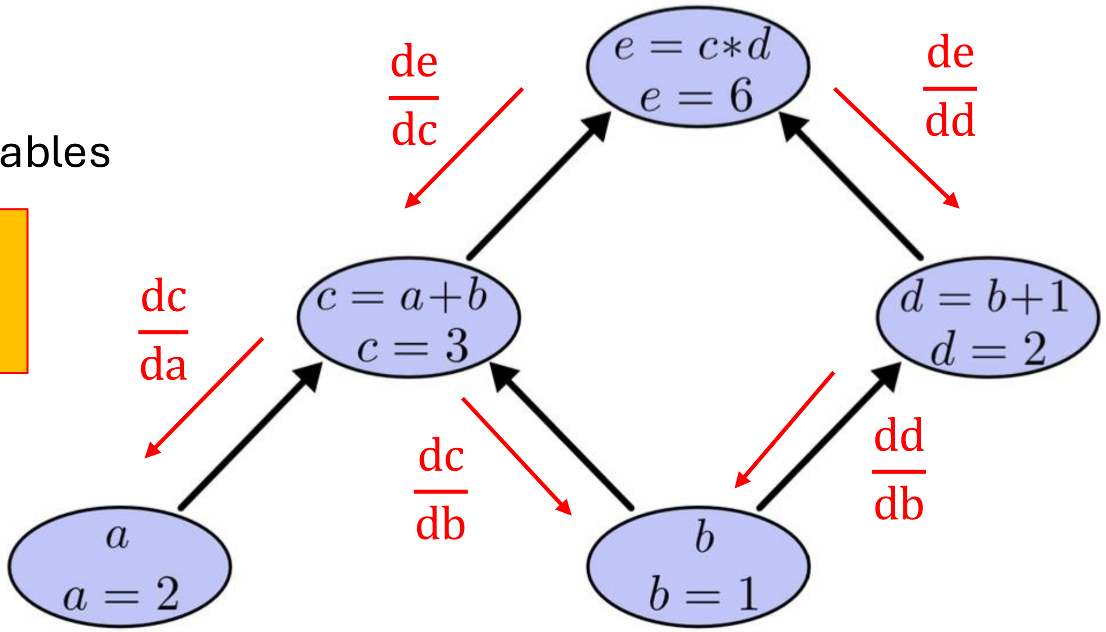
$$\frac{\mathrm{de}}{\mathrm{db}} = \frac{\mathrm{de}}{\mathrm{dd}}\frac{\mathrm{dd}}{\mathrm{db}} + \frac{\mathrm{de}}{\mathrm{dc}}\frac{\mathrm{dc}}{\mathrm{db}} = \mathrm{c} + \mathrm{d}$$

Computation Graph

$$e = (a + b) \cdot (b + 1)$$

For each node, gradient tape tracks the operation and input variables

How to compute derivatives of *e* with respect to each input?
Want $\frac{de}{da}$, $\frac{de}{db}$



1. Run compute graph in "forward direction"
   Compute *e* by executing each operation
2. Run compute graph in "reverse direction"
   Compute derivatives at each node

$$\frac{de}{da} = \frac{de}{dc}\frac{dc}{da} = d \qquad \frac{de}{db} = \frac{de}{dd}\frac{dd}{db} + \frac{de}{dc}\frac{dc}{db} = c + d$$

Order of Backward Pass

Where should we start our backward pass? Which order should we calculate derivatives in?

Start at output nodes (nodes with no children in forward compute graph), then move to parents, then parents' parents, and so on.

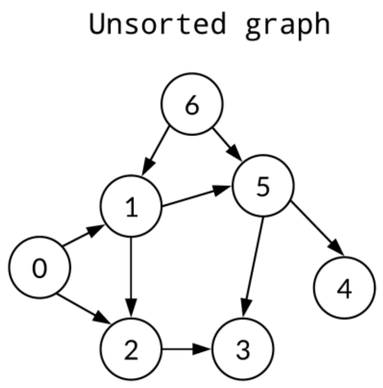
Order of Backward Pass

Where should we start our backward pass? Which order should we calculate derivatives in?

Start at output nodes (nodes with no children in forward compute graph), then move to parents, then parents' parents, and so on.

Fully-correct order: Topological order of reverse graph!

For HW 3, running Breadth-First-Search order is sufficient



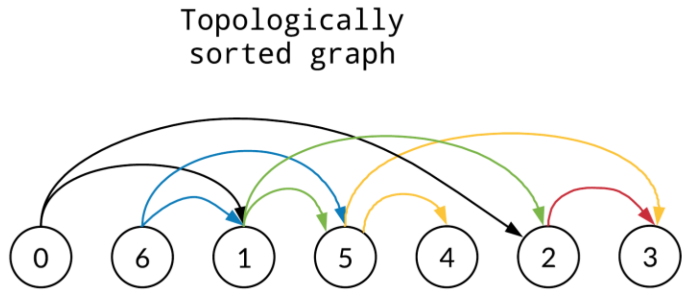


Image source: https://guides.codepath.com/compsci/Topological-Sort

Want to find $\frac{\mathrm{dL}}{\mathrm{d}\theta}$, where θ is the set of trainable parameters (w, b)

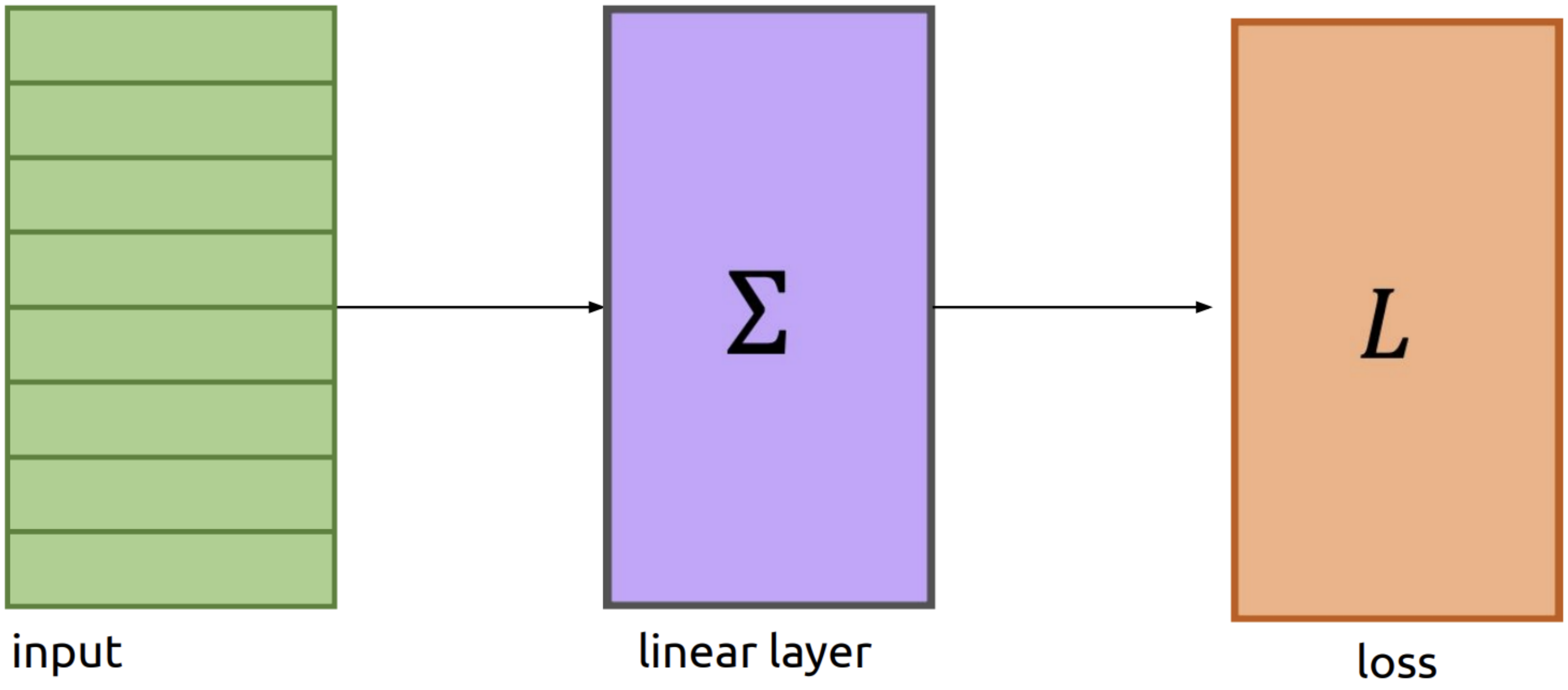
j=1

$$l_j = \sum_k w_{j,k}\, x_k + b_j$$

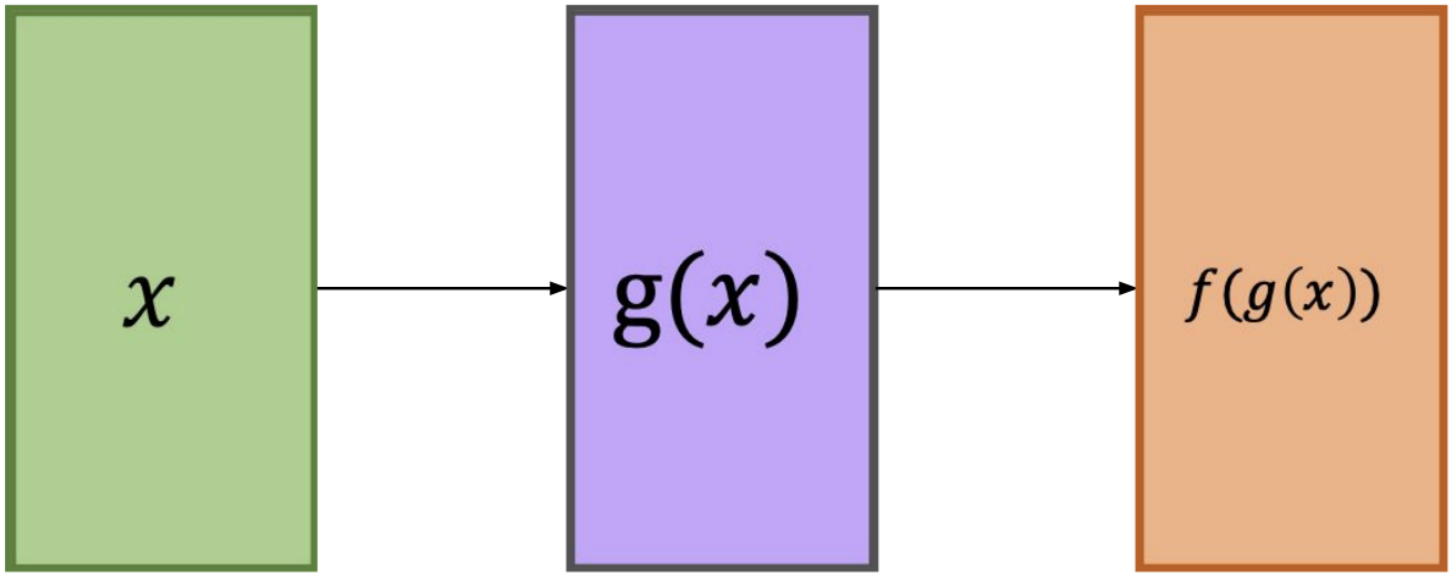
$$l \;= \boldsymbol{w}.\boldsymbol{x} \;+ \boldsymbol{b}$$

$\mathbf{x}$

$L = (y - l)^2$



Looking at composite function!



Applying Chain rule [Example]

$$f(x) = x^2 \qquad \mathrm{g}(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad \text{g}(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion:

Applying Chain rule [Example]

$$f(x) = x^2 \qquad \mathrm{g}(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{\mathrm{F(x)} = 4\mathrm{x}^4 + 4\mathrm{x}^2 + 1}$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$

By Chain rule:

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

By Chain rule:

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

By Chain rule:

$$\boxed{\frac{dF}{dx} = \frac{dF}{df}\frac{df}{dg}\frac{dg}{dx}}$$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

By Chain rule:

$\boxed{\frac{dF}{dx} = \frac{dF}{df}\frac{df}{dg}\frac{dg}{dx}}$ $\boxed{\frac{dF}{df} = 1}$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

By Chain rule:

$\boxed{\frac{dF}{dx} = \frac{dF}{df}\frac{df}{dg}\frac{dg}{dx}}$ $\boxed{\frac{dF}{df} = 1}$ $\boxed{\frac{df}{dg} = 2g(x) = 4x^2 + 2}$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad\qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

By Chain rule:

$\boxed{\frac{dF}{dx} = \frac{dF}{df}\frac{df}{dg}\frac{dg}{dx}}$ $\boxed{\frac{dF}{df} = 1}$ $\boxed{\frac{df}{dg} = 2g(x) = 4x^2 + 2}$ $\boxed{\frac{dg}{dx} = 4x}$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

By Chain rule:

$\boxed{\frac{dF}{dx} = \frac{dF}{df}\frac{df}{dg}\frac{dg}{dx}}$ $\boxed{\frac{dF}{df} = 1}$ $\boxed{\frac{df}{dg} = 2g(x) = 4x^2 + 2}$ $\boxed{\frac{dg}{dx} = 4x}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

By Expansion: $\boxed{F(x) = 4x^4 + 4x^2 + 1}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

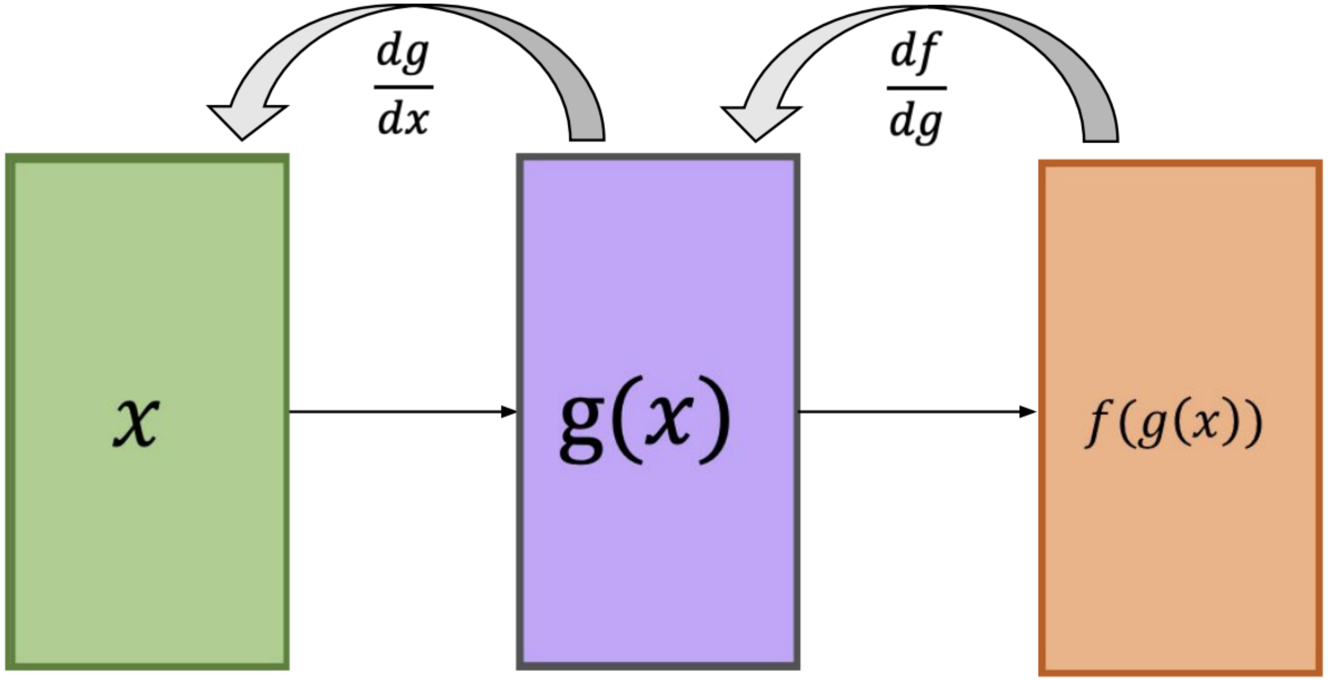
By Chain rule:

$\boxed{\frac{dF}{dx} = \frac{dF}{df}\frac{df}{dg}\frac{dg}{dx}}$ $\boxed{\frac{dF}{df} = 1}$ $\boxed{\frac{df}{dg} = 2g(x) = 4x^2 + 2}$ $\boxed{\frac{dg}{dx} = 4x}$ $\boxed{\frac{dF}{dx} = 16x^3 + 8x}$

Important: We will often need the value of the function to find derivatives through the chain rule

The Chain Rule (for Differentiation)

- Given arbitrary function: $f(g(x)) \Rightarrow \frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$



Each layer computes the gradients with respect to it's variables and passes the result backwards

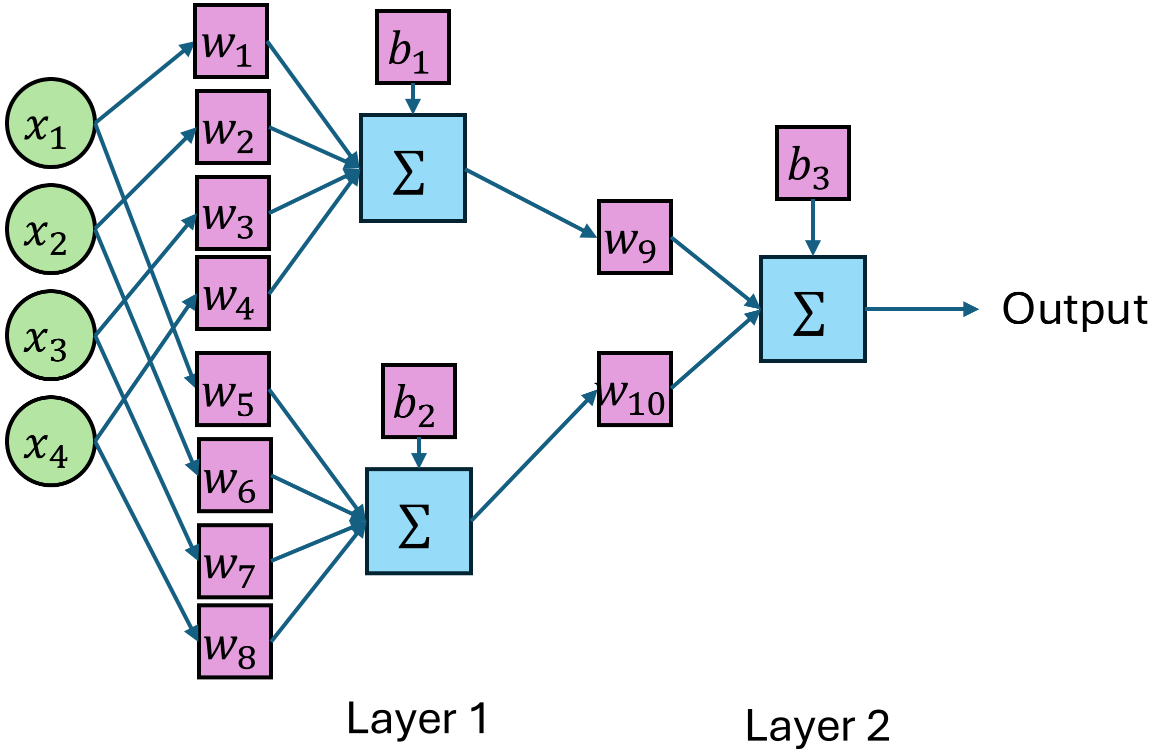
Backpropagation

(or backward pass)

Backpropagation

Notation:
w, b: weights and biases
z: Intermediate Value
a: Value after activation function

Simple Neural Network with 1 hidden layer,
Assume activation function in hidden layer is
ReLU: (max(0, x))



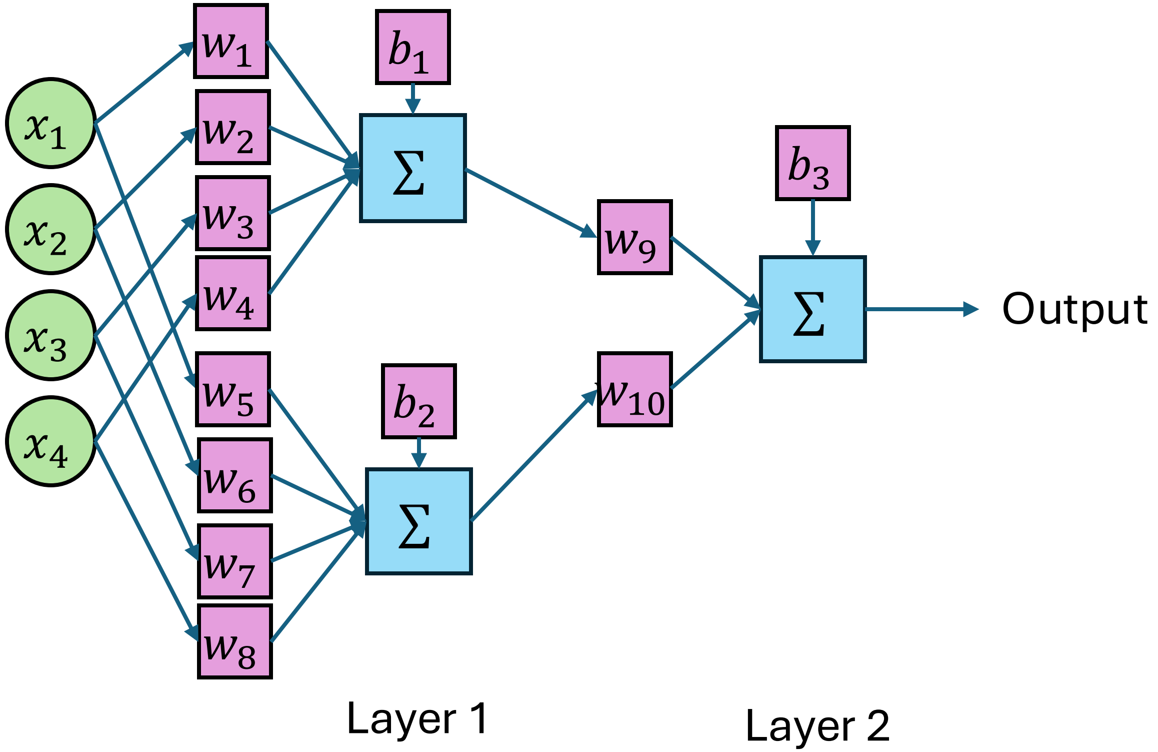
Backpropagation

Notation:
w, b: weights and biases
z: Intermediate Value
a: Value after activation function

Layer 1:

$$z_1 = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b_1$$
$$a_1 = relu(z_1)$$
$$z_2 = w_5 x_1 + w_6 x_2 + w_7 x_3 + w_8 x_4 + b_2$$
$$a_2 = relu(z_2)$$

Simple Neural Network with 1 hidden layer,
Assume activation function in hidden layer is
ReLU: (max(0, x))



Backpropagation

Notation:
w, b: weights and biases
z: Intermediate Value
a: Value after activation function

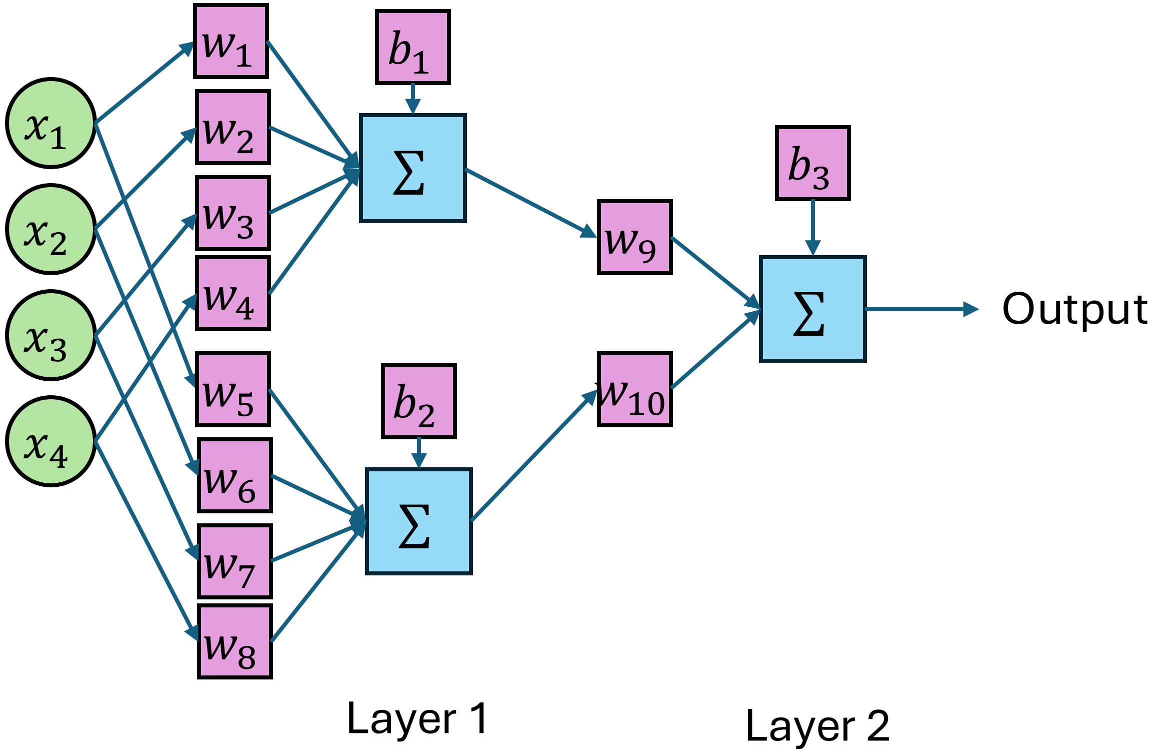
Layer 1:

$$z_1 = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b_1$$
$$a_1 = relu(z_1)$$
$$z_2 = w_5x_1 + w_6x_2 + w_7x_3 + w_8x_4 + b_2$$
$$a_2 = relu(z_2)$$

$$z_3 = w_9a_1 + w_{10}a_2 + b_3$$

No activation function on final output (assume we are performing a regression task that can have any output value)

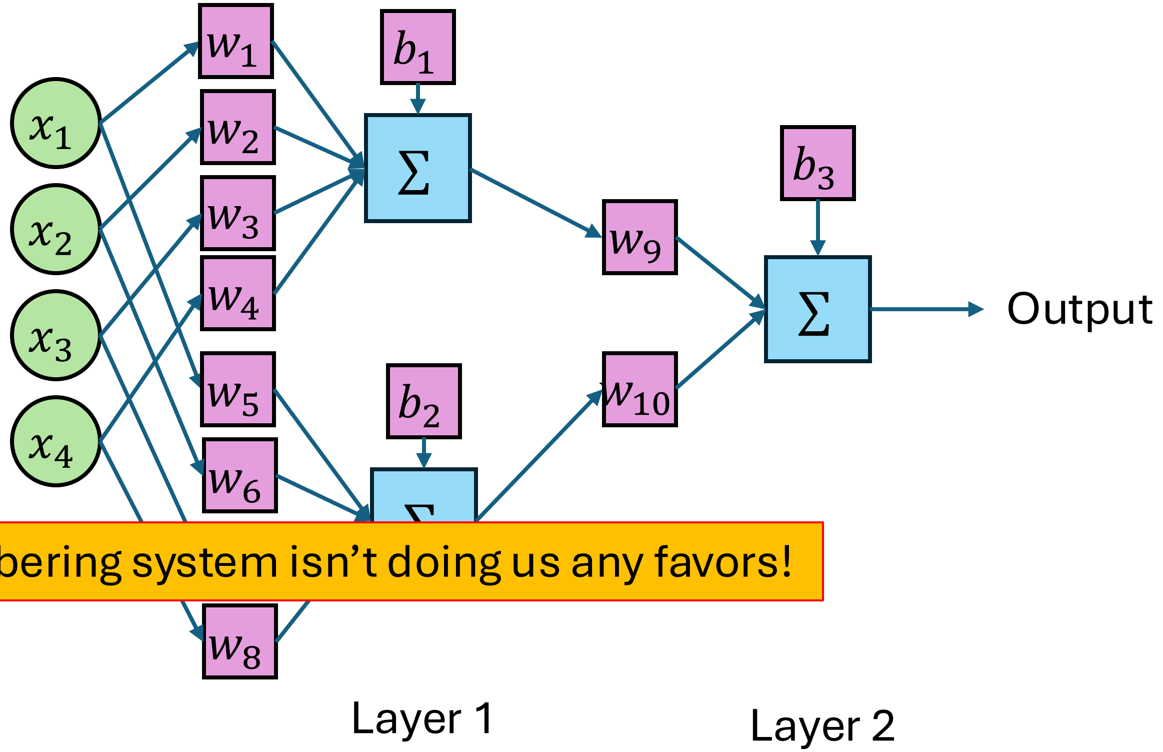
Simple Neural Network with 1 hidden layer,
Assume activation function in hidden layer is ReLU: (max(0, x))



Backpropagation

Notation:
w, b: weights and biases
z: Intermediate Value
a: Value after activation function

Simple Neural Network with 1 hidden layer,
Assume activation function in hidden layer is
ReLU: (max(0, x))



Our numbering system isn't doing us any favors!

Layer 1:

$$z_1 = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b_1$$
$$a_1 = relu(z_1)$$
$$z_2 = w_5 x_1 + w_6 x_2 + w_7 x_3 + w_8 x_4 + b_2$$
$$a_2 = relu(z_2)$$

$$z_3 = w_9 a_1 + w_{10} a_2 + b_3$$

No activation function on final output (assume we are performing a regression task that can have any output value)

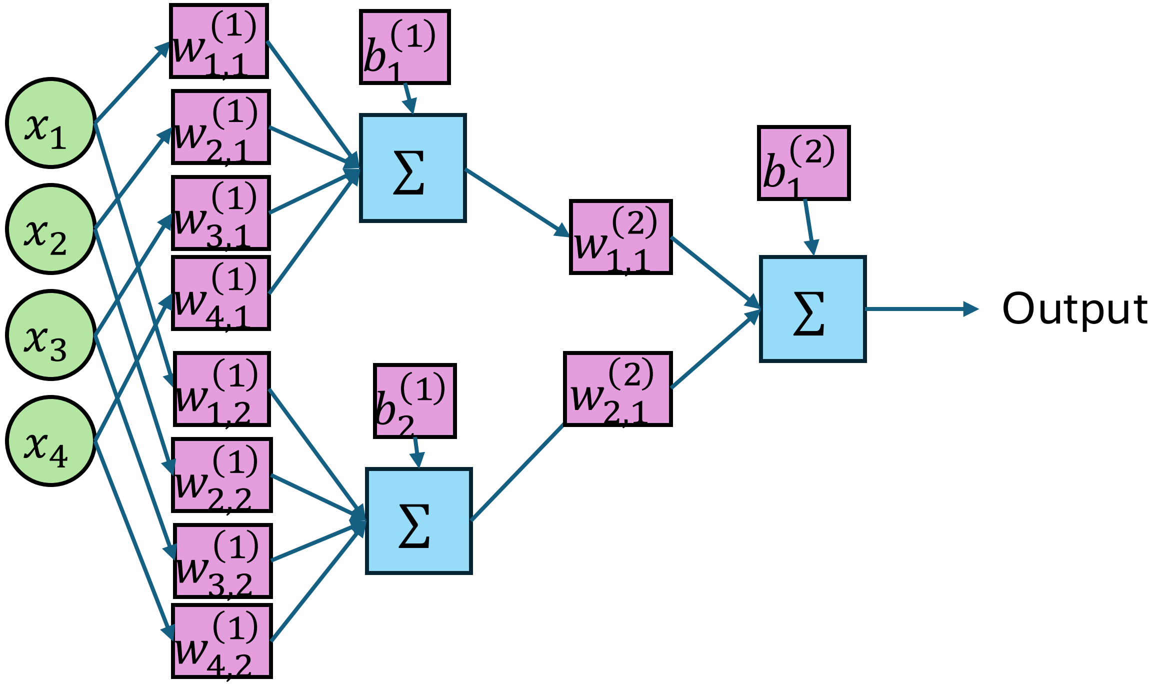
Backpropagation

Relabel weights based on layer number, input number, and output number.

$w_{i,out}^{(l)}$ is the weight associated with layer l, input i, and output out.

$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \dots \\ b_{out}^{(l)} \end{bmatrix}$$

$$\mathrm{x}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \dots \\ x_i \end{bmatrix}$$



Backpropagation

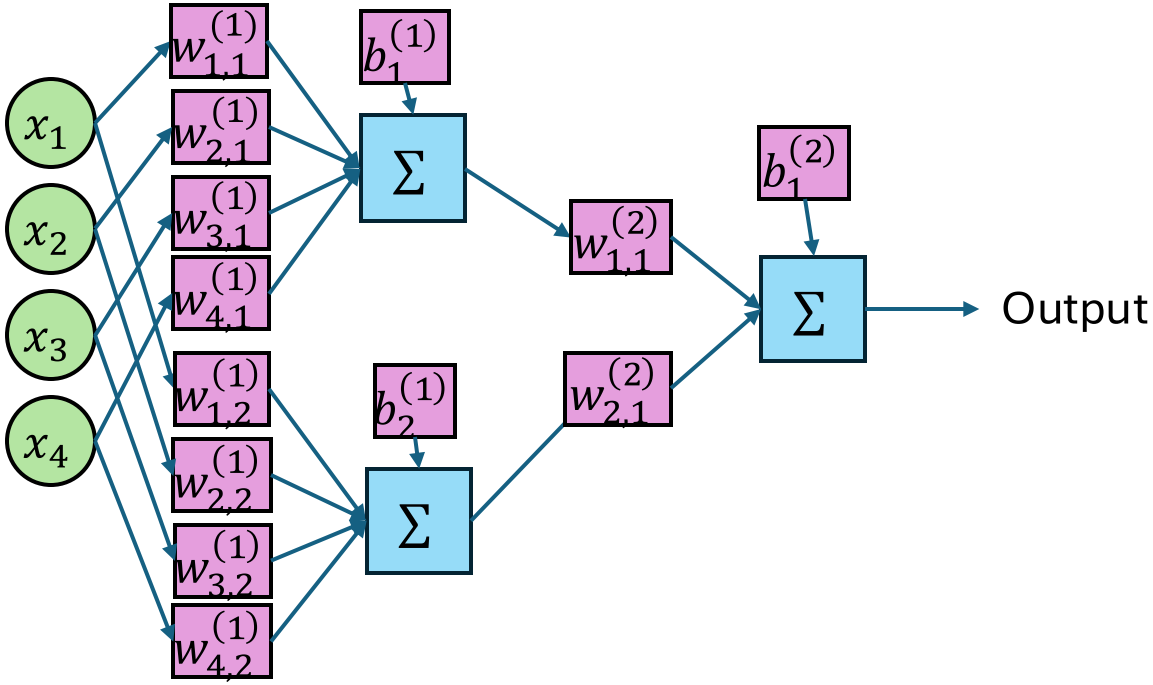
Relabel weights based on layer number, input number, and output number.

$w_{i,out}^{(l)}$ is the weight associated with layer l, input i, and output out.

$$W^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & \cdots & w_{1,\text{out}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{\text{in},1}^{(l)} & \cdots & w_{\text{in,out}}^{(l)} \end{bmatrix}$$

$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \dots \\ b_{out}^{(l)} \end{bmatrix}$$

$$\mathrm{x}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \dots \\ x_i \end{bmatrix}$$



Backpropagation

$z^{(1)} = W^{(1)}x + b^{(1)}$ or $z^{(1)} = xW^{(1)} + b^{(1)}$?

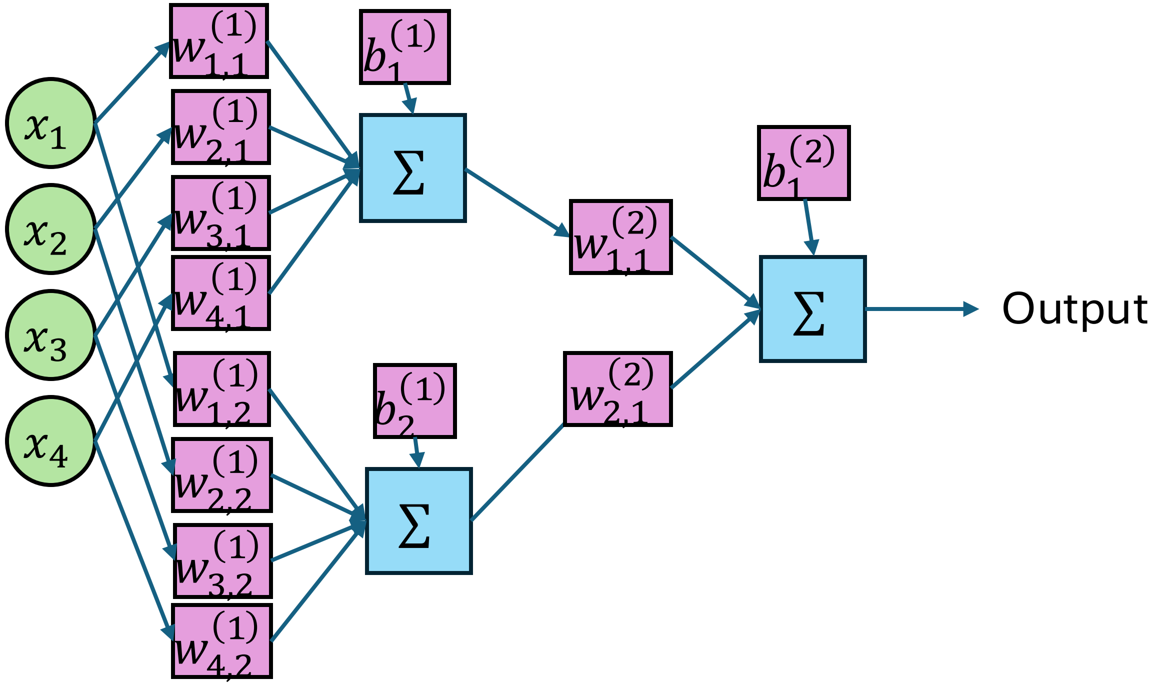
Relabel weights based on layer number, input number, and output number.

$w_{i,out}^{(l)}$ is the weight associated with layer l, input i, and output out.

$$W^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & \cdots & w_{1,\text{out}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{\text{in},1}^{(l)} & \cdots & w_{\text{in,out}}^{(l)} \end{bmatrix}$$

$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \cdots \\ b_{out}^{(l)} \end{bmatrix}$$

$$\mathrm{x}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \cdots \\ x_i \end{bmatrix}$$



Backpropagation

$z^{(1)} = W^{(1)}x + b^{(1)}$ or $z^{(1)} = xW^{(1)} + b^{(1)}$?

Shapes:
W: in x out
b: out x 1
x: 1 x in

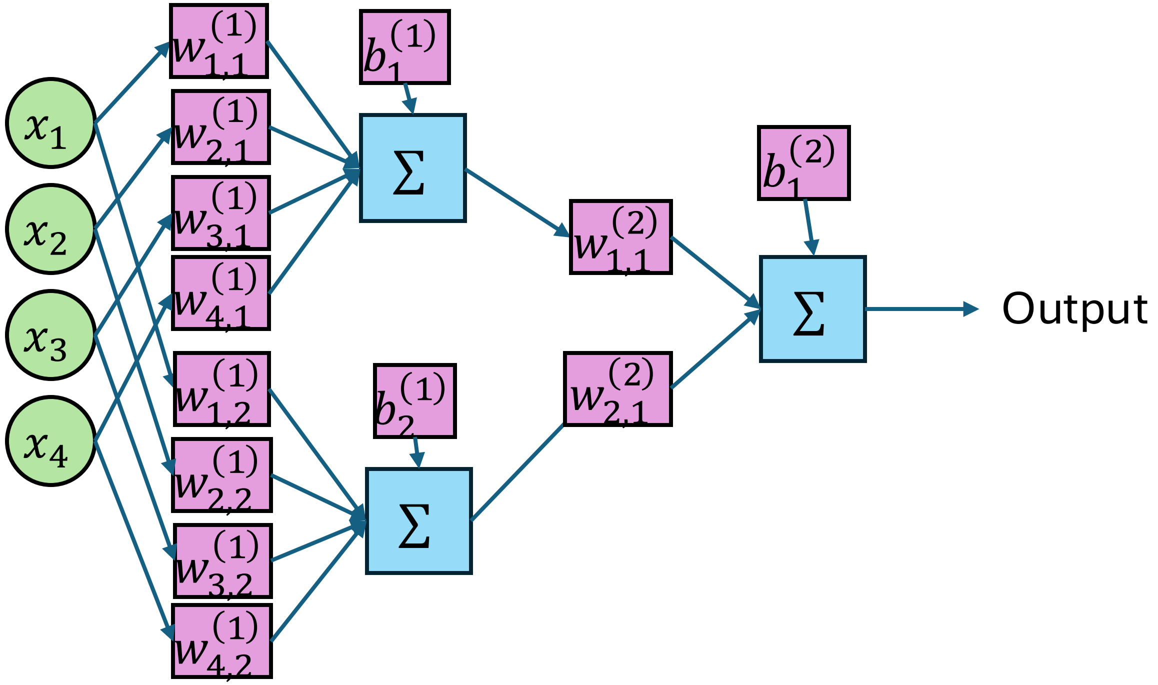
Relabel weights based on layer number, input number, and output number.

$w_{i,out}^{(l)}$ is the weight associated with layer l, input i, and output out.

$$W^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & \cdots & w_{1,\text{out}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{\text{in},1}^{(l)} & \cdots & w_{\text{in,out}}^{(l)} \end{bmatrix}$$

$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \dots \\ b_{out}^{(l)} \end{bmatrix}$$

$$\mathrm{x}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \dots \\ x_i \end{bmatrix}$$



Backpropagation

$\boxed{z^{(1)} = W^{(1)}x + b^{(1)} \text{ or } z^{(1)} = xW^{(1)} + b^{(1)}?}$

$$z^{(1)} = xW^{(1)} + b^{(1)}$$
$$a^{(1)} = ReLU(z^{(1)})$$

Shapes:
W: in x out
b: out x 1
x: 1 x in

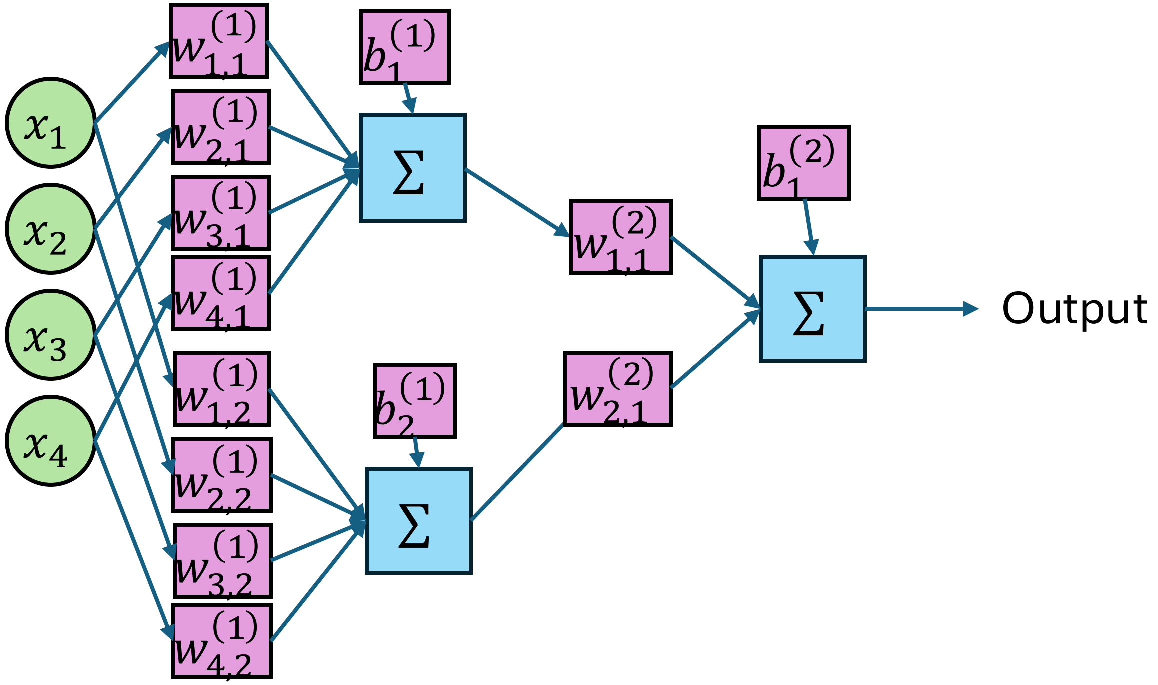
Relabel weights based on layer number, input number, and output number.

$w_{i,out}^{(l)}$ is the weight associated with layer l, input i, and output out.

$$W^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & \cdots & w_{1,\text{out}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{\text{in},1}^{(l)} & \cdots & w_{\text{in,out}}^{(l)} \end{bmatrix}$$

$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \cdots \\ b_{out}^{(l)} \end{bmatrix}$$

$$\mathrm{x}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \cdots \\ x_i \end{bmatrix}$$



Backpropagation

$\boxed{z^{(1)} = W^{(1)}x + b^{(1)} \text{ or } z^{(1)} = xW^{(1)} + b^{(1)}?}$

$$z^{(1)} = xW^{(1)} + b^{(1)}$$
$$a^{(1)} = ReLU(z^{(1)})$$

ReLU performed on each element of $z^{(1)}$

Shapes:
W: in x out
b: out x 1
x: 1 x in

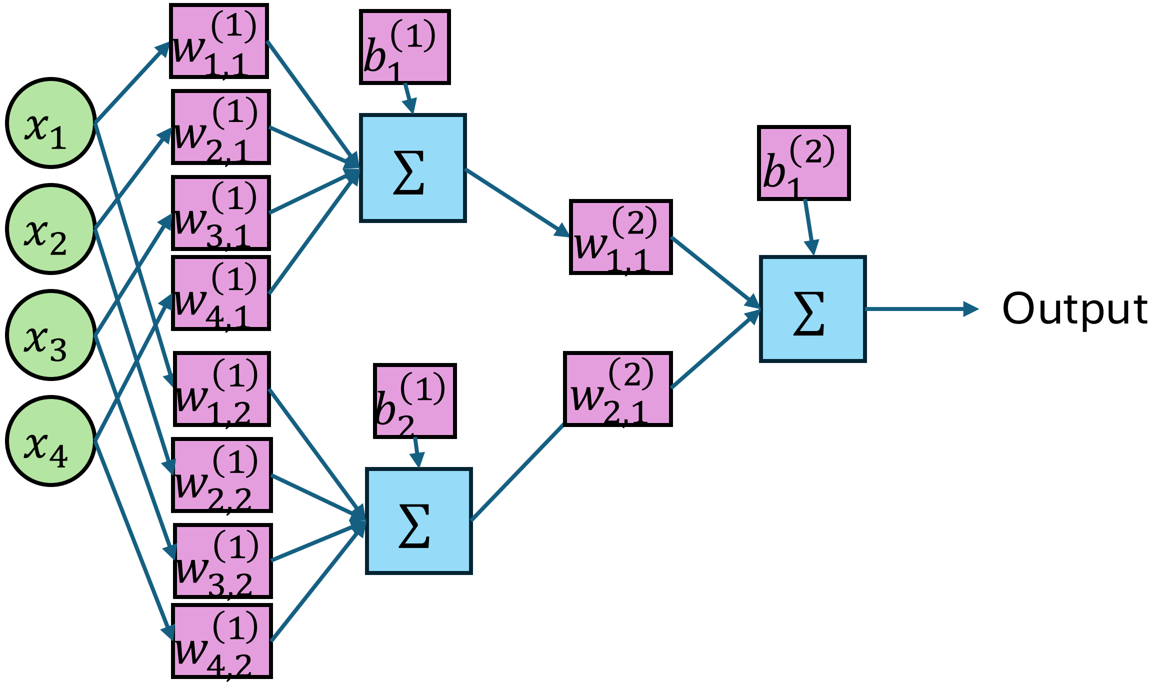
Relabel weights based on layer number, input number, and output number.

$w_{i,out}^{(l)}$ is the weight associated with layer l, input i, and output out.

$$W^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & \cdots & w_{1,\text{out}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{\text{in},1}^{(l)} & \cdots & w_{\text{in,out}}^{(l)} \end{bmatrix}$$

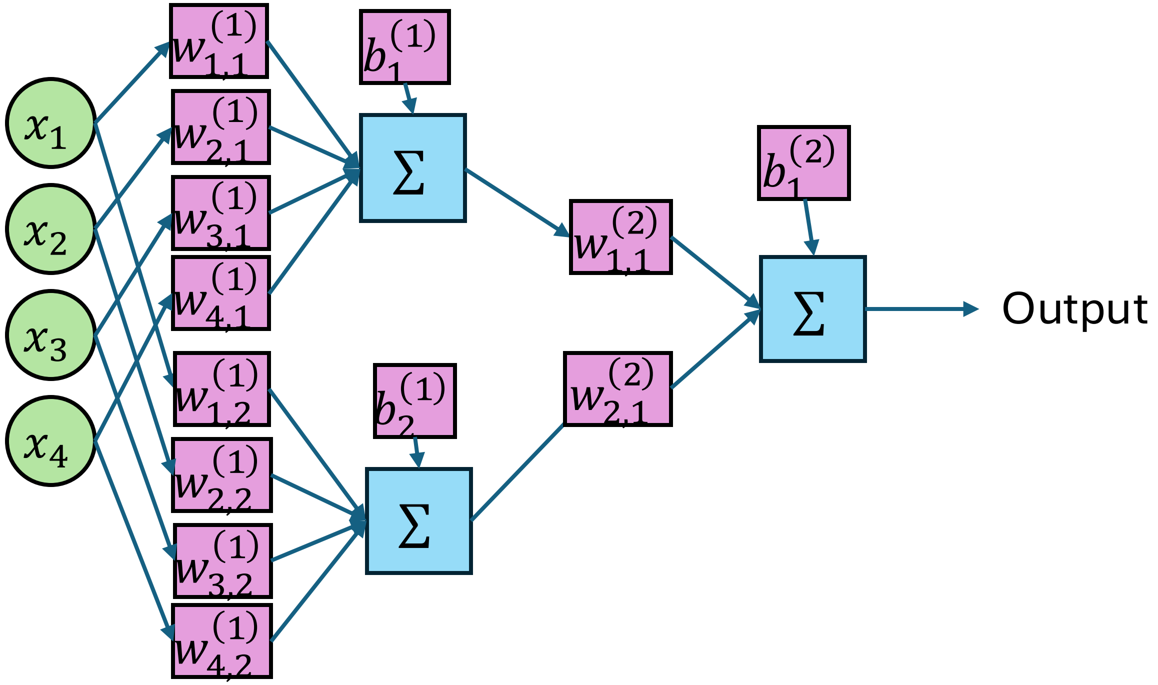
$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \dots \\ b_{out}^{(l)} \end{bmatrix}$$

$$\mathrm{x}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \dots \\ x_i \end{bmatrix}$$



Backpropagation

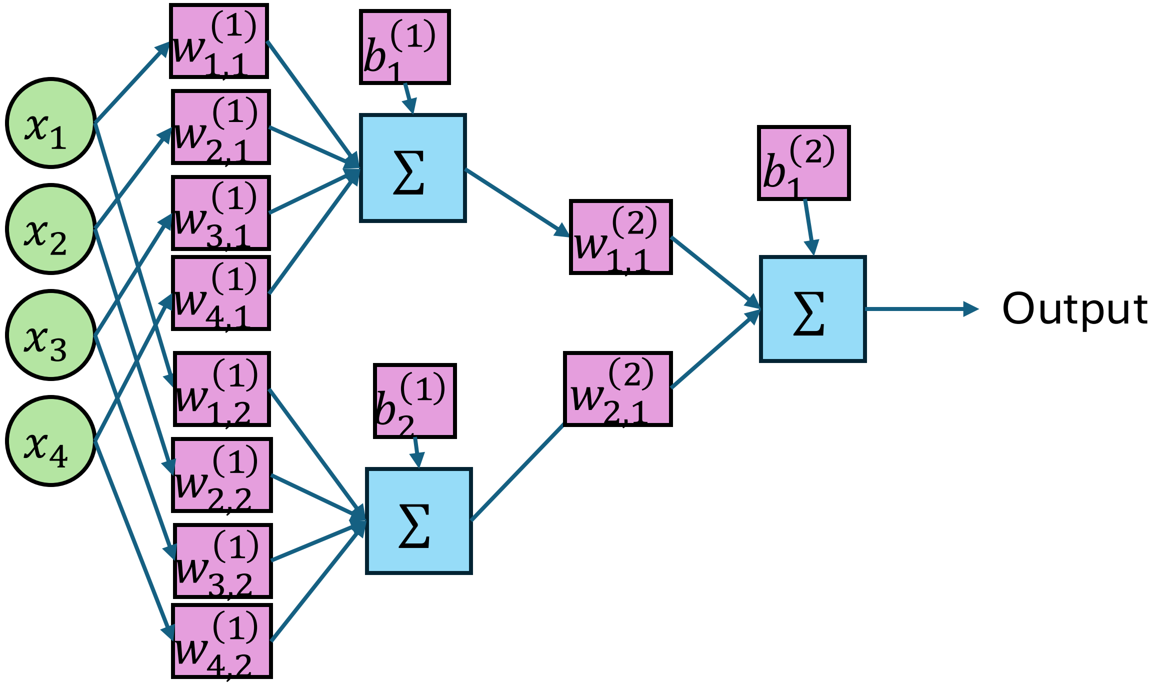
Output, $z^{(2)}$ = $ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$



Backpropagation

Output, $z^{(2)}$ = $ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.

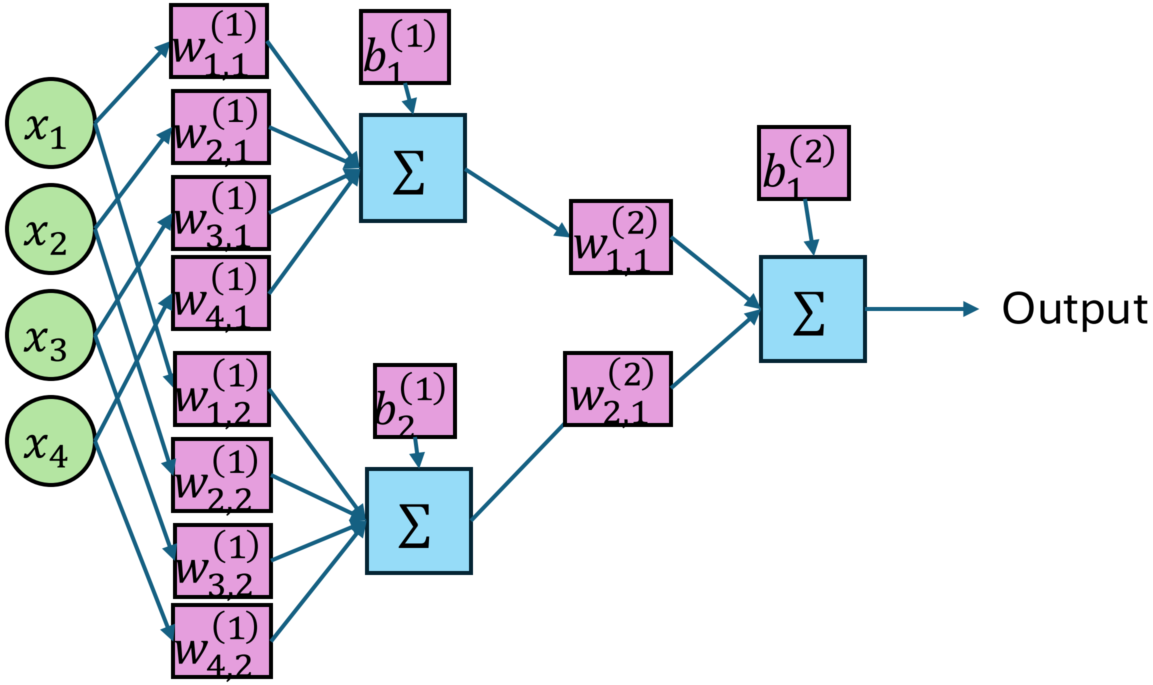


Backpropagation

Output, $z^{(2)} = ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.

What is the mean squared error?



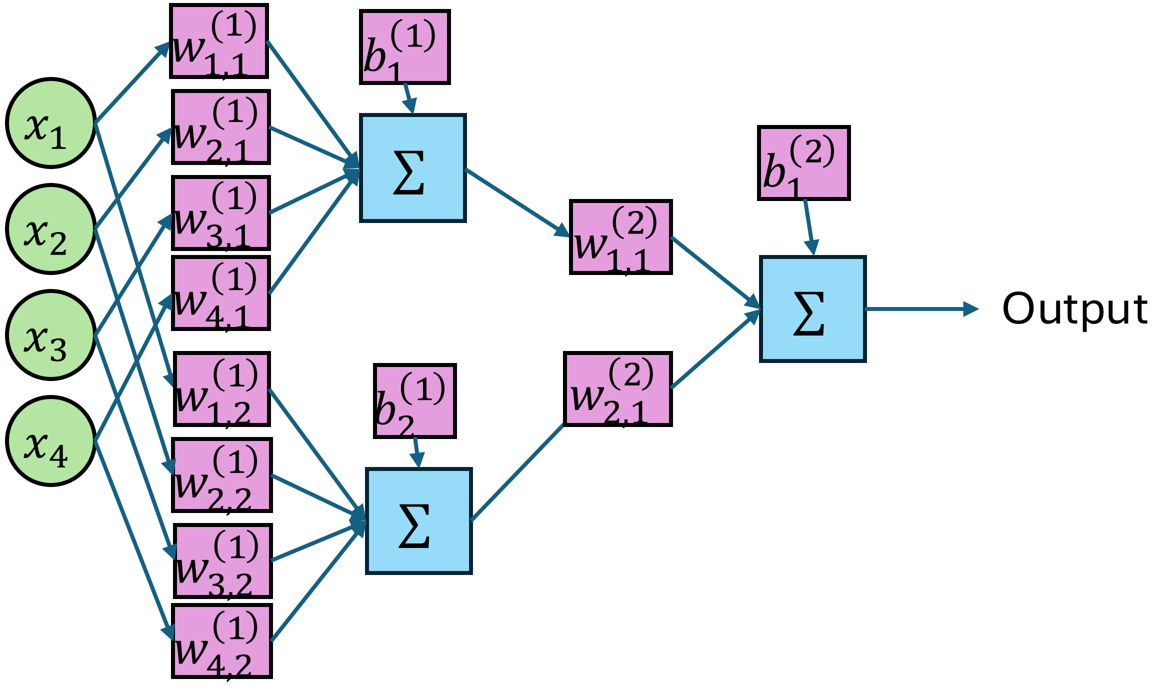
Backpropagation

Output, $z^{(2)}$ = $ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.

What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\hat{\ }2}{n}$$



Backpropagation

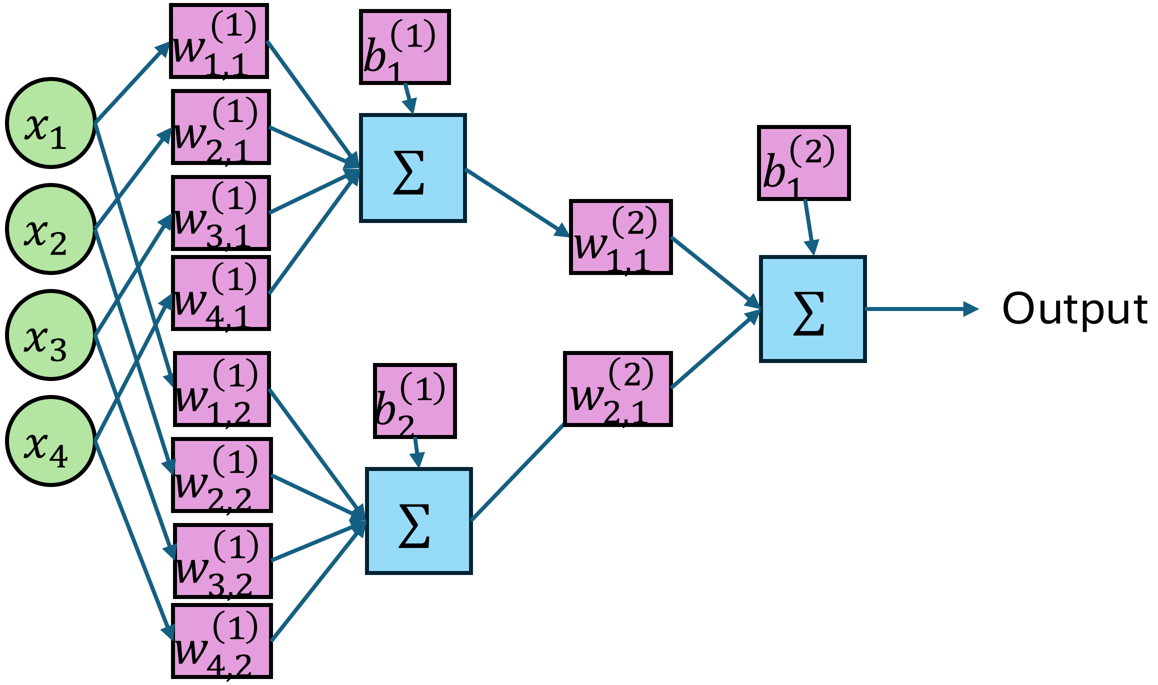
Output, $z^{(2)}$ = $ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.

What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\text{^}2}{n}$$

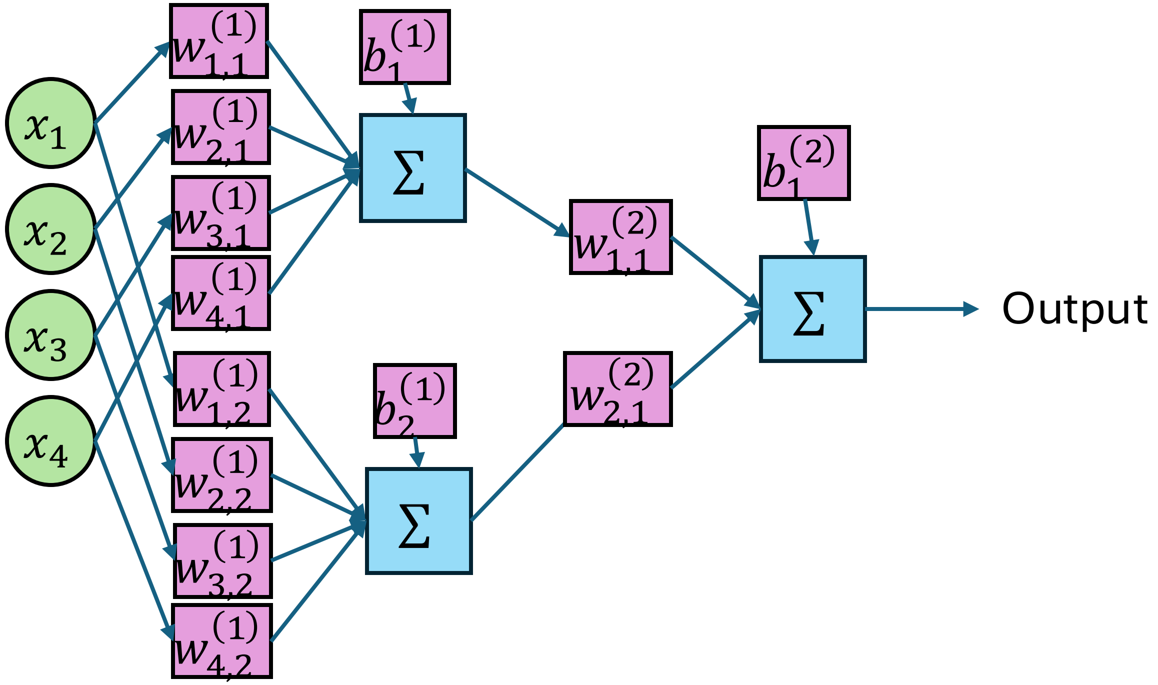
$$L = \left(z^{(2)} - y\right)^2$$



Backpropagation

Output, $z^{(2)} = ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.



What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\hat{}2}{n}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$L = (10 - 7)^2$$

$$L = 9$$

Backpropagation

Output, $z^{(2)} = ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.

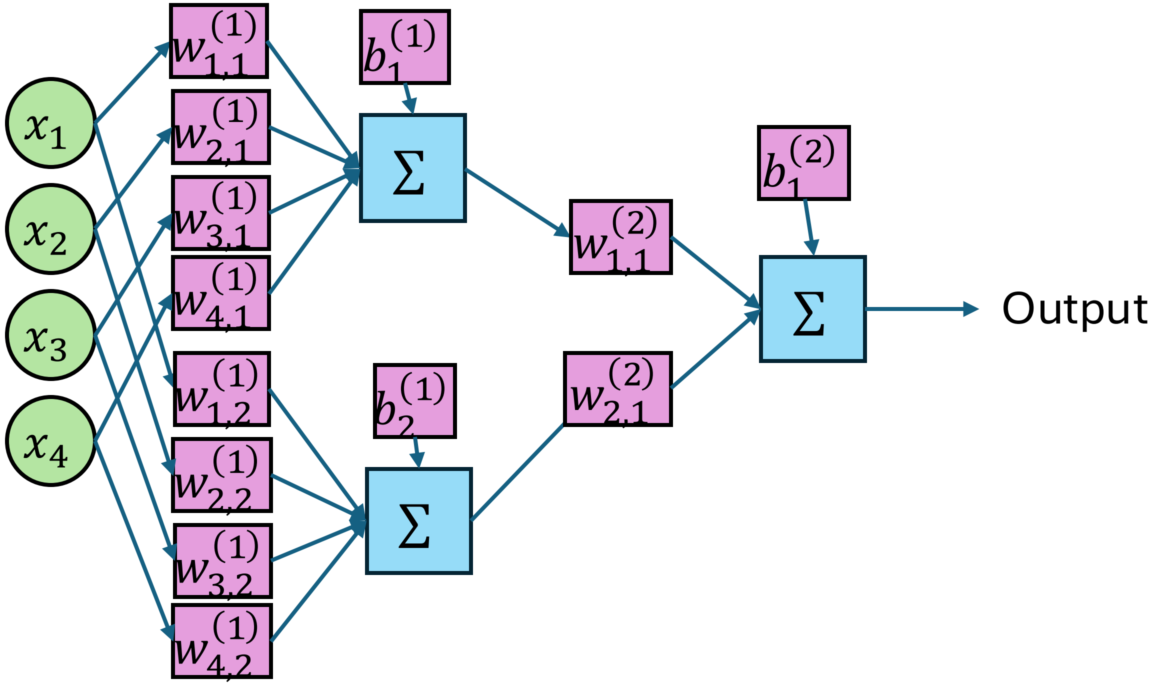
What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\hat{}2}{n}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$L = (10 - 7)^2$$

$$L = 9$$

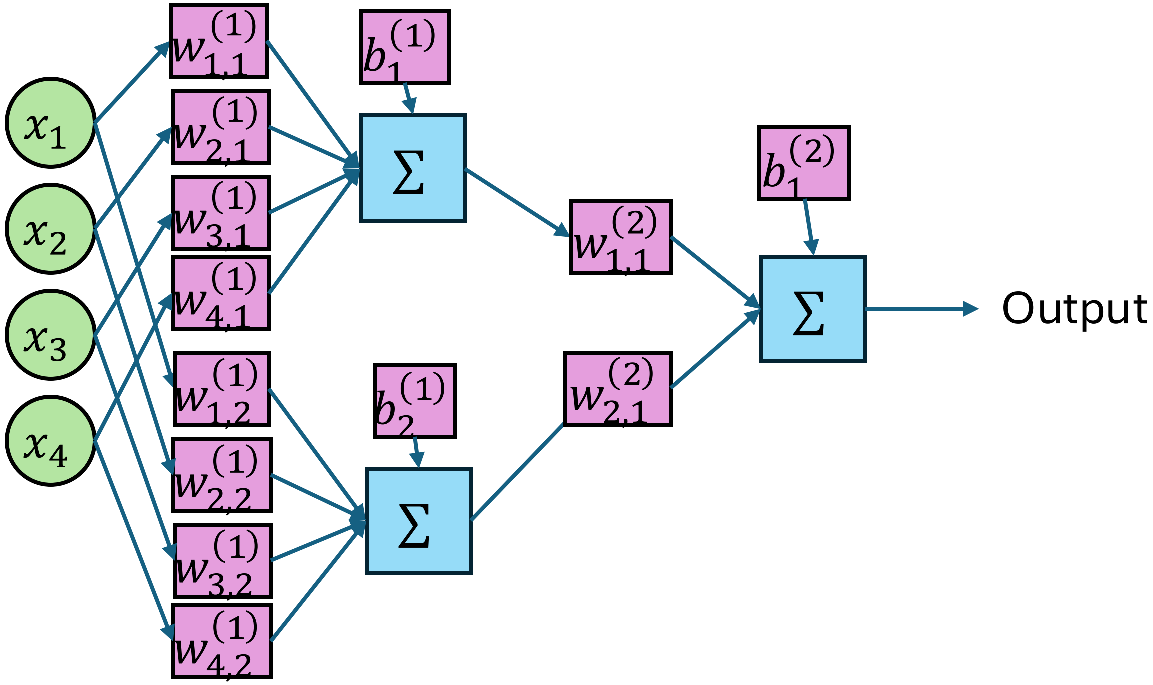


What is $\frac{dL}{db^{(2)}}$?

Backpropagation

Output, $z^{(2)}$ = $ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.



What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\text{^}2}{n}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$L = (10 - 7)^2$$

$$L = 9$$

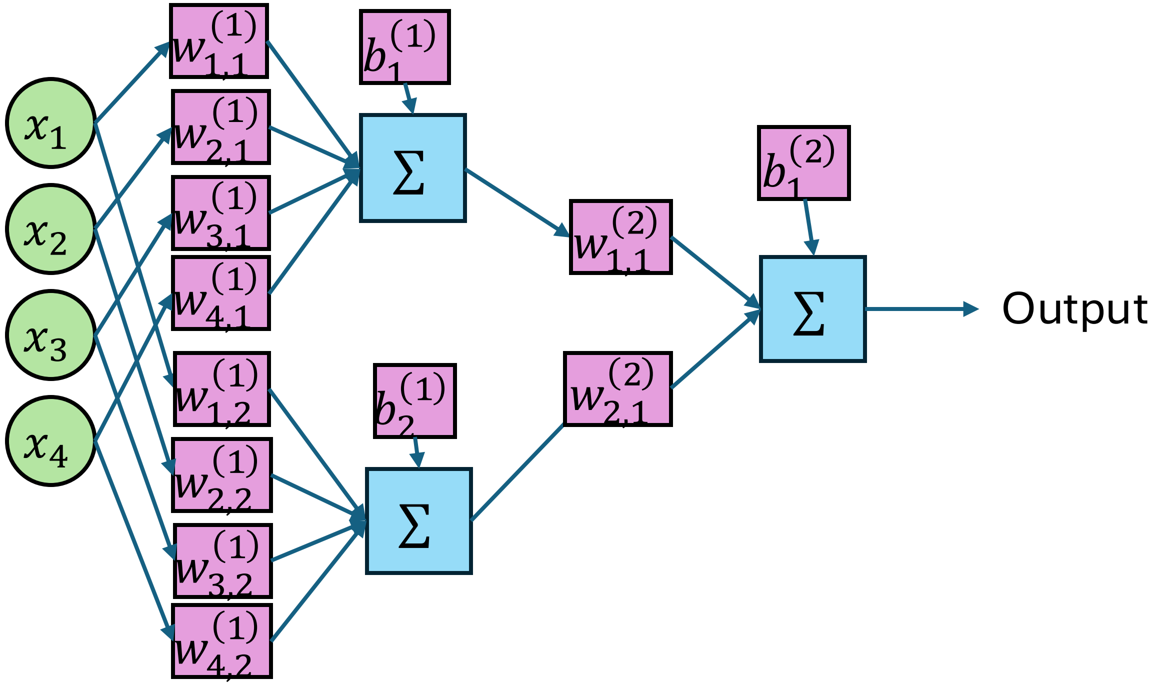
What is $\frac{dL}{db^{(2)}}$?

$$\frac{dL}{db^{(2)}} = \frac{dL}{dz^{(2)}} \cdot \frac{dz^{(2)}}{db^{(2)}} = 2(z^{(2)} - y) \cdot 1 = 6$$

Backpropagation

Output, $z^{(2)}$ = $ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.



What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\hat{}2}{n}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$L = (10 - 7)^2$$

$$L = 9$$

What is $\frac{dL}{db^{(2)}}$?

$$\frac{dL}{db^{(2)}} = \frac{dL}{dz^{(2)}} \cdot \frac{dz^{(2)}}{db^{(2)}} = 2(z^{(2)} - y) \cdot 1 = 6$$

What is $\frac{dL}{dw_{1,1}^{(1)}}$?

Backpropagation

Output, $z^{(2)} = ReLU\left(xW^{(1)} + b^{(1)}\right)W^{(2)} + b^{(2)}$

For a single example, suppose we get an output of 10, when the ground truth was 7.

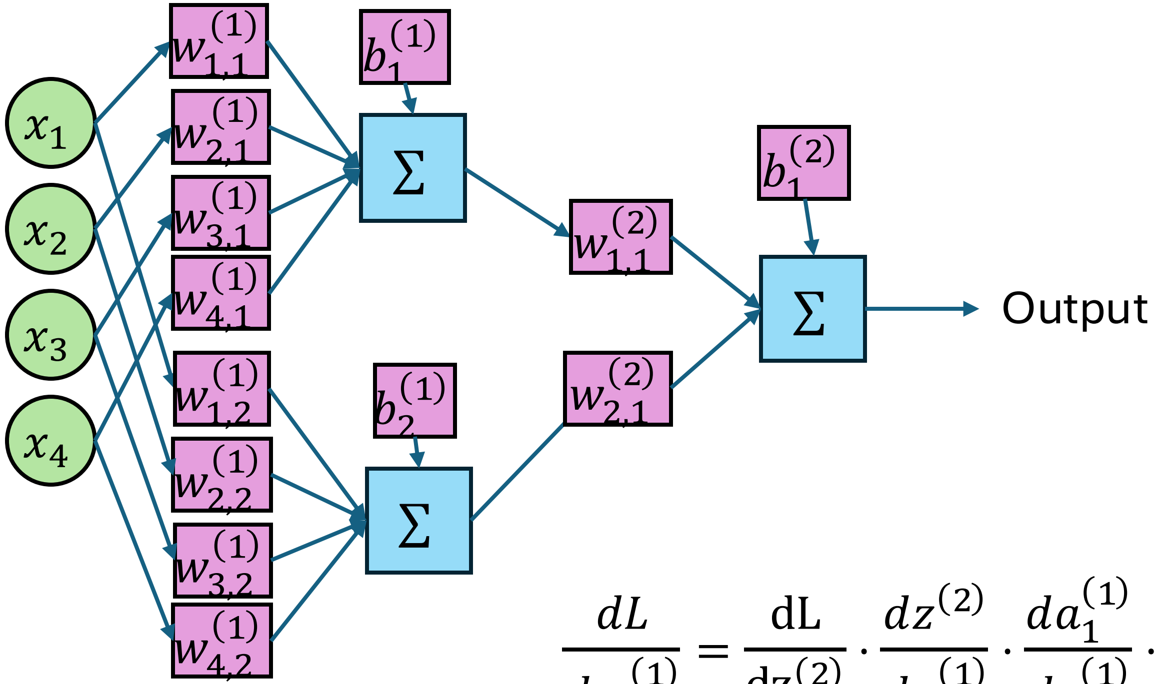
What is the mean squared error?

$$L = \frac{\sum_{i=1}^{n}(z^{(2)} - y^{(i)})\hat{}2}{n}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$L = (10 - 7)^2$$

$$L = 9$$



$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

What is $\frac{dL}{db^{(2)}}$?

$$\frac{dL}{db^{(2)}} = \frac{dL}{dz^{(2)}} \cdot \frac{dz^{(2)}}{db^{(2)}} = 2(z^{(2)} - y) \cdot 1 = 6$$

What is $\frac{dL}{dw_{1,1}^{(1)}}$?

Component-wise derivation

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

$$2\left(z^{(2)} - y\right)$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$2\left(z^{(2)} - y\right) \qquad w_{1,1}^{2}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$2\left(z^{(2)} - y\right)$ $w_{1,1}^{2}$ 0 if $z_1^{(1)}$<0,
1 otherwise

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$2\left(z^{(2)} - y\right)$ $w_{1,1}^2$ 0 if $z_1^{(1)}$<0,
1 otherwise x_1

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \underbrace{\frac{\mathrm{dL}}{\mathrm{dz}^{(2)}}}_{2\left(z^{(2)}-y\right)} \cdot \underbrace{\frac{dz^{(2)}}{da_1^{(1)}}}_{w_{1,1}^{2}} \cdot \underbrace{\frac{da_1^{(1)}}{dz_1^{(1)}}}_{\substack{0 \text{ if } z_1^{(1)}<0, \\ 1 \text{ otherwise}}} \cdot \underbrace{\frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}}_{x_1}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Matrix Form derivation

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

$2\left(z^{(2)} - y\right)$

$w_{1,1}^2$

0 if $z_1^{(1)}$<0,
1 otherwise

x_1

Matrix Form derivation

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

$2\left(z^{(2)} - y\right)$ $w_{1,1}^2$ 0 if $z_1^{(1)}$<0,
1 otherwise x_1

Matrix Form derivation

$$L = \left(z^{(2)} - y\right)^2$$

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \underbrace{\frac{\mathrm{dL}}{\mathrm{dz}^{(2)}}}_{2\left(z^{(2)}-y\right)} \cdot \underbrace{\frac{dz^{(2)}}{da_1^{(1)}}}_{w_{1,1}^{2}} \cdot \underbrace{\frac{da_1^{(1)}}{dz_1^{(1)}}}_{0 \text{ if } z_1^{(1)}<0,\ 1 \text{ otherwise}} \cdot \underbrace{\frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}}_{x_1}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Matrix Form derivation

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a^{(1)} W^{(2)} + b^{(2)}$$

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$2\left(z^{(2)} - y\right)$ $\quad w_{1,1}^{2}$ $\quad$ 0 if $z_1^{(1)}$<0, 1 otherwise $\quad x_1$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Matrix Form derivation

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a^{(1)} W^{(2)} + b^{(2)}$$

$$a^{(1)} = ReLU(z^{(1)})$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

$2\left(z^{(2)} - y\right)$

$w_{1,1}^2$

0 if $z_1^{(1)}$<0,
1 otherwise

x_1

Matrix Form derivation

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a^{(1)} W^{(2)} + b^{(2)}$$

$$a^{(1)} = ReLU(z^{(1)})$$

$$z^{(1)} = xW^{(1)} + b$$

$2\left(z^{(2)} - y\right)$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$2\left(z^{(2)} - y\right)$

$w_{1,1}^{2}$

0 if $z_1^{(1)}$<0,
1 otherwise

x_1

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Matrix Form derivation

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

$2\left(z^{(2)} - y\right)$

$W^{(2)\mathrm{T}}$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a^{(1)} W^{(2)} + b^{(2)}$$

$$a^{(1)} = ReLU(z^{(1)})$$

$$z^{(1)} = xW^{(1)} + b$$

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da_1^{(1)}} \cdot \frac{da_1^{(1)}}{dz_1^{(1)}} \cdot \frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

$2\left(z^{(2)} - y\right)$ $w_{1,1}^2$ 0 if $z_1^{(1)}$<0,
1 otherwise x_1

Matrix Form derivation

$$\frac{dL}{dW^{(1)}} = \frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \cdot \frac{dz^{(2)}}{da^{(1)}} \cdot \frac{da^{(1)}}{dz^{(1)}} \cdot \frac{dz^{(1)}}{dW^{(1)}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a^{(1)} W^{(2)} + b^{(2)}$$

$$a^{(1)} = ReLU(z^{(1)})$$

$$z^{(1)} = xW^{(1)} + b$$

$2\left(z^{(2)} - y\right)$

$W^{(2)\mathrm{T}}$ 0 if $z_1^{(1)}$<0,
1 otherwise

Component-wise derivation

$$\frac{dL}{dw_{1,1}^{(1)}} = \underbrace{\frac{\mathrm{dL}}{\mathrm{dz}^{(2)}}}_{2\left(z^{(2)}-y\right)} \cdot \underbrace{\frac{dz^{(2)}}{da_1^{(1)}}}_{w_{1,1}^{2}} \cdot \underbrace{\frac{da_1^{(1)}}{dz_1^{(1)}}}_{\substack{0 \text{ if } z_1^{(1)}<0, \\ 1 \text{ otherwise}}} \cdot \underbrace{\frac{dz_1^{(1)}}{dw_{1,1}^{(1)}}}_{x_1}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + \mathrm{b}^{(2)}$$

$$a_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_1^{(1)} = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,4}x_4 + b$$

Matrix Form derivation

$$\frac{dL}{dW^{(1)}} = \underbrace{\frac{\mathrm{dL}}{\mathrm{dz}^{(2)}}}_{2\left(z^{(2)}-y\right)} \cdot \underbrace{\frac{dz^{(2)}}{da^{(1)}}}_{W^{(2)\mathrm{T}}} \cdot \underbrace{\frac{da^{(1)}}{dz^{(1)}}}_{\substack{0 \text{ if } z_1^{(1)}<0, \\ 1 \text{ otherwise}}} \cdot \underbrace{\frac{dz^{(1)}}{dW^{(1)}}}_{\mathrm{x}^{\mathrm{T}}}$$

$$L = \left(z^{(2)} - y\right)^2$$

$$z^{(2)} = a^{(1)} W^{(2)} + b^{(2)}$$

$$a^{(1)} = ReLU(z^{(1)})$$

$$z^{(1)} = xW^{(1)} + b$$

Shape Help

Consider each step to be a function (which they are). What shapes are the input and outputs? Ther derivative of each function must have shape $\mathbb{R}^{\mathrm{in}\times\mathrm{out}}$, where in and out are the input and output dimensions for that function.

The Jacobian of a function with respect to an input matrix in $\mathbb{R}^{\mathrm{n}\times\mathrm{d}}$ must have the shape $\mathbb{R}^{\mathrm{n}\times\mathrm{d}}$!

Why? Recall that a derivative/gradient/Jacobian tells you if the function will increase/decrease with small changes to input. Each element of the Jacobian corresponds to a specific input value.

Multiple Examples

- The previous example took in one example to compute MSE
- The full gradient of the loss function is not determined by one example, but multiple examples!

What changes?

$$\frac{\mathrm{dL}}{\mathrm{dz}^{(2)}} \in \mathbb{R}^{\mathrm{n}\times 1}$$

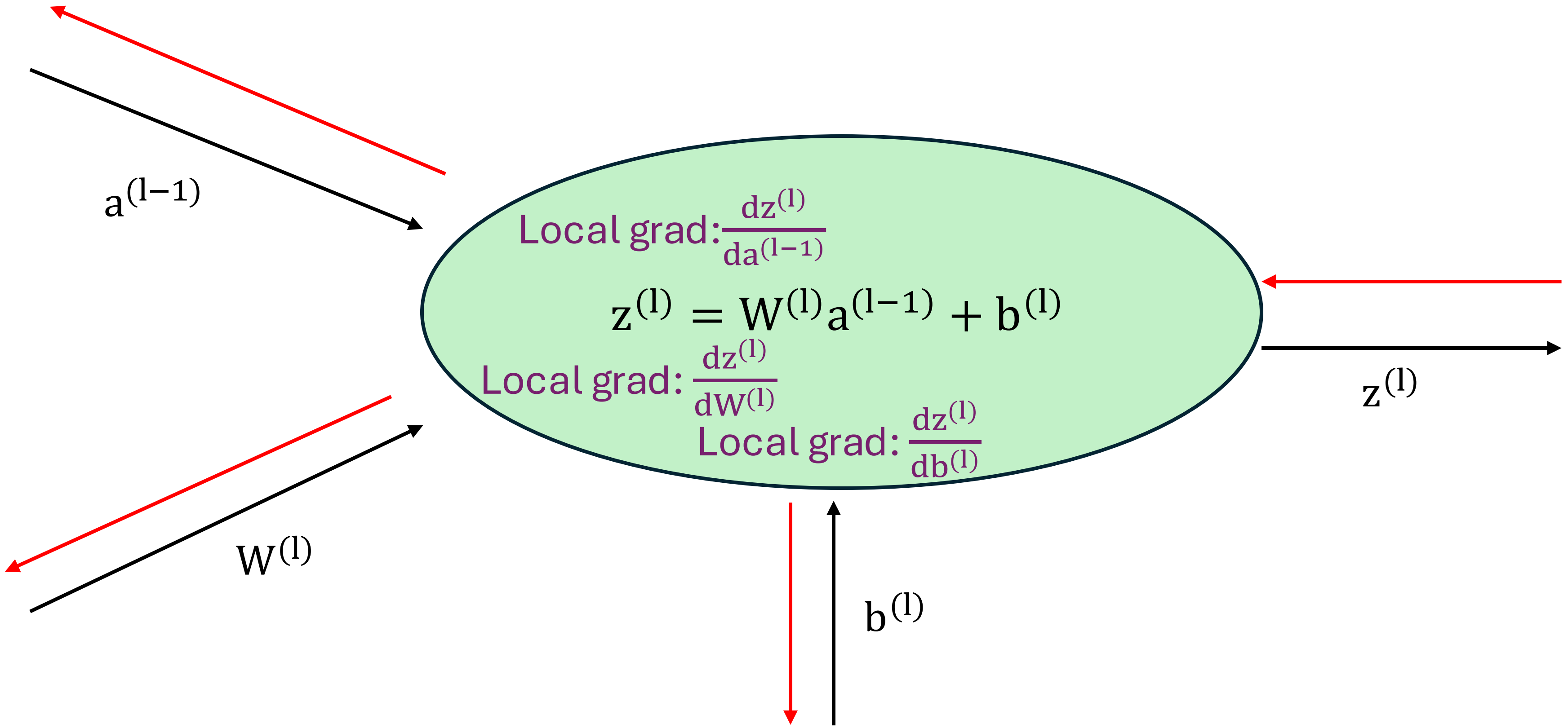
$\mathrm{X} \in \mathbb{R}^{\mathrm{n}\times\mathrm{d}}$ instead of $\mathrm{x} \in \mathbb{R}^{1\times\mathrm{d}}$

Many other Jacobians add a batch dimension for the n examples.

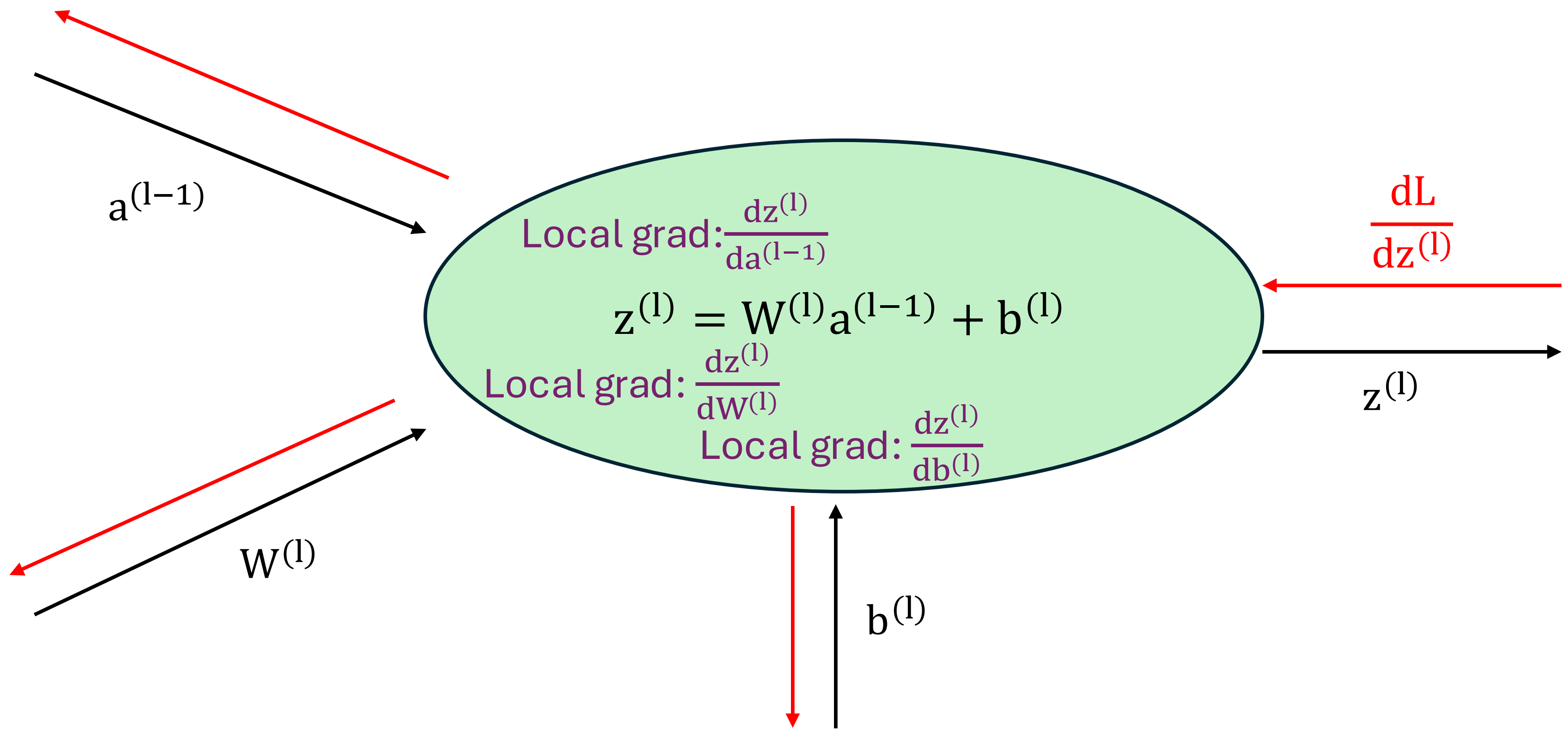
How many inputs and outputs does the term have?

Derivative shapes for parameters (weights and biases) never change. Why is that expected?

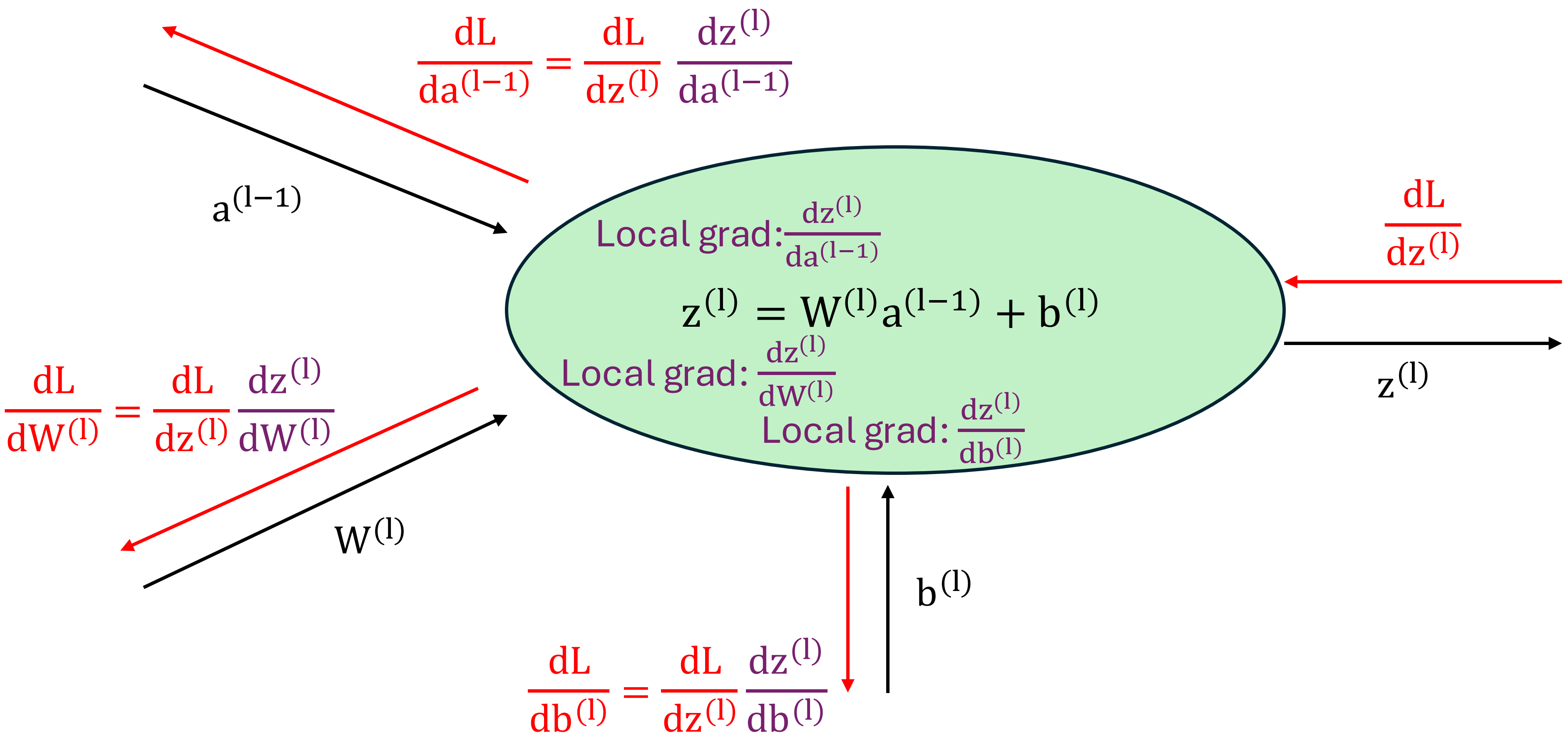
Compute Graph for a Single Layer



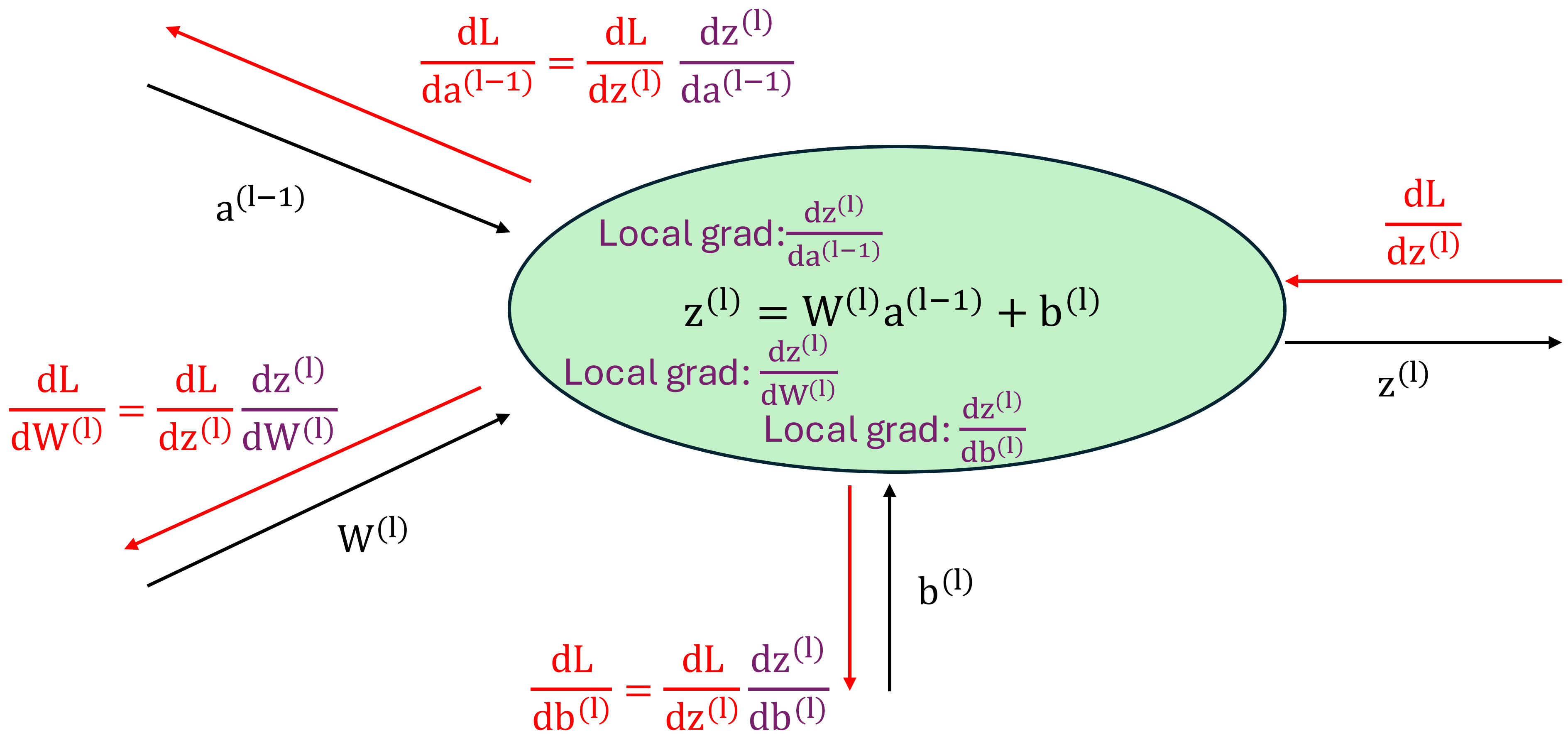
Compute Graph for a Single Layer



Compute Graph for a Single Layer



Compute Graph for a Single Layer



For each node:

Compose cumulative gradient $\frac{dL}{dz^{(l)}}$ with local gradients

Pass new cumulative gradient to parent nodes, repeat

That was a lot of math, let’s take a break

Weekly quiz is available on Gradescope!

Why should you care about compute graphs?

(This is much more of a common issue in pytorch than tensorflow)

```python
def train_with_memory_leak():
    running_loss = 0.0
    for epoch in range(100):
        for i, (inputs, targets) in enumerate(loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss

            if i % 10 == 9:
                print(f'Loss: {running_loss / 10}')
                running_loss = 0.0
```

Why should you care about compute graphs?

(This is much more of a common issue in pytorch than tensorflow)

```
def train_with_memory_leak():
    running_loss = 0.0
    for epoch in range(100):
        for i, (inputs, targets) in enumerate(loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss

            if i % 10 == 9:
                print(f'Loss: {running_loss / 10}')
                running_loss = 0.0
```

Running loss' compute graph will contain the compute graph of loss!

Why should you care about compute graphs?

(This is much more of a common issue in pytorch than tensorflow)

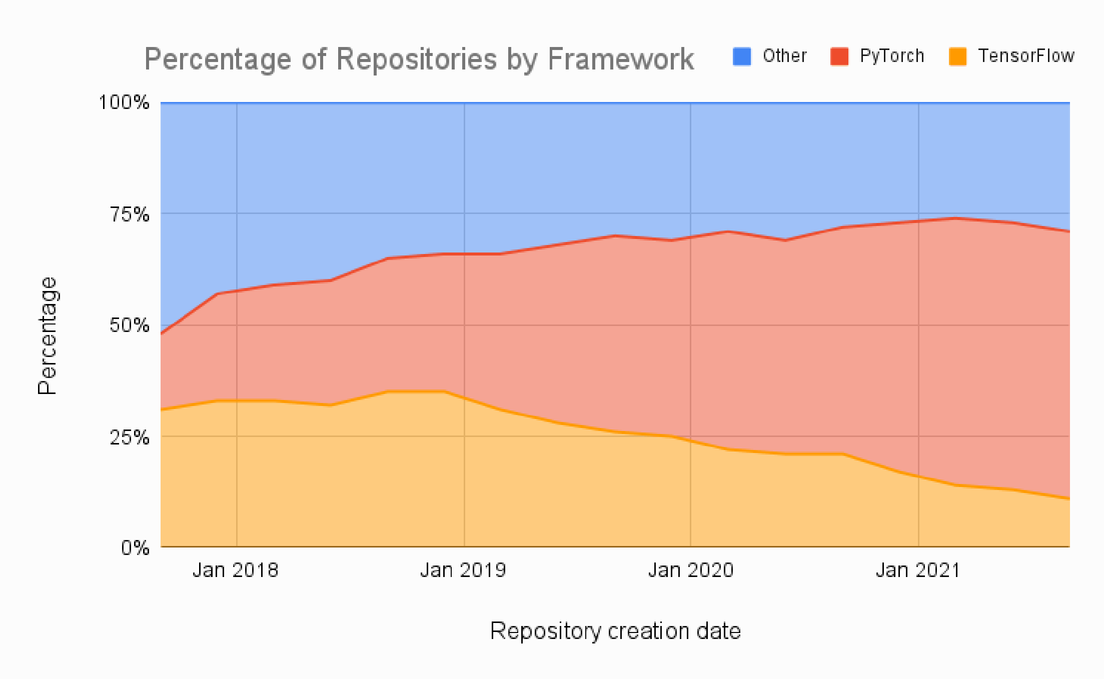
```python
def train_with_memory_leak():
    running_loss = 0.0
    for epoch in range(100):
        for i, (inputs, targets) in enumerate(loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss

            if i % 10 == 9:
                print(f'Loss: {running_loss / 10}')
                running_loss = 0.0
```

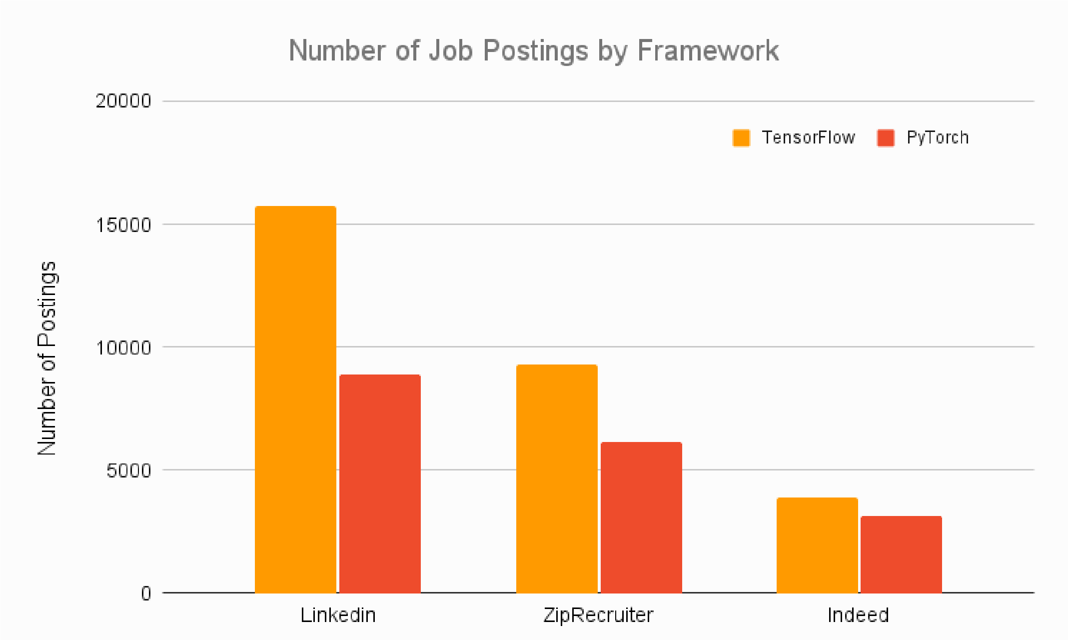
Running loss' compute graph will contain the compute graph of loss!

The memory required to store running_loss will only ever increase!

DL Frameworks

- Main current frameworks are Tensorflow, Pytorch, and Jax
- TF and torch are becoming increasingly similar in style and performance
- Jax is new and different





https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/

Tensorflow

Tensorflow

- Developed and maintained by Google

Tensorflow

- Developed and maintained by Google
- In addition to autodiff features it also provides:
  - Many common functions (i.e., Softmax, Sigmoid, Cross Entropy, etc.)
  - An easy way to train models (**Keras**)
  - Strong support for hardware acceleration (i.e., if you have a GPU, TF will figure out how to use it)

Tensorflow

- Developed and maintained by Google
- In addition to autodiff features it also provides:
  - Many common functions (i.e., Softmax, Sigmoid, Cross Entropy, etc.)
  - An easy way to train models (Keras)
  - Strong support for hardware acceleration (i.e., if you have a GPU, TF will figure out how to use it)
- “Easier to deploy to production” (has been the general consensus previously, but other frameworks have caught up)

Tensorflow

- Developed and maintained by Google
- In addition to autodiff features it also provides:
  - Many common functions (i.e., Softmax, Sigmoid, Cross Entropy, etc.)
  - An easy way to train models (Keras)
  - Strong support for hardware acceleration (i.e., if you have a GPU, TF will figure out how to use it)
- “Easier to deploy to production” (has been the general consensus previously, but other frameworks have caught up)
- TF lite for on device applications (e.g., phones)

Pytorch

Pytorch

- Developed by Facebook AI (now Meta)

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.
- Slightly more work to use GPUs or other hardware

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.
- Slightly more work to use GPUs or other hardware
- Harder to track stats
  - (I still use TF’s tensorboard stat tracker when using Pytorch)

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.
- Slightly more work to use GPUs or other hardware
- Harder to track stats
  - (I still use TF’s tensorboard stat tracker when using Pytorch)
- Easier to learn and use than tensorflow
  - Better error reporting, training code is harder to write but easier to debug

Jax

Jax

- Also developed by Google…

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow
- ***Much Faster***

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow
- ***Much Faster***
- Takes advantage of Just In Time (JIT) compiling to speed up execution

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow
- ***Much Faster***
- Takes advantage of Just In Time (JIT) compiling to speed up execution
- Functional programming paradigm

Improving Gradient Descent

Improving Gradient Descent

Computing the full gradient for a large dataset takes a very long time and it often will not fit in memory, slowing it down even further

Improving Gradient Descent

Computing the full gradient for a large dataset takes a very long time and it often will not fit in memory, slowing it down even further

Solution: Approximate the gradient by sampling a selection of examples (i.e., a batch). Run a gradient descent step with that batch

Stochastic Gradient Descent

For N epochs:

sample a batch B from dataset X

compute predictions and loss function

compute gradient

update weights with small step in direction of negative grad.

Stochastic Gradient Descent

For N epochs:

sample a batch B from dataset X

compute predictions and loss function

compute gradient

update weights with small step in direction of negative grad.

Training is non-deterministic because batches are sampled randomly from dataset

Stochastic Gradient Descent

For N epochs:

sample a batch B from dataset X

compute predictions and loss function

compute gradient

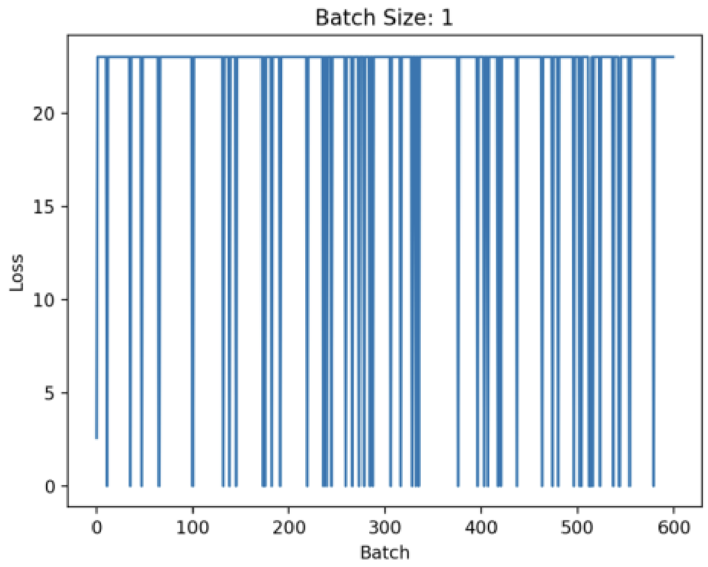
update weights with small step in direction of negative grad.

Training is non-deterministic because batches are sampled randomly from dataset

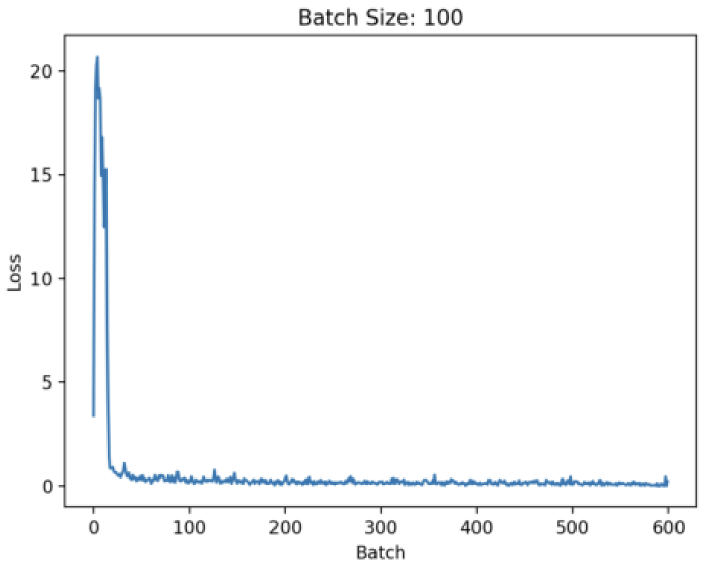
Why does this work? The expectation of the gradient is equal to the gradient itself!

What size should the batch be?

Small batch size:
Fast, jittery updates



Large batch size:
Slower, stable updates



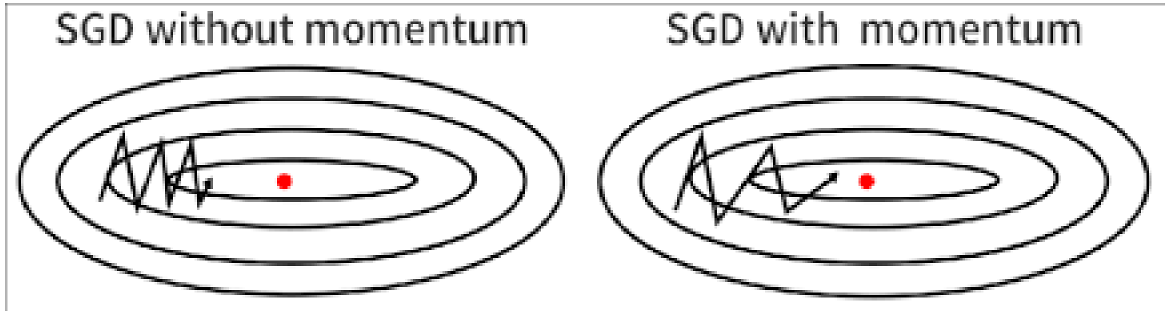
- Empirically, modern optimizers can handle larger batch size well
- Try to pick the largest batch size you can fit on your GPU!

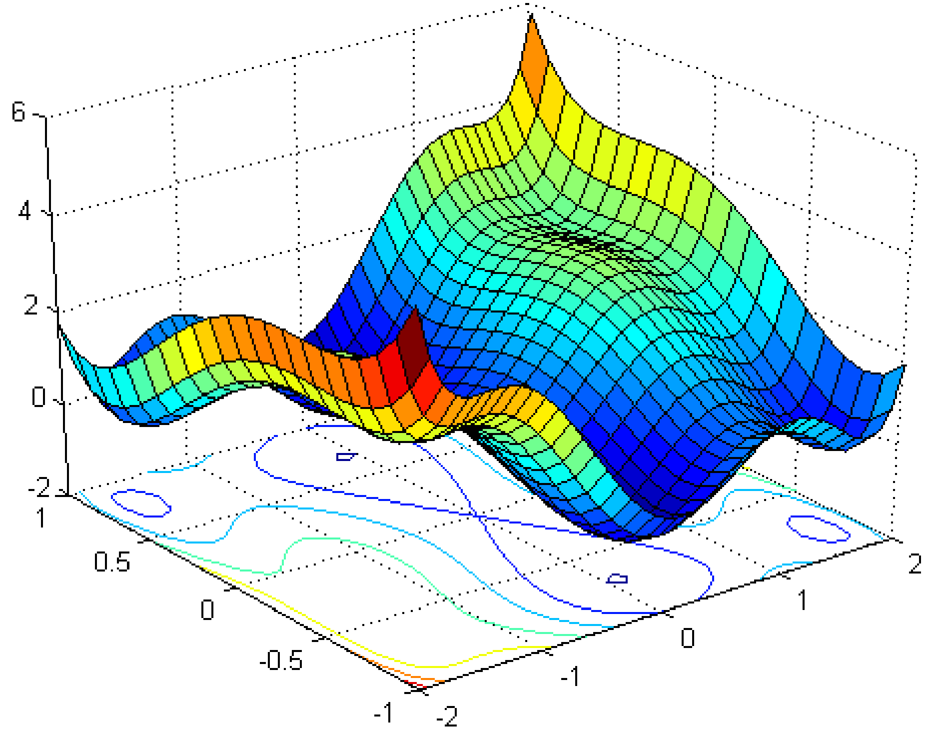
Further Improvements

$$\Delta w := \alpha \Delta w - \eta \nabla Q_i(w)$$
$$w := w + \Delta w$$

If gradient descent is like a ball rolling down a hill… What is that ball's mass?

SGD can be further improved by adding momentum term

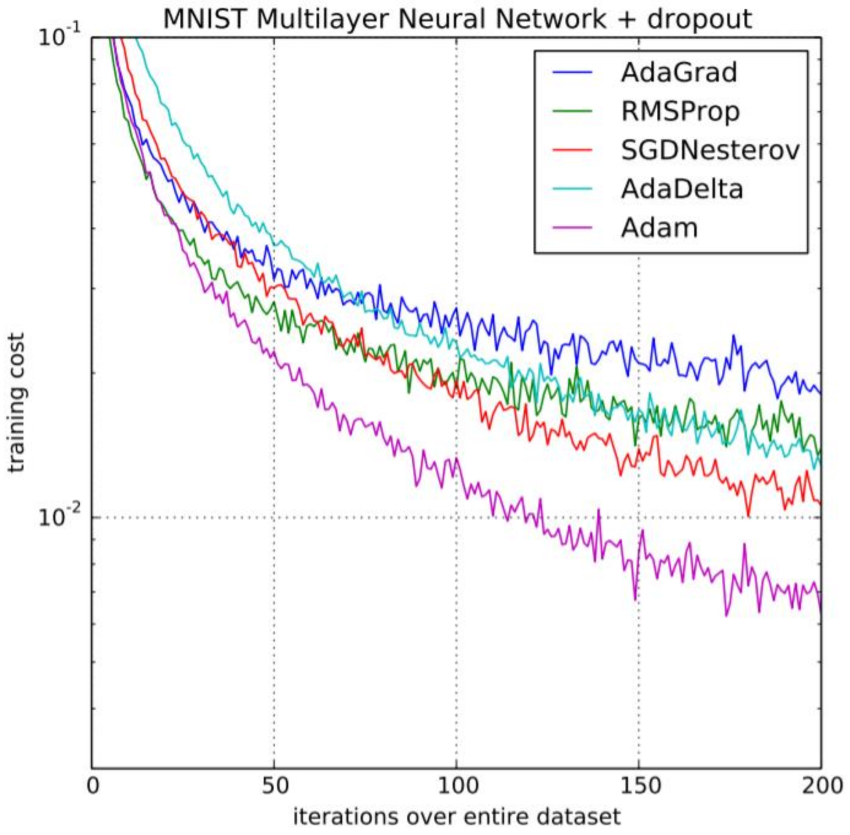




AdaM: SGD + Adaptive Momentum

Generally recommended as the best performing and easiest to use optimizer!

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)



Classification

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\, Correct}{n}$

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

The Issue: most of the time, the gradient of this loss function is $\nabla \mathrm{L}_{\theta} = 0$

Classification

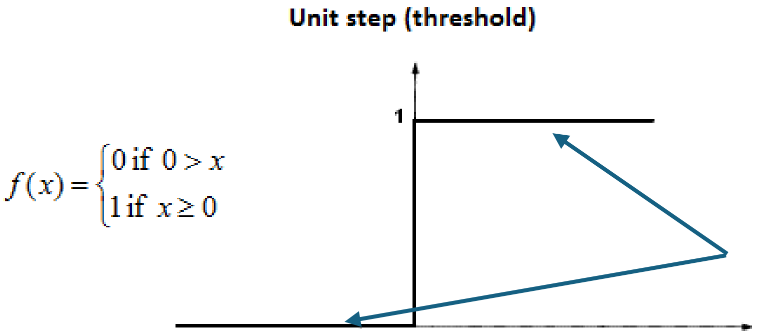
In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

The Issue: most of the time, the gradient of this loss function is $\nabla L_\theta = 0$

Gradient is only non-zero when changing a θ has an impact on output predictions



0 gradient everywhere except x=0

X=0 is not differentiable, but it does have a sub-gradient

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

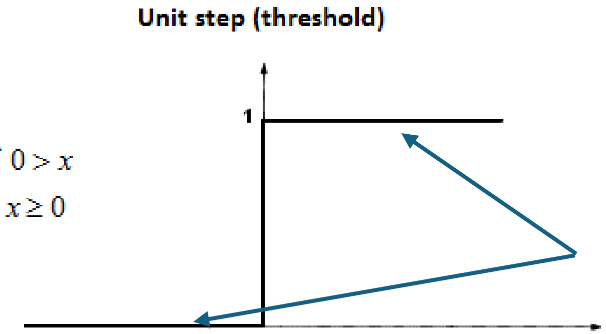
Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

The Issue: most of the time, the gradient of this loss function is $\nabla L_\theta = 0$

Gradient is only non-zero when changing a θ has an impact on output predictions

We cannot use classification as a loss function because it is incompatible with gradient descent. **Understanding Gradients is key to understanding all decisions related to neural networks!**

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



0 gradient everywhere except x=0

X=0 is not differentiable, but it does have a sub-gradient

What is a reasonable loss function to use?

- Accuracy is a “hard” function
  - Hard to take meaningful derivatives of
- Other examples:
  - Max vs. Softmax
  - Ranking vs Softrank
  - Sign function (i.e., perceptron activation) vs. Softsign
  - Argmax

What is a reasonable loss function to use?

- Accuracy is a “hard” function
  - Hard to take meaningful derivatives of
- Other examples:
  - Max vs. Softmax
  - Ranking vs Softrank
  - Sign function (i.e., perceptron activation) vs. Softsign
  - Argmax

My (somewhat) old research

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \, \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

Defined for two probability distributions, P and Q

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \, \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

Defined for two probability distributions, P and Q

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Think of Q as what we predict and P as the ground truth Probabilities

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

Defined for two probability distributions, P and Q

When P(x) is high, Q(x) should also be high… (Log(1) = 0)

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Think of Q as what we predict and P as the ground truth Probabilities

One-Hot Vectors Revisited

datagy.io

Island	Biscoe	Dream	Torgensen
Biscoe	1	0	0
Torgensen	0	0	1
Dream	0	1	0

One-Hot Vectors Revisited

datagy.io

Island	Biscoe	Dream	Torgensen
Biscoe	1	0	0
Torgensen	0	0	1
Dream	0	1	0

Can be interpreted as a probability!

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \, \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

"Ground truth" Probability (i.e., label)

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

“Ground truth” Probability (i.e., label)

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Predicted probabilities

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

"Ground truth" Probability (i.e., label)

P(x) is 1 if x is correct label, 0 otherwise

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \, \log\left(\frac{P(x)}{Q(x)}\right)$$

Predicted probabilities

Binary Cross Entropy

KL Divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \, \log\left(\frac{P(x)}{Q(x)}\right)$$

Cross Entropy (CE)

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

Binary Cross Entropy

KL Divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Cross Entropy (CE)

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

”Categorical Cross Entropy”

Binary Cross Entropy

KL Divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Cross Entropy (CE)

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

”Categorical Cross Entropy”

For Binary problems “Binary Cross Entropy” (BCE)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}} \sum_{i}^{n} y_i \log \hat{y}_i$$

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}}\sum_{i}^{n} y_i \log \hat{y}_i$$

What is this? (vector, scalar, matrix)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}}\sum_{i}^{n} y_i \log \hat{y}_i$$

What is this? (vector, scalar, matrix)

What is this? (vector, scalar, matrix)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}}\sum_{i}^{n} y_i \log \hat{y}_i$$

What is this? (vector, scalar, matrix)

What is this? (vector, scalar, matrix)

What is this? (vector, scalar, matrix)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}}\sum_{i}^{n} y_i \log \hat{y}_i$$

$$\frac{dL}{d\hat{y}} = -\sum_{i}^{n} -\frac{1}{p_i}$$

Probability of predicting correct label for example i

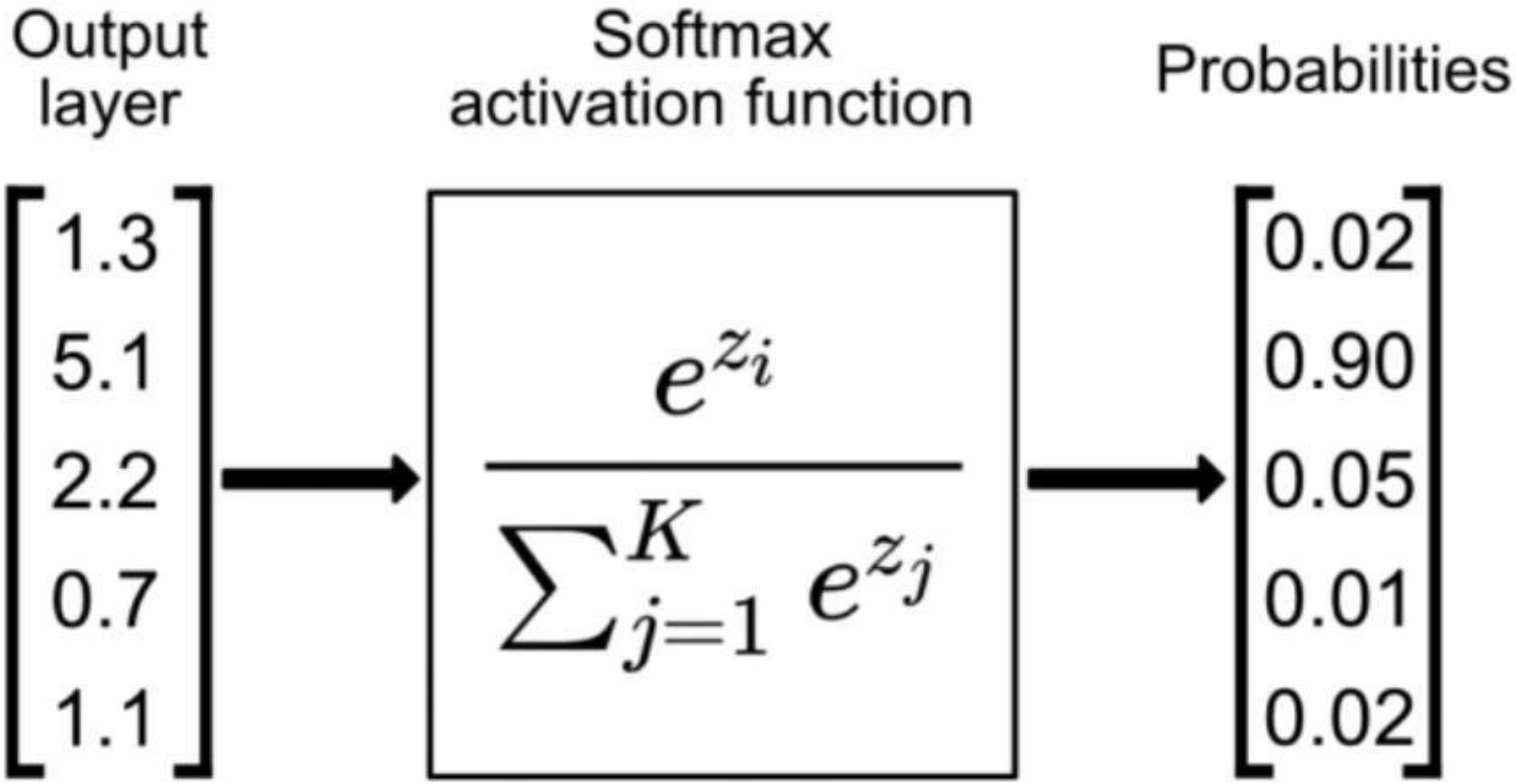
Probabilities

- If we have probabilities, we can use Cross Entropy
- How do we get probabilities?

Option #1: Normalize outputs (i.e., divide by their total)

Option #2: Use another function (i.e., softmax)

Softmax Function



Source: https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with normalization?

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with normalization?

[1/3, 2/3] for both examples

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with Softmax?

What’s the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with Softmax?

[0.26, 0.73] for [1, 2]
[0.00005, 0.99995] for [10, 20]

What’s the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Normalization?

What's the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Normalization?

[0.47, 0.53] for [11, 12]
[0.4, 0.6] for [20, 30]

What’s the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Softmax?

What’s the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Softmax?

[0.26, 0.73] for [11, 12]
[0.00005, 0.99995] for [20, 30]
Exactly the same as [1, 2] and [10, 20]

What’s the difference?

What's the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

What’s the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

Softmax is sensitive to multiplicative changes, but not additive

What’s the difference?

- Normalization is sensitive to additive changes, but not multiplicative changes

- Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

What’s the difference?

- Normalization is sensitive to additive changes, but not multiplicative changes

- Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

- \- Tends to handle smaller probabilities better (less float underflow)

What's the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

- Tends to handle smaller probabilities better (less float underflow)
- Remember that log in our loss function? Remember the e^z in softmax? Our loss function becomes ~linear for our neuron outputs z

What’s the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

- Tends to handle smaller probabilities better (less float underflow)
- Remember that log in our loss function? Remember the e^z in softmax? Our loss function becomes ~linear for our neuron outputs z
- Maybe has issues with overflow… (outputs can become inf or NaN)