University of Colorado **Boulder**

Department of Computer Science

CSCI 5622: Machine Learning

Chenhao Tan

# Lecture 13: Back propagation

Slides adapted from Chris Ketelsen, Jordan Boyd-Graber, and Noah Smith

# Administrivia

- Exam on Wednesday

# Learning Objectives

- Understanding back propagation
- Understanding the training algorithm for neural networks

# Outline

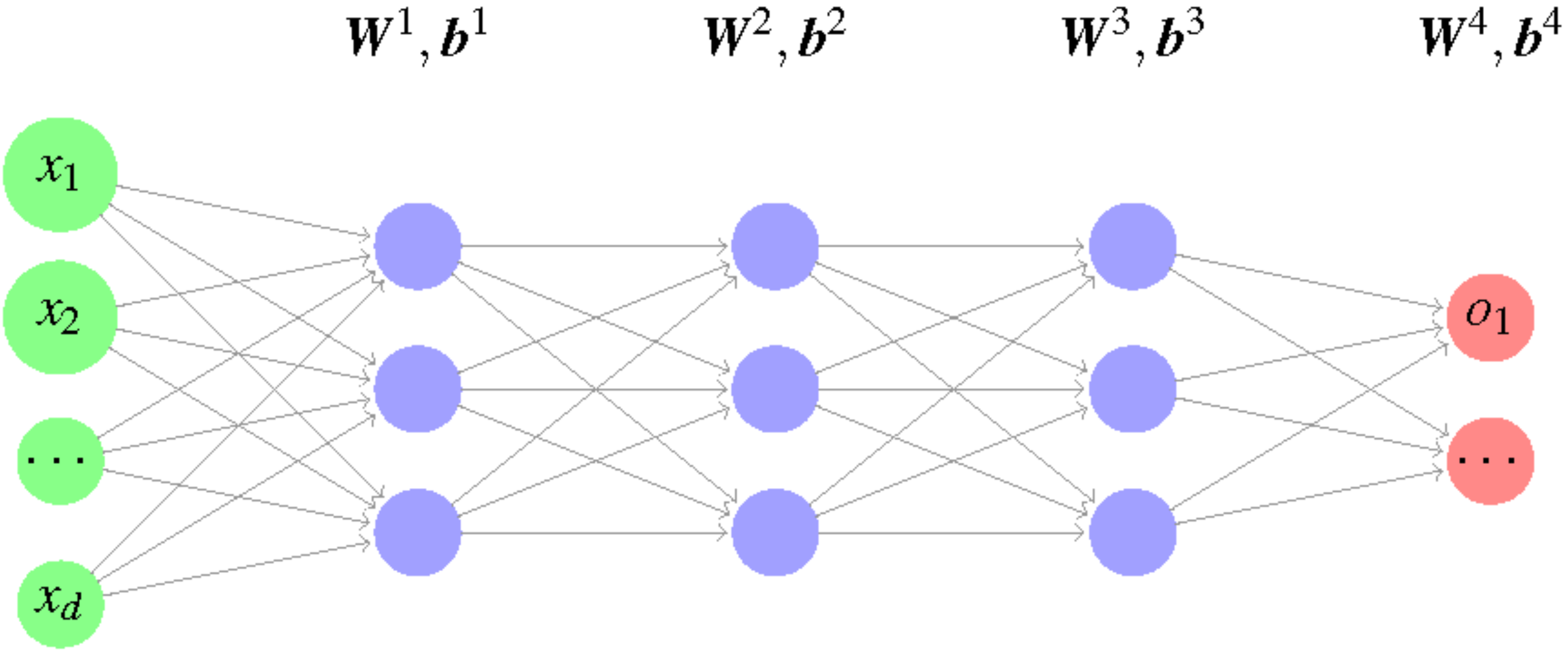
- Forward propagation recap
- Back propagation
- Practical issues of back propagation

# Outline

- Forward propagation recap
- Back propagation
- Practical issues of back propagation

## Forward propagation algorithm

How do we make predictions based on a multi-layer neural network?
Store the biases for layer $l$ in $\boldsymbol{b}^l$, weight matrix in $\boldsymbol{W}^l$



## Forward propagation algorithm

Suppose your network has $L$ layers

Make prediction for an instance $\boldsymbol{x}$

Initialize $\boldsymbol{a}^0 = \boldsymbol{x}$

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

  $\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$

  $\boldsymbol{a}^l = g(\boldsymbol{z}^l)$ // $g$ represents the nonlinear activation

**end for**

The prediction $\hat{y}$ is simply $\boldsymbol{a}^L$

## Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_w(\boldsymbol{x})$$

$$\boldsymbol{W}^l, \boldsymbol{b}^l, l = 1, \ldots, L$$

- Loss function (objective function)

$$\mathscr{L}(y, \hat{y})$$

- How do we learn the parameters?

## Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_W(\boldsymbol{x})$$

$$\boldsymbol{W}^l, \boldsymbol{b}^l, l = 1, \ldots, L$$

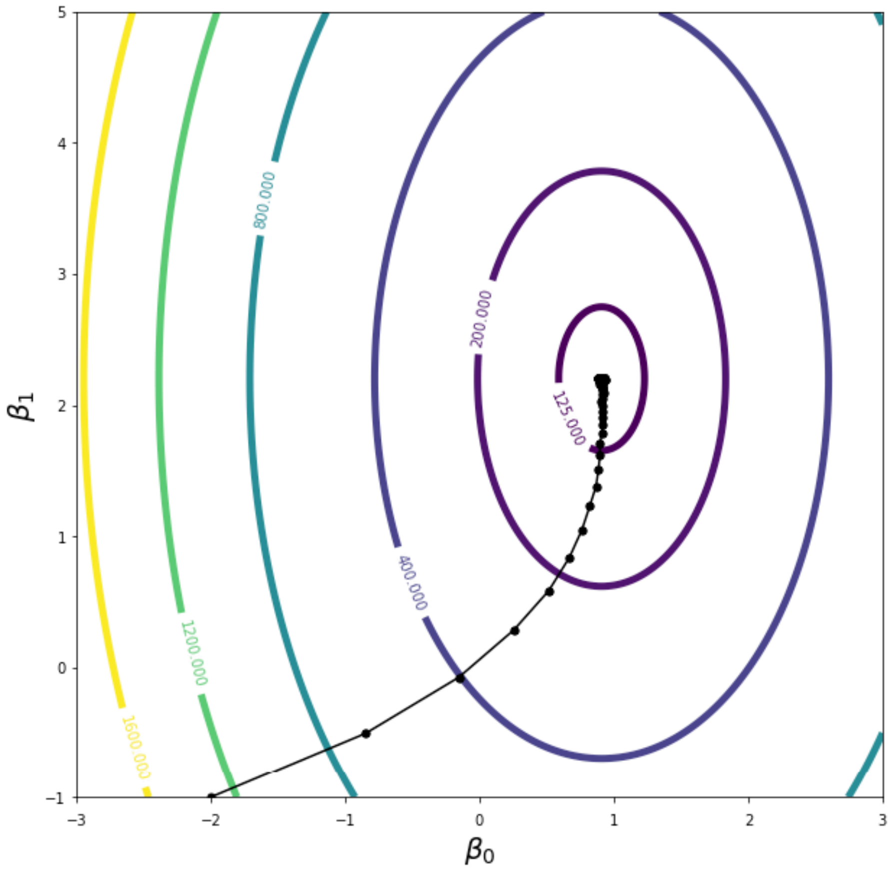
- Loss function (objective function)

$$\mathscr{L}(y, \hat{y})$$

- How do we learn the parameters?
  Stochastic gradient descent,

$$\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \frac{\partial \mathscr{L}(y, \hat{y})}{\partial \boldsymbol{W}^l}$$

# Reminder of gradient descent



## Challenge

- **Challenge**: How do we compute derivatives of the loss function with respect to weights and biases?
- **Solution**: Back propagation

# Outline

- Forward propagation recap
- Back propagation
- Practical issues of back propagation

## The Chain Rule

The chain rule allows us to take derivatives of nested functions.

## The Chain Rule

The chain rule allows us to take derivatives of nested functions.
**Univariate chain rule**:

$$\frac{d}{dx} f(g(x)) = f'(g(x))\, g'(x) = \frac{df}{dg}\frac{dg}{dx}$$

## The Chain Rule

The chain rule allows us to take derivatives of nested functions.

**Univariate chain rule**:

$$\frac{d}{dx} f(g(x)) = f'(g(x))\, g'(x) = \frac{df}{dg}\frac{dg}{dx}$$

Example:

$$\frac{d}{dx}\, \frac{1}{1+\exp(-x)}$$

## The Chain Rule

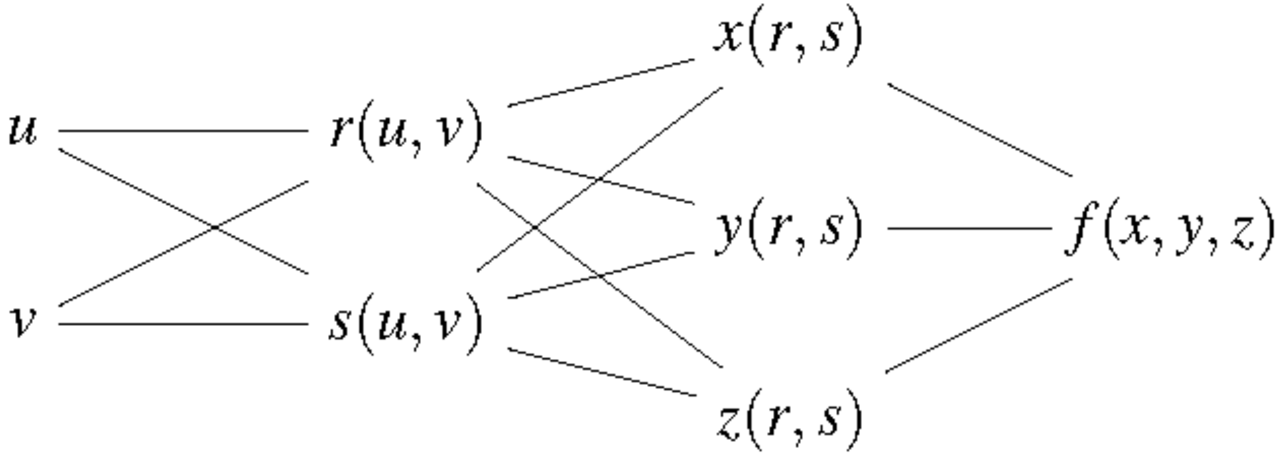
The chain rule allows us to take derivatives of nested functions.
**Univariate chain rule**:

$$\frac{d}{dx} f(g(x)) = f'(g(x))\, g'(x) = \frac{df}{dg}\frac{dg}{dx}$$

Example:
$$\frac{d}{dx} \frac{1}{1+\exp(-x)} = -\frac{1}{(1+\exp(-x))^2} \cdot \exp(-x) \cdot -1$$

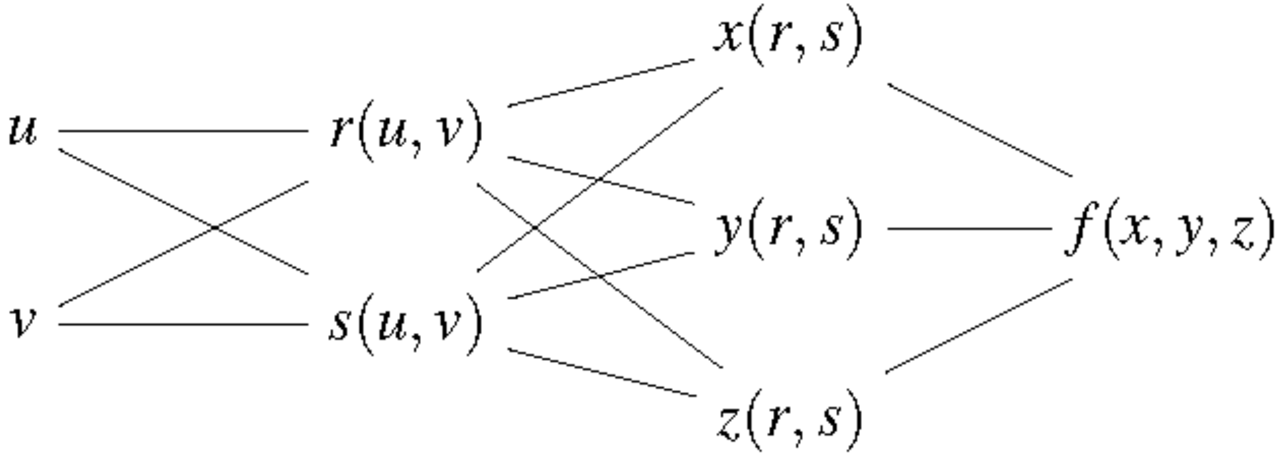
$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

**Multivariate chain rule**:



## The Chain Rule

**Multivariate chain rule**:



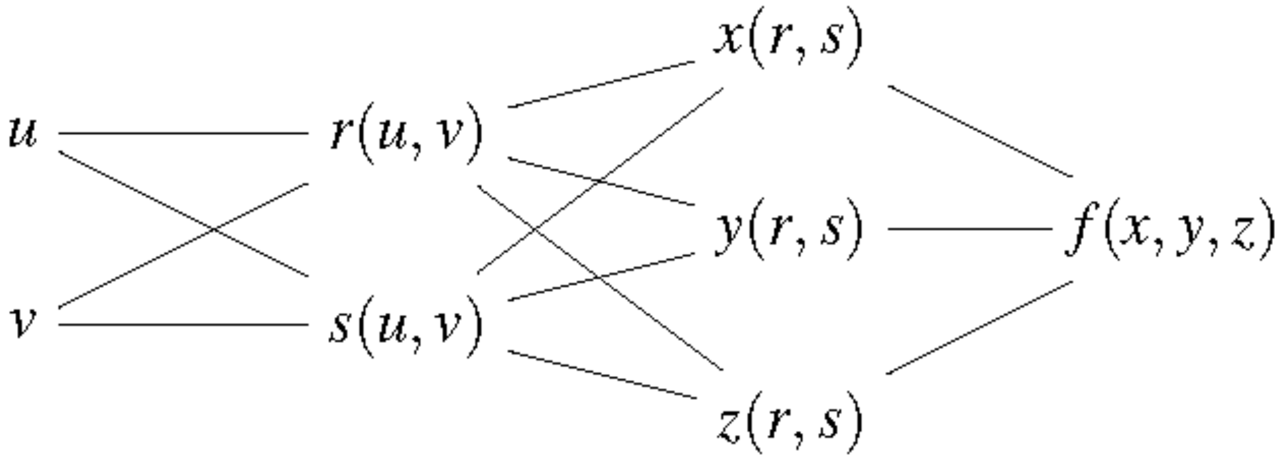

Derivative of $\mathscr{L}$ with respect to $x$:

$$\frac{\partial f}{\partial x}$$

Similarly, $\frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

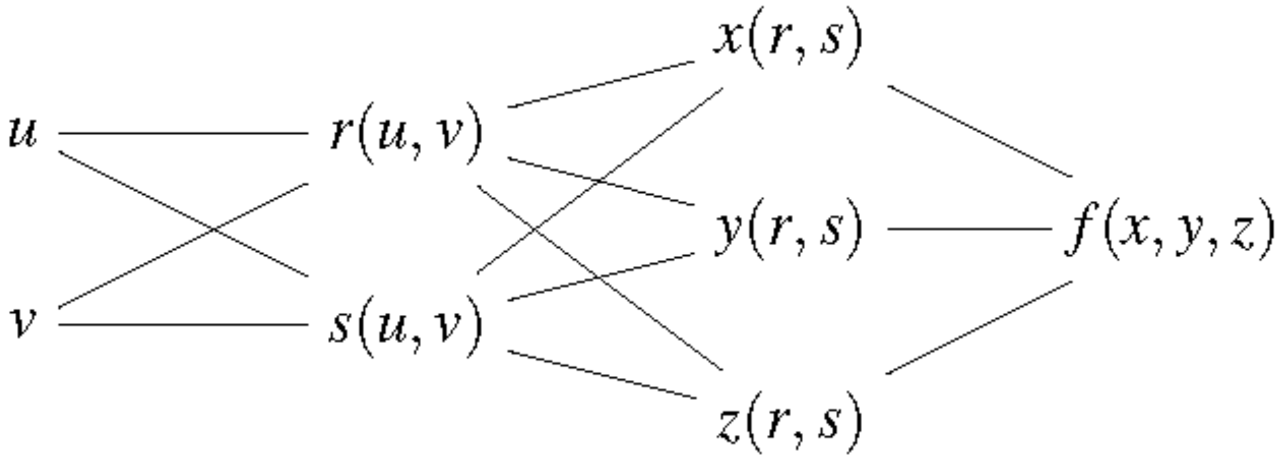
## The Chain Rule

What is the derivative of $f$ with respect to $r$?



## The Chain Rule

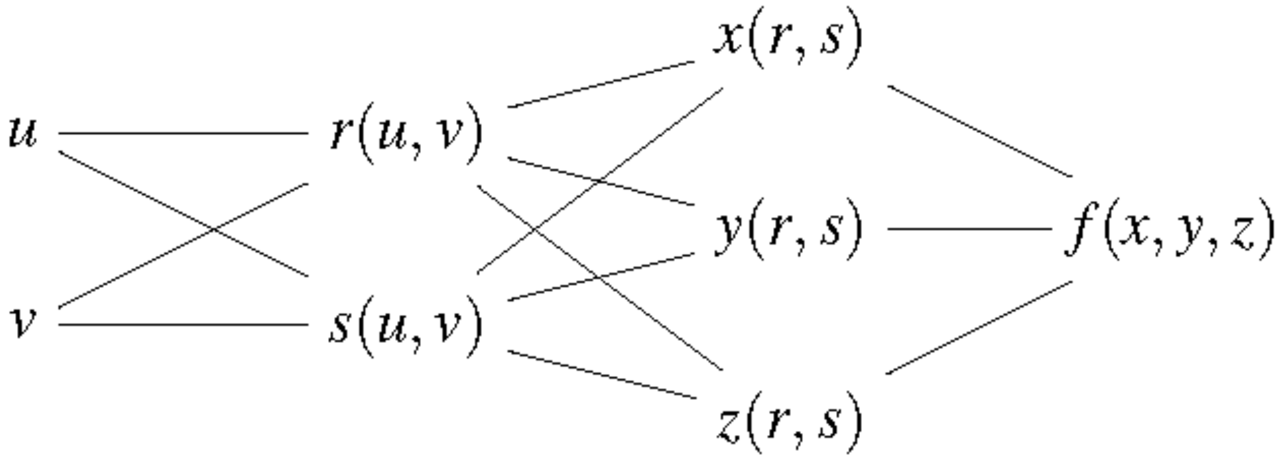
What is the derivative of $f$ with respect to $r$?



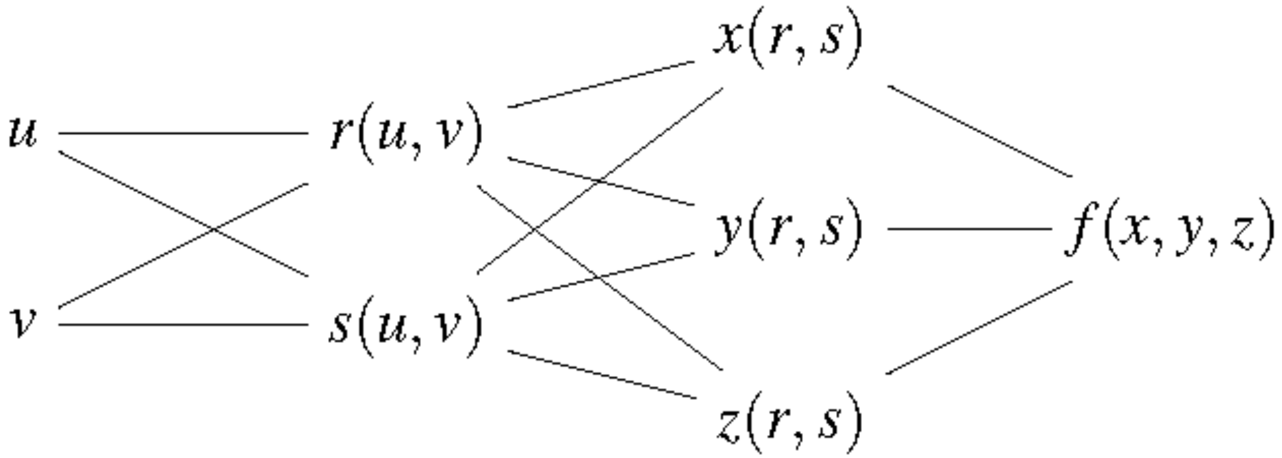

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial r}$$

## The Chain Rule

What is the derivative of $f$ with respect to $s$?



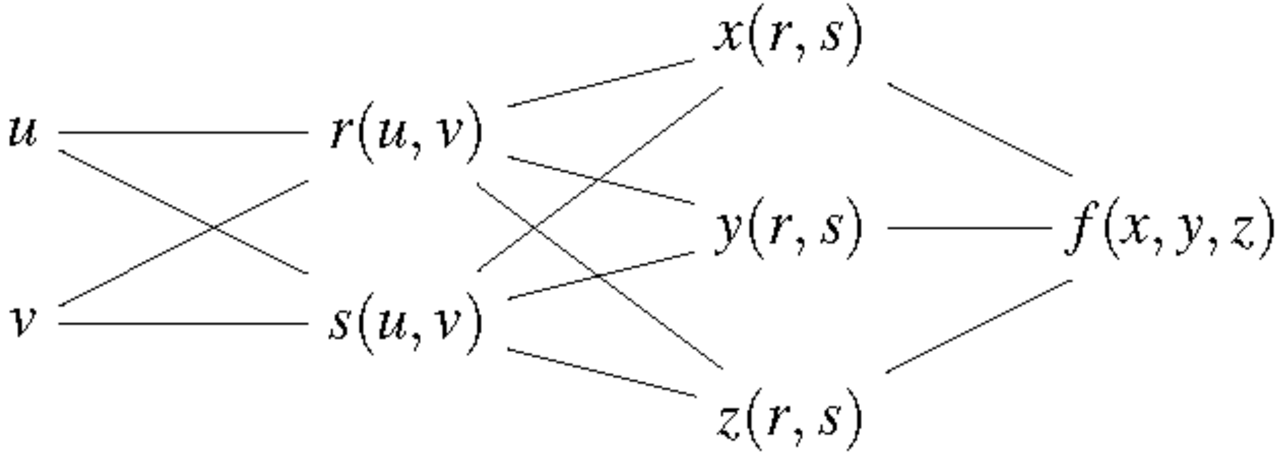
What is the derivative of $f$ with respect to $s$?




$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial s}$$

## The Chain Rule

What is the derivative of $f$ with respect to $s$?

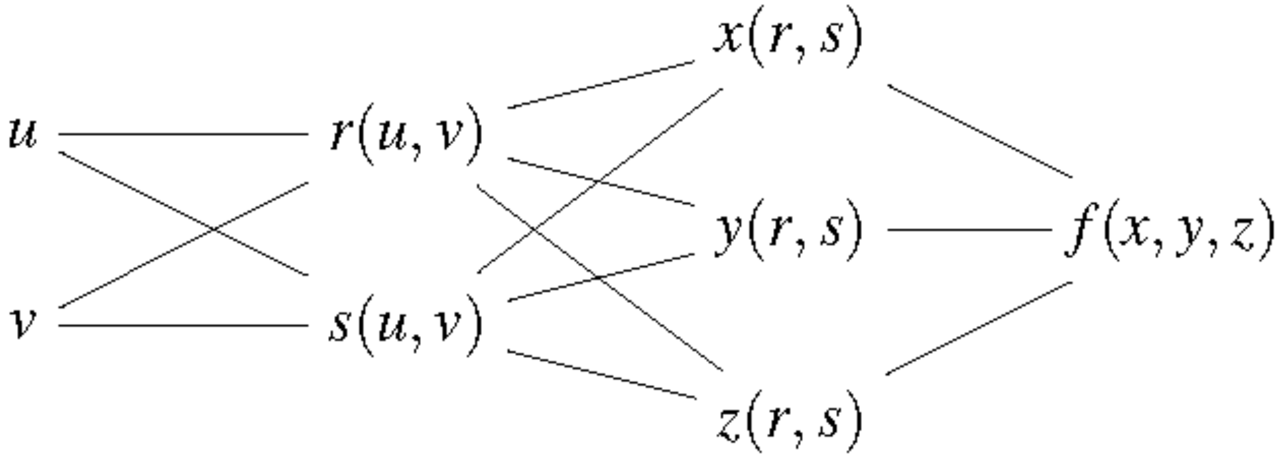



**Example**: Let $f = xyz$ , $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial s}$$

## The Chain Rule

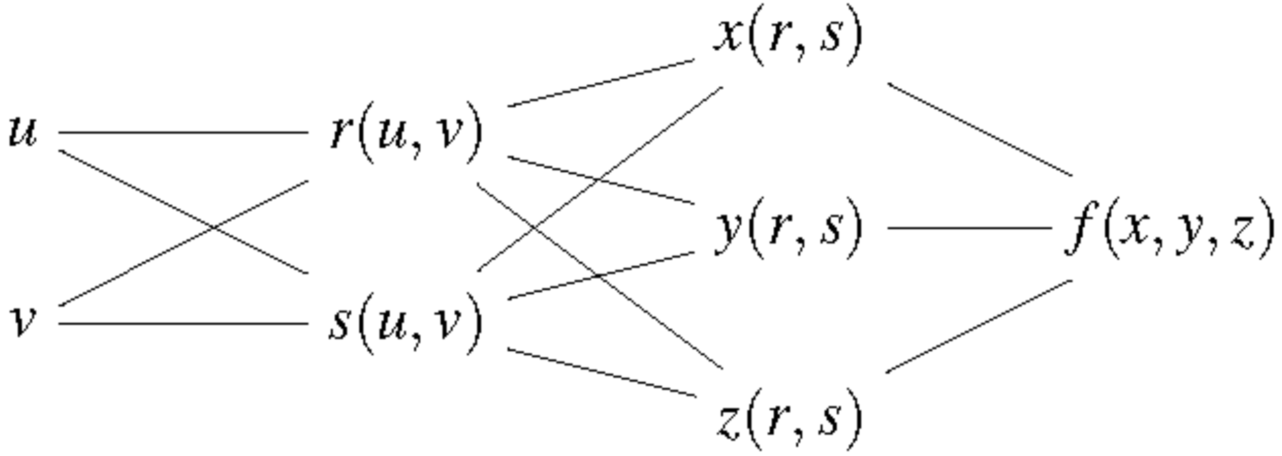
What is the derivative of $f$ with respect to $s$?




**Example**: Let $f = xyz$ , $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = yz \cdot 0 + xz \cdot r + xy \cdot 1$$

What is the derivative of $f$ with respect to $s$?

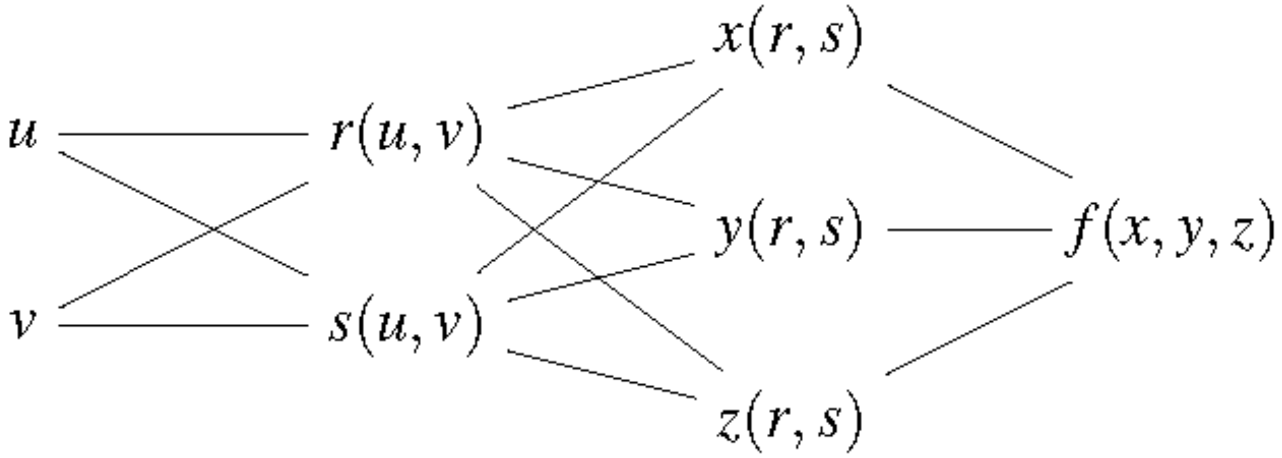



**Example**: Let $f = xyz$ , $x = r$, $y = rs$, and $z = s$. Find $\partial f/\partial s$

$$\frac{\partial f}{\partial s} = yz \cdot 0 + xz \cdot r + xy \cdot 1$$

## The Chain Rule

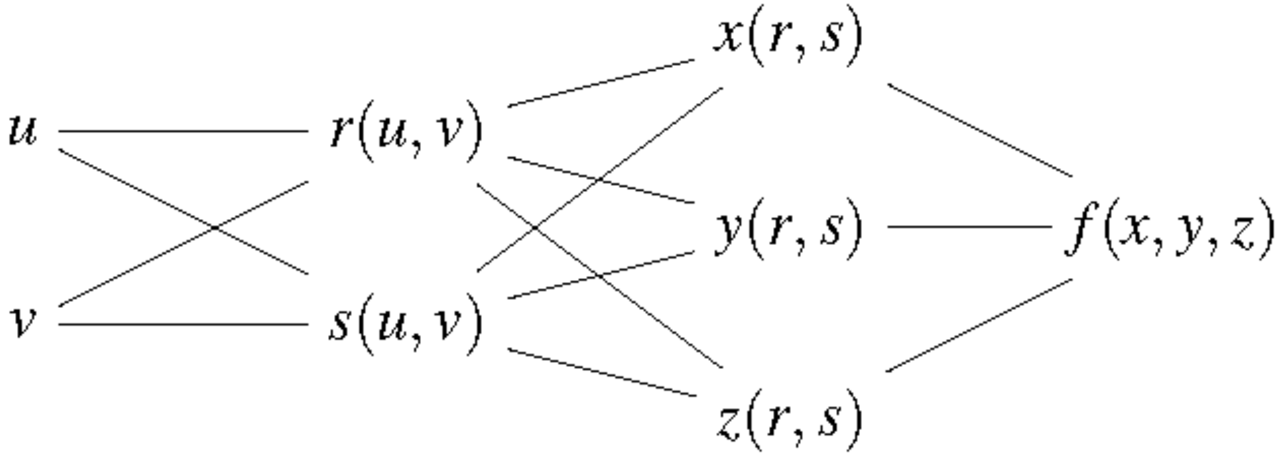
What is the derivative of $f$ with respect to $s$?




**Example**: Let $f = xyz$ , $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = rs^2 \cdot 0 + rs \cdot r + r^2 s \cdot 1$$

What is the derivative of $f$ with respect to $s$?

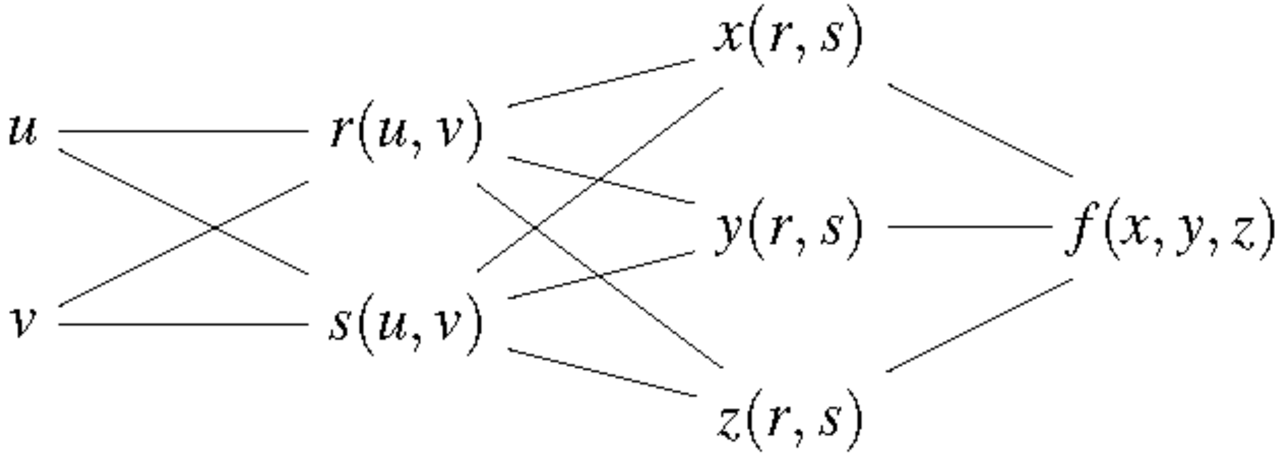



**Example**: Let $f = xyz$ , $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = 2r^2 s$$

## The Chain Rule

What is the derivative of $f$ with respect to $s$?

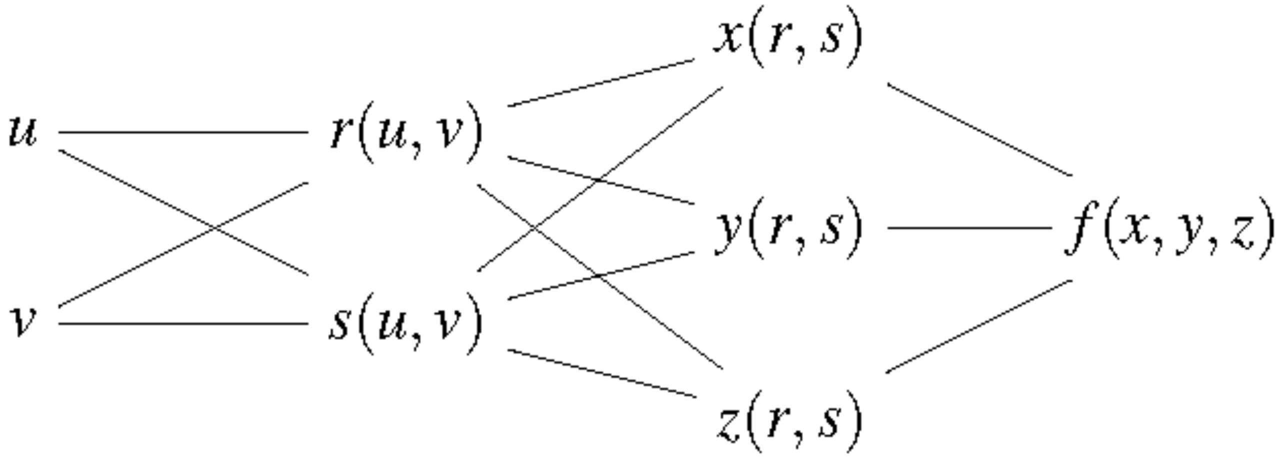



**Example**: Let $f = xyz$ , $x = r$, $y = rs$, and $z = s$. Find $\partial f/\partial s$

$$f(r,s) = r \cdot rs \cdot s = r^2 s^2 \quad \Rightarrow \quad \frac{\partial f}{\partial s} = 2r^2 s \ \checkmark$$

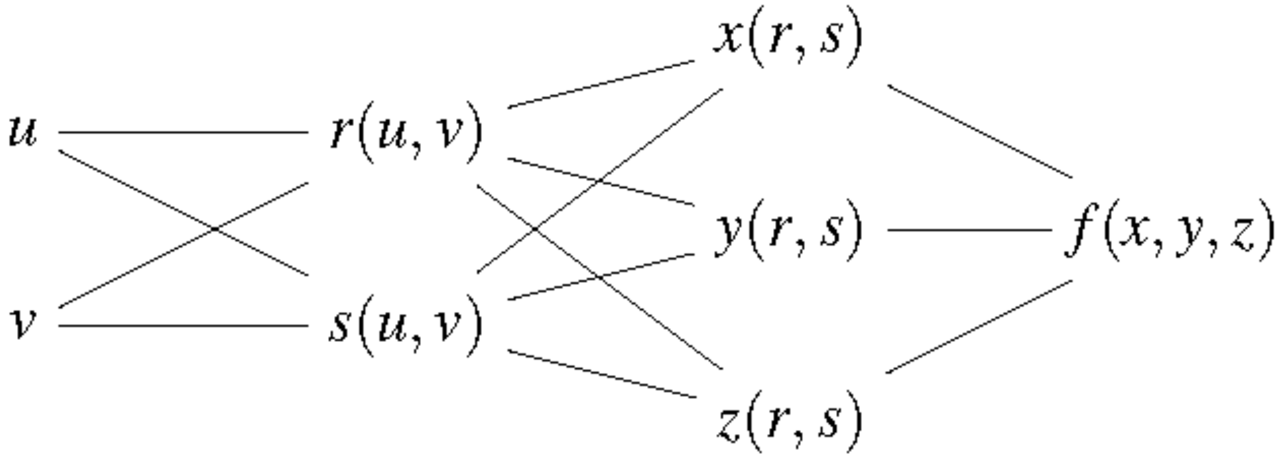
## The Chain Rule

What is the derivative of $f$ with respect to $u$?



## The Chain Rule

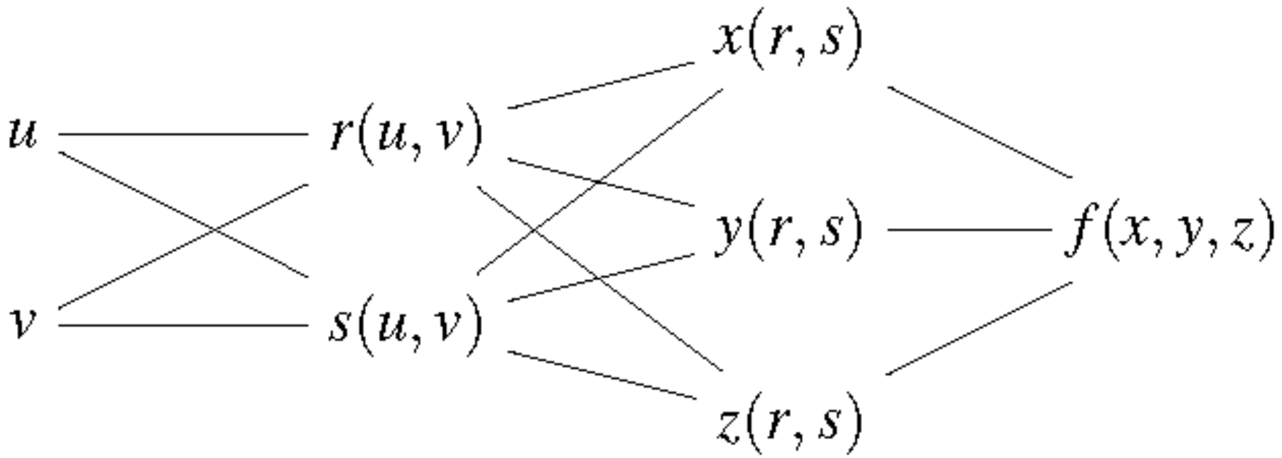
What is the derivative of $f$ with respect to $u$?




$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial r}\frac{\partial r}{\partial u} + \frac{\partial f}{\partial s}\frac{\partial s}{\partial u}$$

## The Chain Rule

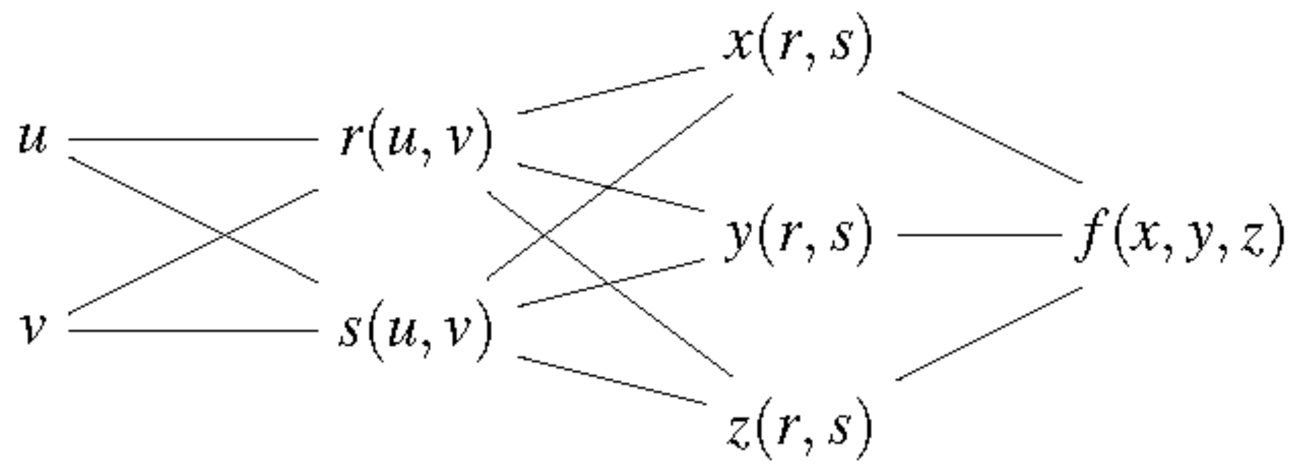
What is the derivative of $f$ with respect to $u$?




**Crux**: If you know the derivative of objective w.r.t. intermediate value in the chain, can eliminate everything in between.

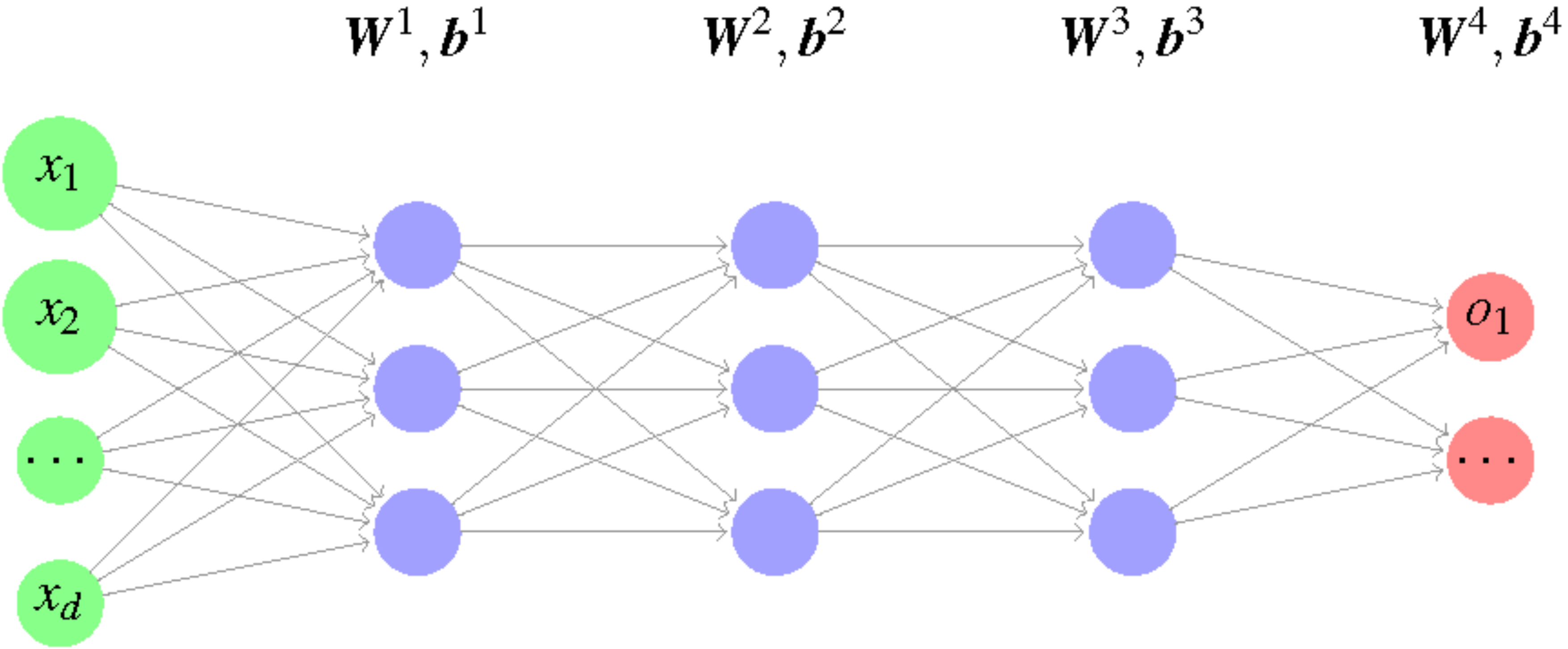
## The Chain Rule

What is the derivative of $f$ with respect to $u$?



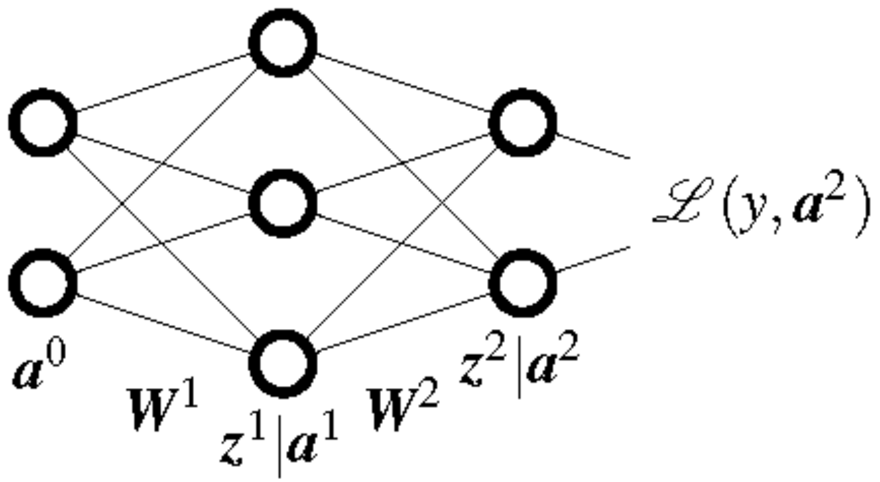

**Crux**: If you know the derivative of objective w.r.t. intermediate value in the chain, can eliminate everything in between.
This is the cornerstone of the *back propagation* algorithm.

## Back Propagation



## Back Propagation

For the derivation, we'll consider a simplified network



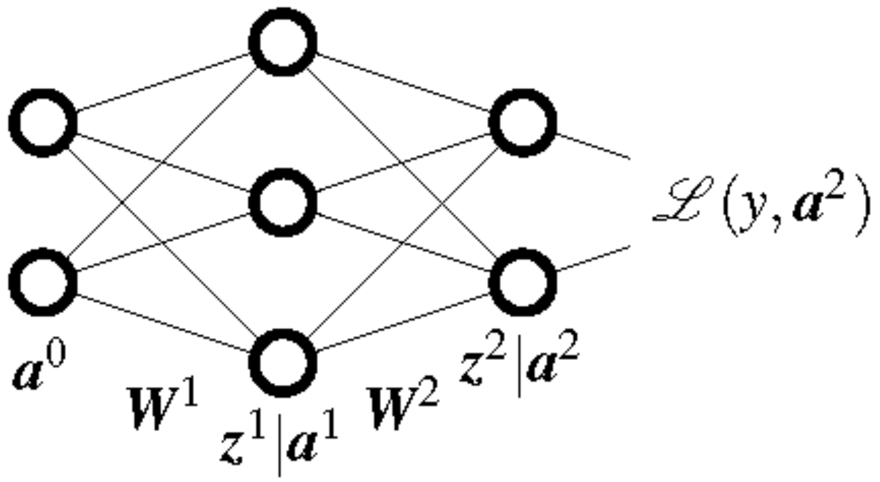

We want to use back propagation to compute partial derivative of $\mathscr{L}$ w.r.t. the weights and biases

$$\frac{\partial \mathscr{L}}{\partial w_{ij}^2}, \text{for } l = 1, 2$$

$w_{ij}^l$ is the weight from node $j$ in layer $l-1$ to node $i$ in layer $l$.

## Back Propagation

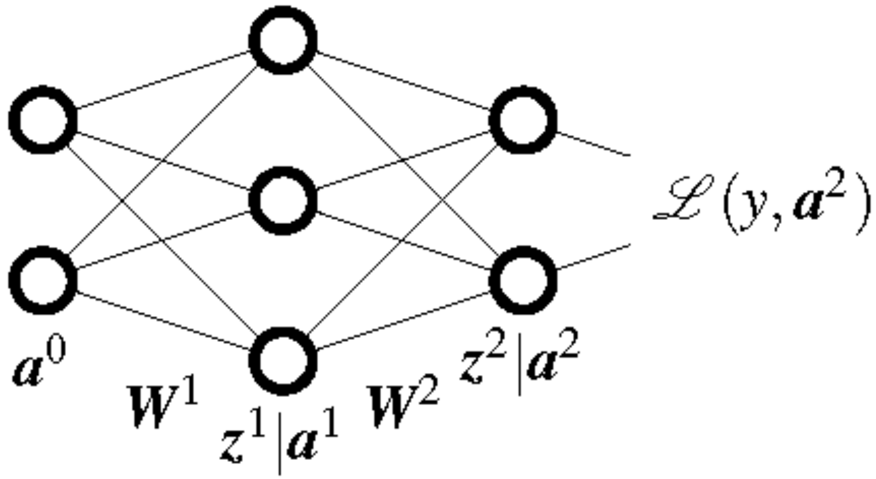
For the derivation, we'll consider a simplified network




We need to choose an intermediate term that lives on the nodes, that we can easily compute derivative with respect to.
Could choose $\boldsymbol{a}$'s, but we'll choose $z$'s because math is easier.

## Back Propagation

For the derivation, we'll consider a simplified network



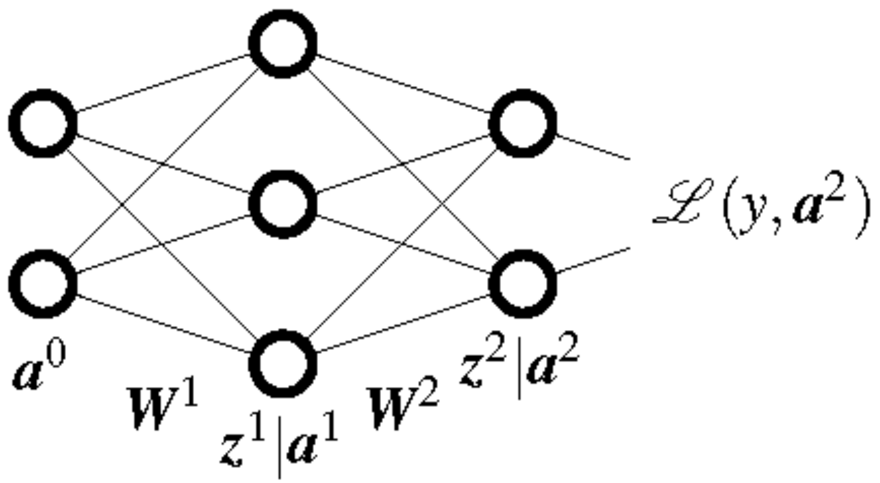

Define the derivative w.r.t. the $z$'s by $\delta$:

$$\delta_j^l = \frac{\partial \mathscr{L}}{\partial z_j^l}$$

Note that $\delta^l$ has the same size as $z^l$ and $\boldsymbol{a}^l$.

## Back Propagation

For the derivation, we'll consider a simplified network




Let's compute $\delta^L$ for output layer $L$:

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial z_j^L} = \frac{\partial \mathscr{L}}{\partial a_j^L} \frac{d a_j^L}{d z_j^L}$$

## Back Propagation

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial z_j^L} = \frac{\partial \mathscr{L}}{\partial a_j^L}\frac{da_j^L}{dz_j^L}$$

We know that $a_j^L = g(z_j^L)$, so $\dfrac{da_j^L}{dz_j^L} = g'(z_j^L)$

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial a_j^L} g'(z_j^L)$$

Note: The first term is $j^{\text{th}}$ entry of gradient of $\mathscr{L}$.

## Back Propagation

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial a_j^L} g'(z_j^L)$$

We can combine all of these into a vector operation

$$\boldsymbol{\delta}^L = \frac{\partial \mathscr{L}}{\partial \boldsymbol{a}^L} \odot g'(\boldsymbol{z}^L)$$

Where $g'(z^L)$ is the activation function applied elementwise to $z^L$.
The symbol $\odot$ indicates element-wise multiplication of vectors.

## Back Propagation

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial a_j^L} g'(z_j^L)$$

We can combine all of these into a vector operation

$$\boldsymbol{\delta}^L = \frac{\partial \mathscr{L}}{\partial \boldsymbol{a}^L} \odot g'(\boldsymbol{z}^L)$$

Where $g'(z^L)$ is the activation function applied elementwise to $z^L$.
The symbol $\odot$ indicates element-wise multiplication of vectors.
Notice that computing $\delta^L$ requires knowing activations.
This means that before we can compute derivatives for SGD through back propagation, we first run forward propagation through the network.

## Back Propagation

Example: Suppose we're in regression setting and choose a sigmoid activation function:

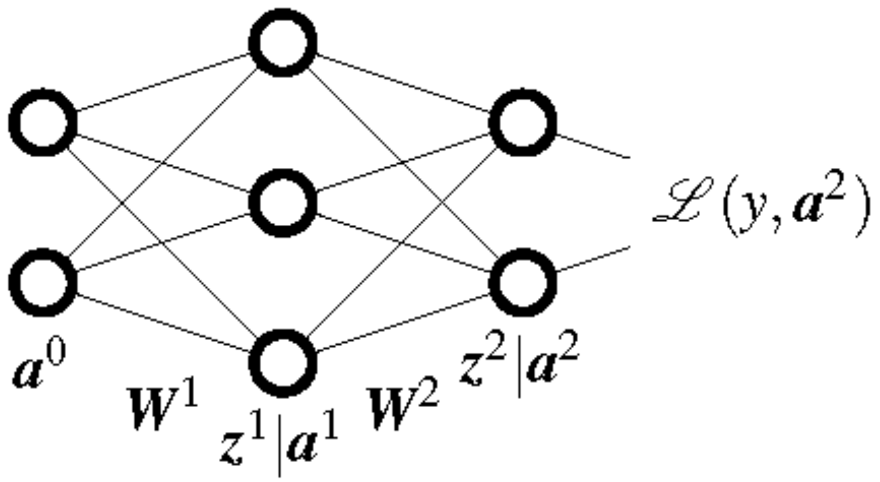
$$\mathscr{L} = \frac{1}{2}\sum_j (y_j - a_j^L)^2 \quad \text{and} \quad a_j^L = \sigma(z)$$

$$\frac{\partial \mathscr{L}}{\partial a_j^L} = (a_j^L - y_j), \quad \frac{da_j^L}{dz_j^L} = \sigma'(z_j^L) = \sigma(z_j^L)(1 - \sigma(z_j^L))$$

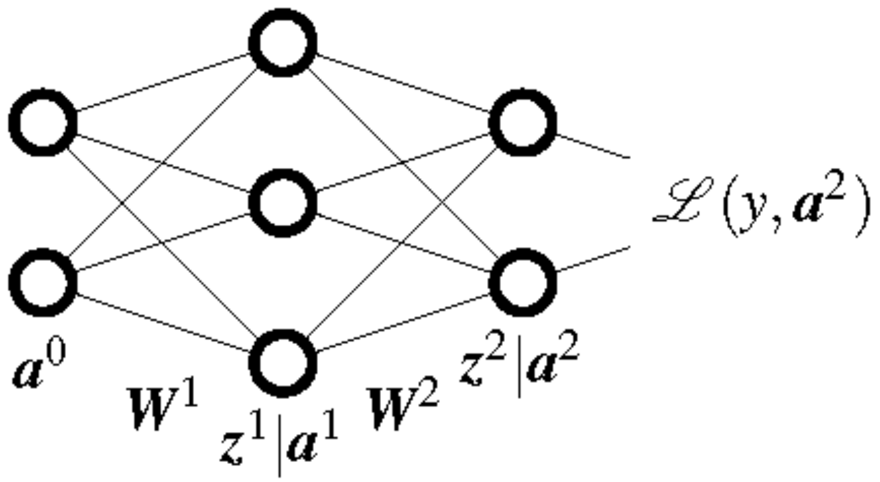
So $\boldsymbol{\delta}^L = (\boldsymbol{a}^L - y) \odot \sigma(\boldsymbol{z}^L) \odot (1 - \sigma(\boldsymbol{z}^L))$

## Back Propagation

OK Great! Now we can easily-ish compute the $\delta$'s for the output layer.
But really we're after partials w.r.t. to weights and biases.



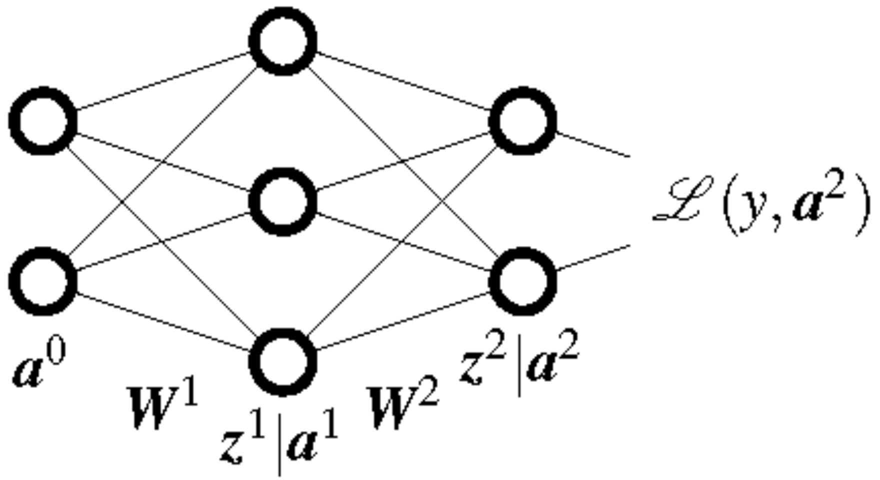
OK Great! Now we can easily-ish compute the $\delta$'s for the output layer.
But really we're after partials w.r.t. to weights and biases.




**Question**: What do you notice?

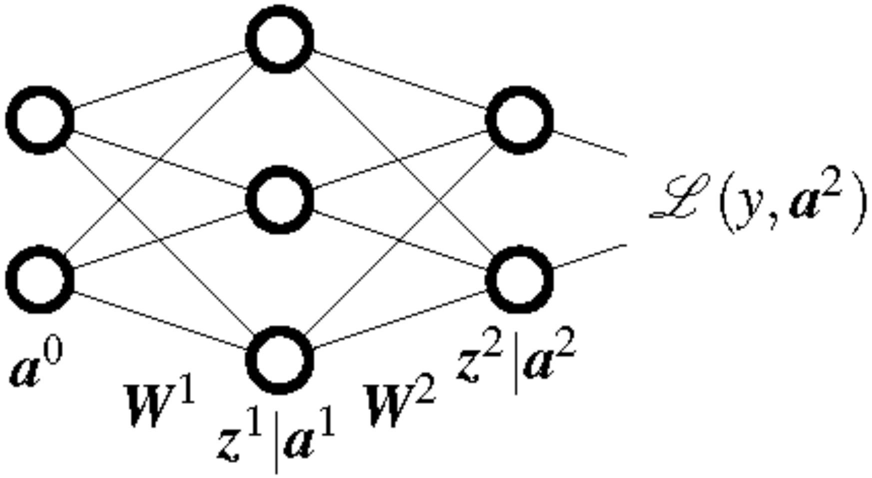
## Back Propagation

We want to find derivative $\mathscr{L}$ w.r.t. to weights and biases




Every weight connected to a node in layer $L$ depends on a single $\delta_j^L$

## Back Propagation

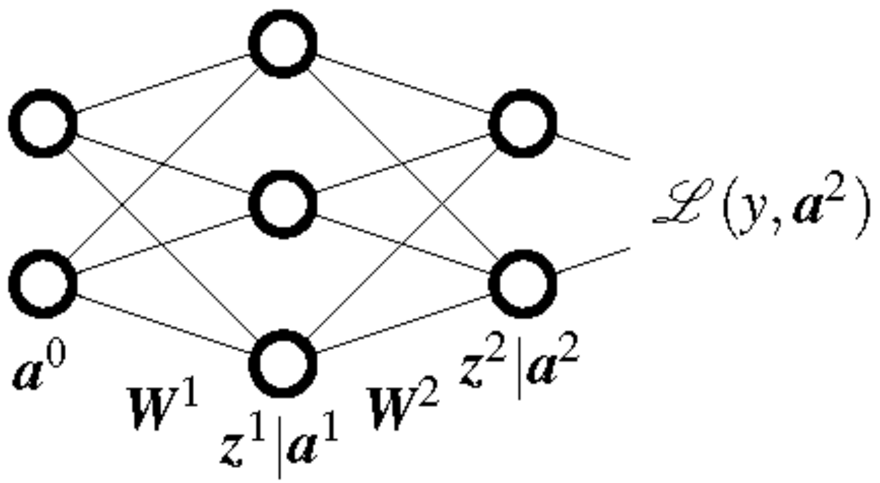



So we have $$\frac{\partial \mathscr{L}}{\partial w_{jk}^L} = \frac{\partial \mathscr{L}}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{jk}^L} = \delta_j^L \frac{\partial z_j^L}{\partial w_{jk}^L}$$

Need to compute $\frac{\partial z_j^L}{\partial w_{jk}^L}$. Recall $\mathbf{z}^L = W^L\mathbf{a}^{L-1} + \mathbf{b}^L$

$$j^{\text{th}} \text{ entry in vector} \Rightarrow \quad z_j^L = \sum_i w_{ji}^L a_i^{L-1} + b_j^L$$

## Back Propagation

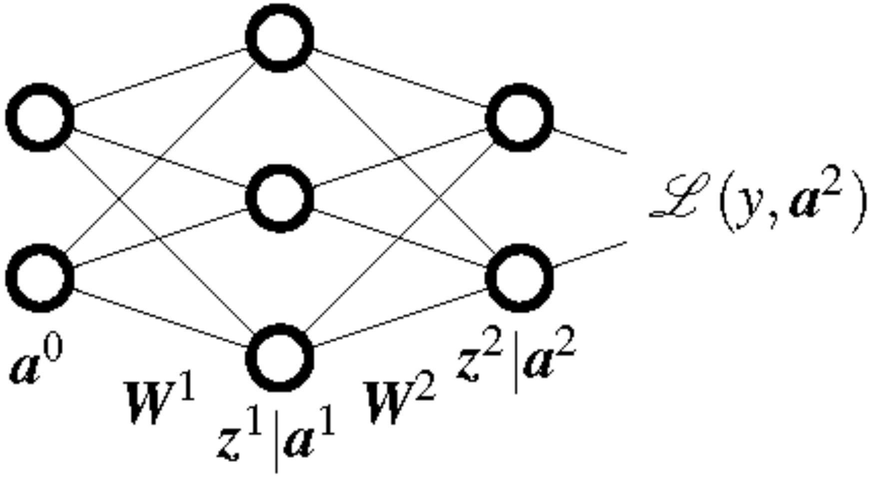



So we have $$\frac{\partial \mathscr{L}}{\partial w_{jk}^L} = \frac{\partial \mathscr{L}}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} = \delta_j^L \frac{\partial z_j^L}{\partial w_{jk}^L}$$

Taking derivative w.r.t. $w_{jk}^L$ gives

$$\Rightarrow \quad \frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1} \quad \Rightarrow \quad \frac{\partial \mathscr{L}}{\partial w_{jk}^L} = a_k^{L-1} \delta_j^L$$

## Back Propagation




So we have $$\frac{\partial \mathscr{L}}{\partial w^L_{jk}} = a^{L-1}_k \delta^L_j$$

Easy expression for derivative w.r.t. every weight leading into layer $L$.

## Back Propagation

Let's make the notation a little more practical.

$$\boldsymbol{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$$

## Back Propagation

Let's make the notation a little more practical.

$$\boldsymbol{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$$

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^2} = \begin{bmatrix} \frac{\partial \mathscr{L}}{\partial w_{11}^2} & \frac{\partial \mathscr{L}}{\partial w_{12}^2} & \frac{\partial \mathscr{L}}{\partial w_{13}^2} \\ \frac{\partial \mathscr{L}}{\partial w_{21}^2} & \frac{\partial \mathscr{L}}{\partial w_{22}^2} & \frac{\partial \mathscr{L}}{\partial w_{23}^2} \end{bmatrix} = \begin{bmatrix} \delta_1^2 a_1^1 & \delta_1^2 a_2^1 & \delta_1^2 a_3^1 \\ \delta_2^2 a_1^1 & \delta_2^2 a_2^1 & \delta_2^2 a_3^1 \end{bmatrix}$$

Now we can write this as an outer-product of $\boldsymbol{\delta}^2$ and $\boldsymbol{a}^1$,

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^2} = \boldsymbol{\delta}^2 (\boldsymbol{a}^1)^T$$

(Exercise for yourself, $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^2}$)

## Intermediate summary

For a giving training example $\boldsymbol{x}$, perform forward propagation to get $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ on each layer.
Then to get the partial derivatives for $\boldsymbol{W}^2$ or $\boldsymbol{W}^L$:

1. Compute $\boldsymbol{\delta}^L = \frac{\partial \mathscr{L}}{\partial a_j^L} \odot g'(\boldsymbol{z}^L)$
2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^L} = \boldsymbol{\delta}^L (\boldsymbol{a}^{L-1})^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L$

OK, that wasn't so bad! We found very simple expressions for the derivatives with respect to the weights in the last hidden layer!

## Intermediate summary

For a giving training example $\boldsymbol{x}$, perform forward propagation to get $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ on each layer.
Then to get the partial derivatives for $\boldsymbol{W}^2$ or $\boldsymbol{W}^L$:

1. Compute $\boldsymbol{\delta}^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \odot g'(\boldsymbol{z}^L)$
2. Compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^L} = \boldsymbol{\delta}^L (\boldsymbol{a}^{L-1})^T$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L$

OK, that wasn't so bad! We found very simple expressions for the derivatives with respect to the weights in the last hidden layer!
**Problem**: How do we do the other layers?

## Intermediate summary

For a giving training example $\boldsymbol{x}$, perform forward propagation to get $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ on each layer.
Then to get the partial derivatives for $\boldsymbol{W}^2$ or $\boldsymbol{W}^L$:

1. Compute $\boldsymbol{\delta}^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \odot g'(\boldsymbol{z}^L)$
2. Compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^L} = \boldsymbol{\delta}^L (\boldsymbol{a}^{L-1})^T$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L$

OK, that wasn't so bad! We found very simple expressions for the derivatives with respect to the weights in the last hidden layer!
**Problem**: How do we do the other layers?
Since the formulas were so nice once we knew the adjacent $\delta^l$, it sure would be nice if we could easily compute the $\delta^l$'s on earlier layers.

## Back Propagation

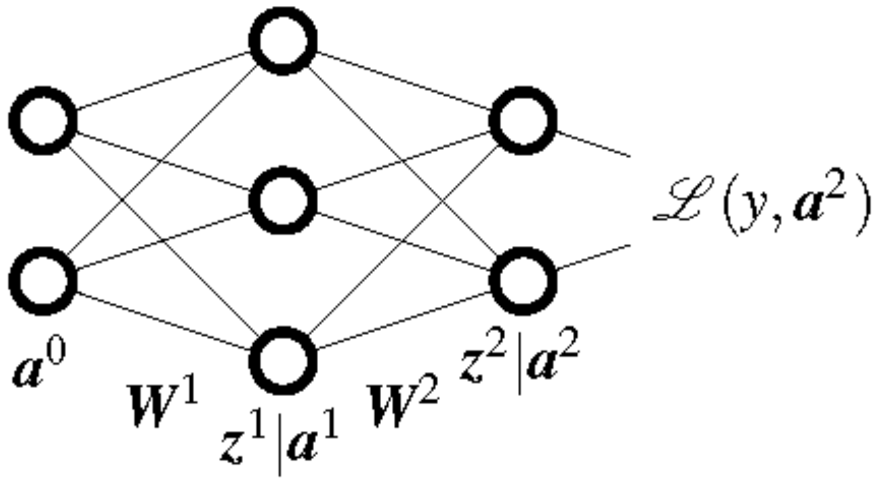
But the relationship between $\mathscr{L}$ and $z^1$ is really complicated because of multiple passes through the activation functions.

## Back Propagation

But the relationship between $\mathscr{L}$ and $z^1$ is really complicated because of multiple passes through the activation functions.
It is OK! Back propagation comes to rescue!

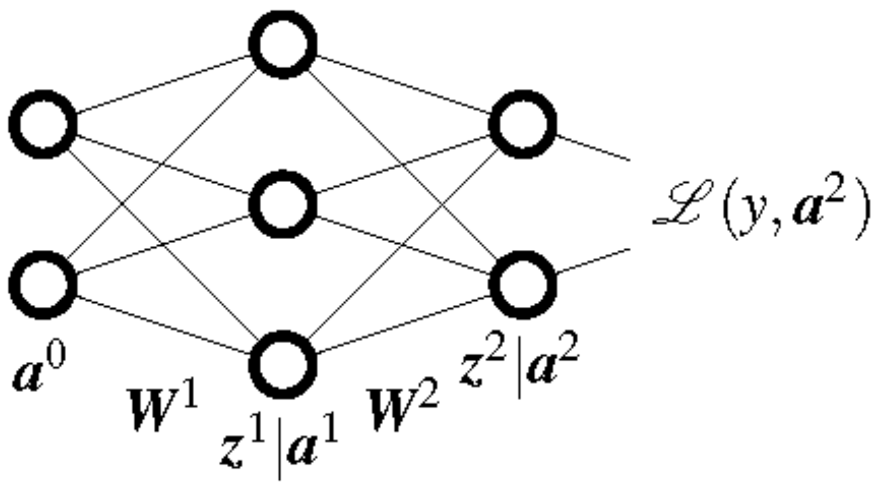
## Back Propagation

But the relationship between $\mathscr{L}$ and $z^1$ is really complicated because of multiple passes through the activation functions.
It is OK! Back propagation comes to rescue!
Notice that $\delta^1$ depends on $\delta^2$.



## Back Propagation

Notice that $\delta^1$ depends on $\delta^2$.

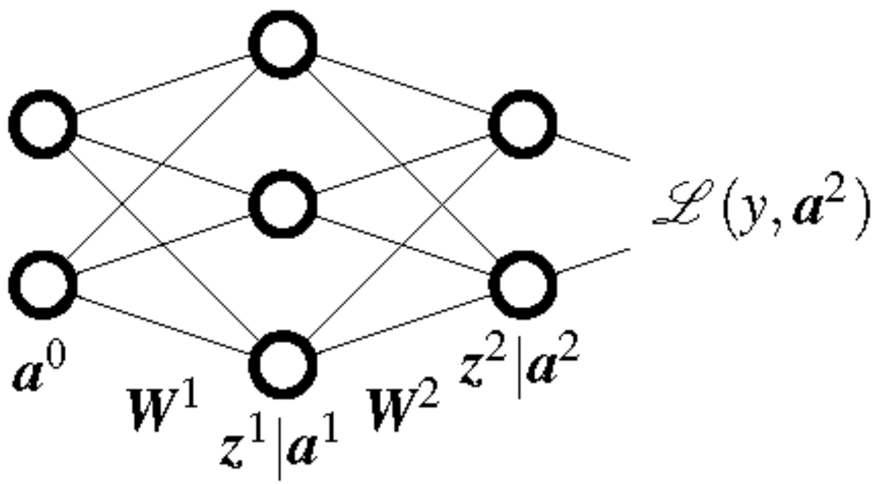



By multivariate chain rule,

$$\frac{\partial \mathscr{L}}{\partial z_k^{l-1}} = \sum_j \frac{\partial \mathscr{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial z_k^{l-1}}$$

## Back Propagation

Notice that $\delta^1$ depends on $\delta^2$.

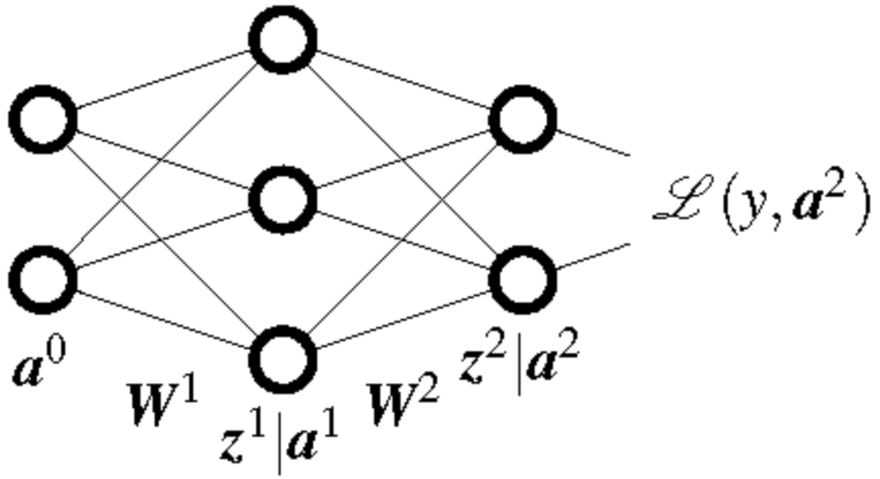



By multivariate chain rule,

$$\delta_k^{l-1} = \frac{\partial \mathscr{L}}{\partial z_k^{l-1}} = \sum_j \frac{\partial \mathscr{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial z_k^{l-1}} = \sum_j \delta_j^l \frac{\partial z_j^l}{\partial z_k^{l-1}}$$

## Back Propagation

Notice that $\delta^1$ depends on $\delta^2$.




By multivariate chain rule,

$$\delta_2^1 = \frac{\partial \mathscr{L}}{\partial z_2^1} = \delta_1^2 \frac{\partial z_1^2}{\partial z_2^1} + \delta_2^2 \frac{\partial z_2^2}{\partial z_2^1}$$

## Back Propagation

$$\delta_2^1 = \frac{\partial \mathscr{L}}{\partial z_2^1} = \delta_1^2 \frac{\partial z_1^2}{\partial z_2^1} + \delta_2^2 \frac{\partial z_2^2}{\partial z_2^1}$$

## Back Propagation

$$\delta_2^1 = \frac{\partial \mathscr{L}}{\partial z_2^1} = \delta_1^2 \frac{\partial z_1^2}{\partial z_2^1} + \delta_2^2 \frac{\partial z_2^2}{\partial z_2^1}$$

Recall that $\boldsymbol{z}^2 = \boldsymbol{W}^2 a^1 + \boldsymbol{b}^2$, it follows that

$$z_i^2 = w_{i1}^2 a_1^1 + w_{i2}^2 a_2^1 + w_{i3}^2 a_3^1 + b_i^2$$

Taking the derivative $\frac{\partial z_i^2}{\partial z_2^1} = w_{i2}^2 g'(z_2^1)$, and plugging in gives

$$\delta_2^1 = \frac{\partial \mathscr{L}}{\partial z_2^1} = \delta_1^2 w_{12}^2 g'(z_2^1) + \delta_2^2 w_{22}^2 g'(z_2^1)$$

If we do this for each of the 3 $\delta_i^1$'s, something nice happens:
(Exercise for yourself: work out $\delta_1^1$ and $\delta_3^1$ for yourself)

$$
\begin{aligned}
\delta_1^1 &= \delta_1^2 w_{11}^2 g'(z_1^1) + \delta_2^2 w_{21}^2 g'(z_1^1) \\
\delta_2^1 &= \delta_1^2 w_{12}^2 g'(z_2^1) + \delta_2^2 w_{22}^2 g'(z_2^1) \\
\delta_3^1 &= \delta_1^2 w_{13}^2 g'(z_3^1) + \delta_2^2 w_{23}^2 g'(z_3^1)
\end{aligned}
$$

If we do this for each of the 3 $\delta_i^1$'s, something nice happens:
(Exercise for yourself: work out $\delta_1^1$ and $\delta_3^1$ for yourself)

$$
\begin{aligned}
\delta_1^1 &= \delta_1^2 w_{11}^2 g'(z_1^1) + \delta_2^2 w_{21}^2 g'(z_1^1) \\
\delta_2^1 &= \delta_1^2 w_{12}^2 g'(z_2^1) + \delta_2^2 w_{22}^2 g'(z_2^1) \\
\delta_3^1 &= \delta_1^2 w_{13}^2 g'(z_3^1) + \delta_2^2 w_{23}^2 g'(z_3^1)
\end{aligned}
$$

Notice that each row of the system gets multiplied by $g'(z_i^1)$, so let's factor those out.

## Back Propagation

If we do this for each of the 3 $\delta_i^2$'s, something nice happens:

$$
\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^1 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^1 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}
$$

## Back Propagation

If we do this for each of the 3 $\delta_i^2$'s, something nice happens:

$$
\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^1 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^1 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}
$$

Remember $\boldsymbol{\delta}^2 = \begin{bmatrix} \delta_1^2 \\ \delta_2^2 \end{bmatrix}$, $\boldsymbol{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$

Do you see $\boldsymbol{\delta}^2$ and $\boldsymbol{W}^2$ lurking anywhere in the above system?

## Back Propagation

If we do this for each of the 3 $\delta_i^2$'s, something nice happens:

$$
\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^2 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^2 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}
$$

Does this help?

$(\boldsymbol{W}^2)^T = \begin{bmatrix} w_{11}^2 & w_{21}^2 \\ w_{12}^2 & w_{22}^2 \\ w_{13}^2 & w_{23}^2 \end{bmatrix}$, $\boldsymbol{\delta}^2 = \begin{bmatrix} \delta_1^2 \\ \delta_2^2 \end{bmatrix}$.

## Back Propagation

If we do this for each of the 3 $\delta_i^2$'s, something nice happens:

$$
\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^1 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^1 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}
$$

$$\boldsymbol{\delta}^1 = (\boldsymbol{W}^2)^T \boldsymbol{\delta}^2 \odot g'(\boldsymbol{z}^1)$$

## Back Propagation

OK Great!
We can easily compute $\boldsymbol{\delta}^1$ from $\boldsymbol{\delta}^2$
Then we can compute derivatives of $\mathscr{L}$ w.r.t. weights $\boldsymbol{W}^1$ and biases $\boldsymbol{b}^1$ exactly the way we did for $\boldsymbol{W}^2$ and biases $\boldsymbol{b}^2$

1. Compute $\boldsymbol{\delta}^1 = (\boldsymbol{W}^2)^T \boldsymbol{\delta}^2 \odot g'(\boldsymbol{z}^1)$
2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^1} = \boldsymbol{\delta}^1 (\boldsymbol{a}^0)^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^1} = \boldsymbol{\delta}^1$

OK Great!
We can easily compute $\boldsymbol{\delta}^1$ from $\boldsymbol{\delta}^2$
Then we can compute derivatives of $\mathscr{L}$ w.r.t. weights $\boldsymbol{W}^1$ and biases $\boldsymbol{b}^1$ exactly the way we did for $\boldsymbol{W}^2$ and biases $\boldsymbol{b}^2$

1. Compute $\boldsymbol{\delta}^1 = (\boldsymbol{W}^2)^T \boldsymbol{\delta}^2 \odot g'(\boldsymbol{z}^1)$
2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^1} = \boldsymbol{\delta}^1 (\boldsymbol{a}^0)^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^1} = \boldsymbol{\delta}^1$

We've worked this out for a simple network with one hidden layer.
Nothing we've done assumed anything about the number of layers, so we can apply the same procedure recursively with any number of layers.

## Back Propagation

$\delta^L = \frac{\partial \mathscr{L}}{\partial a_j^L} \odot g'(\mathbf{z}^L)$ # Compute $\delta$'s on output layer

For $\ell = L, \ldots, 1$

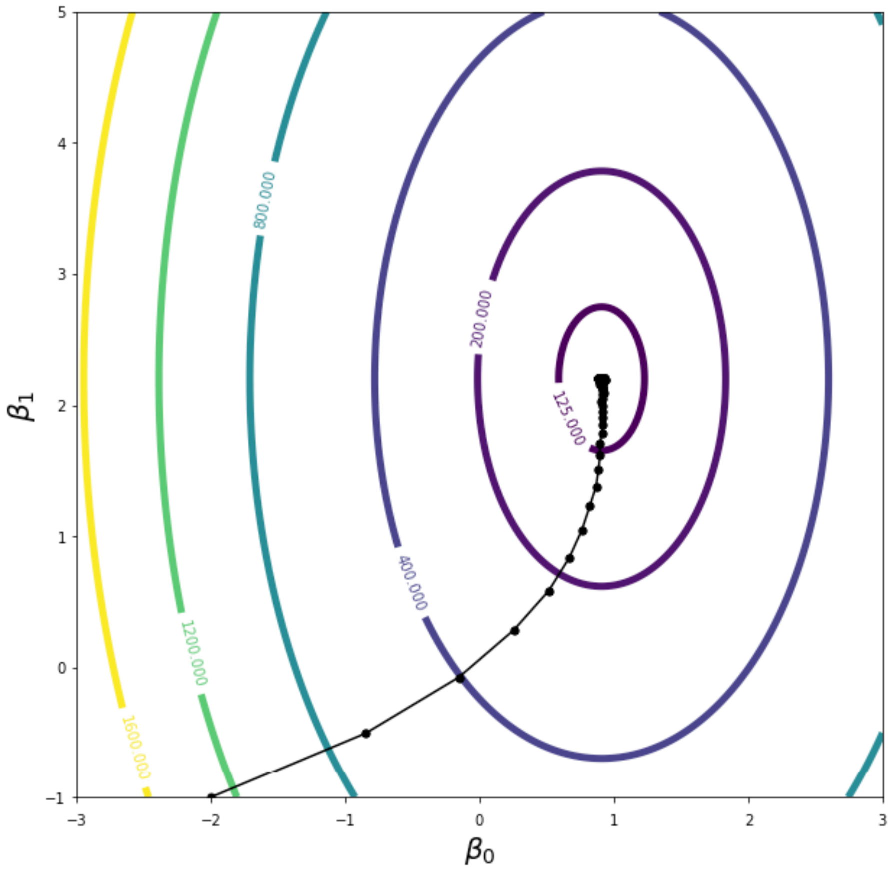
$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^\ell} = \boldsymbol{\delta}^\ell (\boldsymbol{a}^{l-1})^T$ # Compute weight derivatives

$\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^\ell} = \boldsymbol{\delta}^\ell$ # Compute bias derivatives

$\boldsymbol{\delta}^{\ell-1} = \left(W^\ell\right)^T \boldsymbol{\delta}^\ell \odot g'(\mathbf{z}^{\ell-1})$ # Back prop $\delta$'s to previous layer

(After this, ready to do a SGD update on weights/biases)

# Reminder of gradient descent



## Training a Feed-Forward Neural Network

Given initial guess for weights and biases.
Loop over each training example in random order:

1. Forward propagate to get activations on each layer
2. Back propagate to get derivatives
3. Update weights and biases via stochastic gradient descent
4. Repeat

# Outline

- Forward propagation recap
- Back propagation
- Practical issues of back propagation

In practice, many remaining questions may arise.

$\delta^L = \frac{\partial \mathscr{L}}{\partial a_j^L} \odot g'(\mathbf{z}^L)$ # Compute $\delta$'s on output layer

For $\ell = L, \ldots, 1$

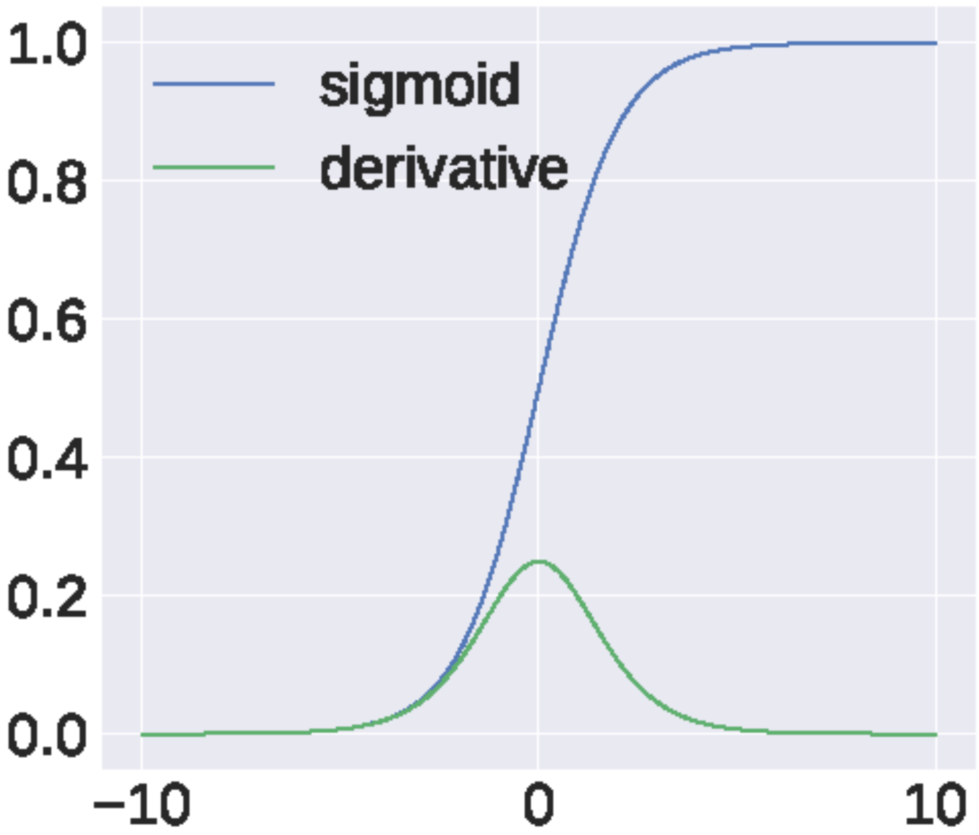
$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^\ell} = \boldsymbol{\delta}^\ell (\boldsymbol{a}^{l-1})^T$ # Compute weight derivatives

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^\ell} = \boldsymbol{\delta}^\ell$$ # Compute bias derivatives

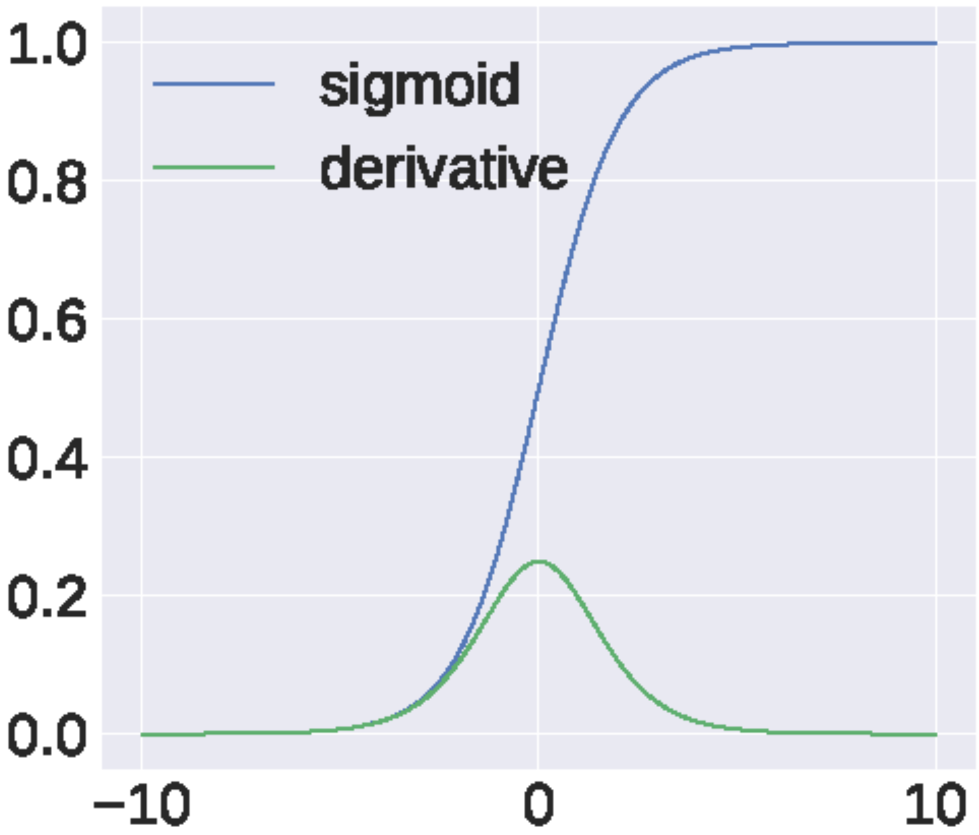
$\boldsymbol{\delta}^{\ell-1} = \left(W^\ell\right)^T \boldsymbol{\delta}^\ell \odot g'(\mathbf{z}^{\ell-1})$ # Back prop $\delta$'s to previous layer

## Unstable gradients

## Unstable gradients



## Unstable gradients



## Vanishing gradients

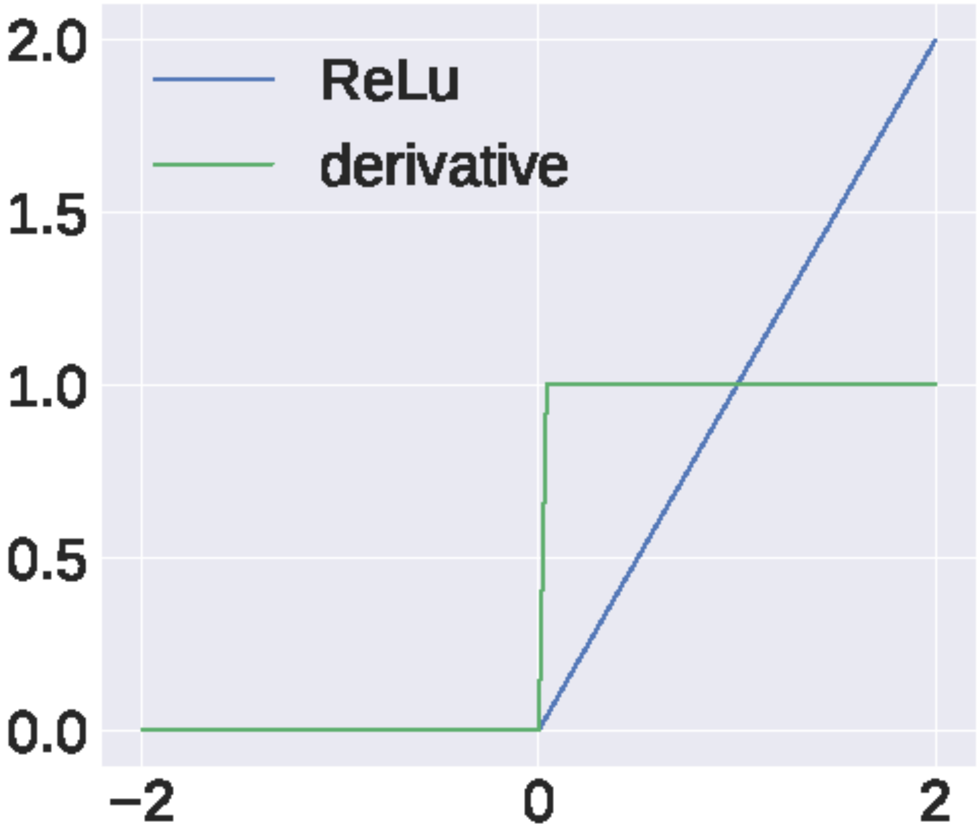
If we use Gaussian initialization for weights, $w^j \sim \mathcal{N}(0,1)$,

$$|w^j| < 1$$

$$|w^j \sigma'(z_j)| < \frac{1}{4}$$

$$\frac{\partial \mathscr{L}}{\partial b^1} \text{ decay to zero exponentially}$$

ReLu



## Exploding gradients

If $w^j = 100$,

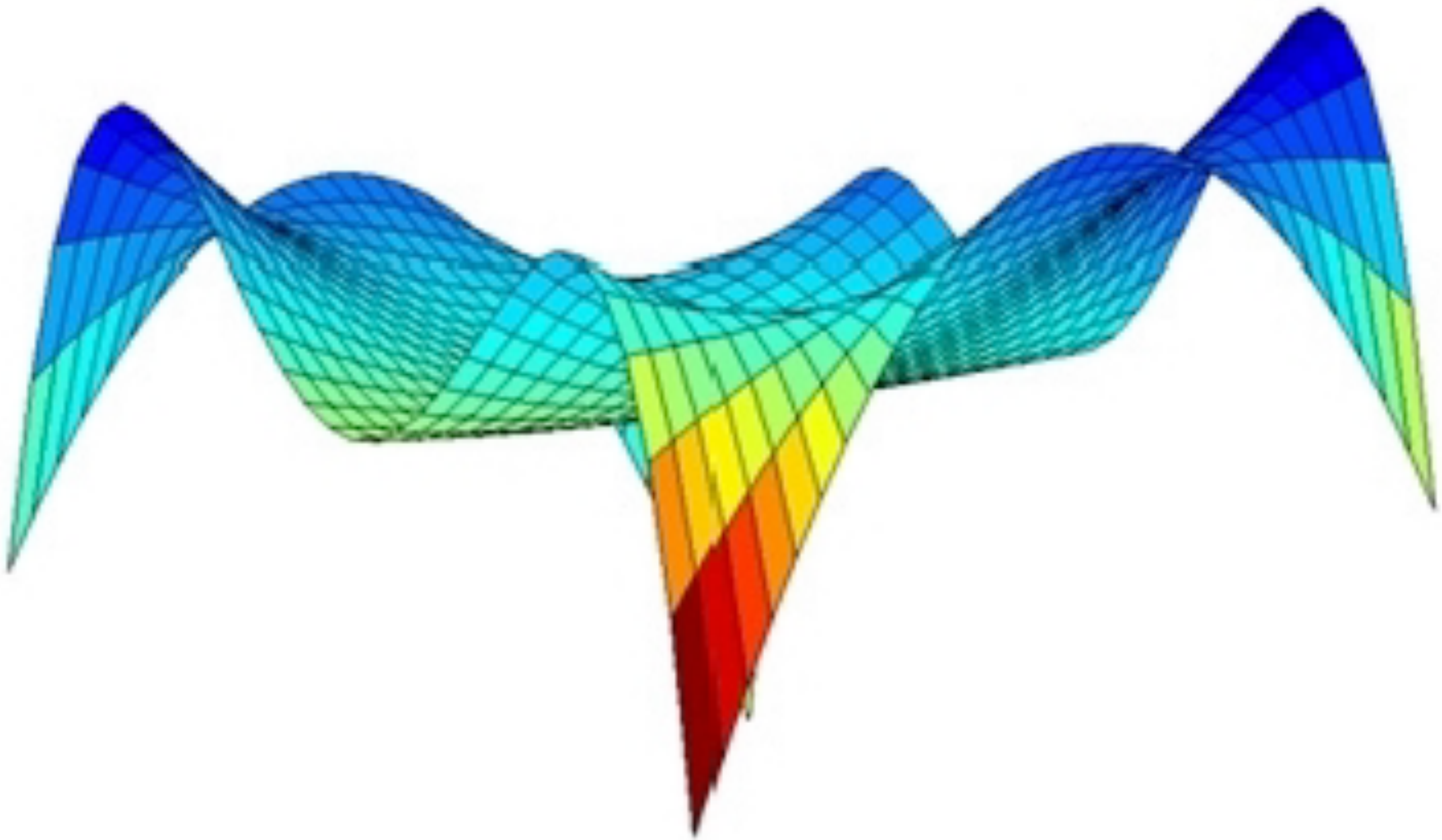
$$|w^j \sigma'(z_j)| \approx k > 1$$

## Training a Feed-Forward Neural Network

In practice, many remaining questions may arise, more examples:

1. How do we initialize weights and biases?
2. How do we regularize?
3. Can I batch this?

## Non-convexity



## Weight initialization

Old idea: $W = 0$, what happens?

## Weight initialization

Old idea: $W = 0$, what happens?
There is no source of asymmetry. (Every neuron looks the same and leads to a slow start.)

## Weight initialization

Old idea: $W = 0$, what happens?

There is no source of asymmetry. (Every neuron looks the same and leads to a slow start.)

$\delta^L = \nabla_{\boldsymbol{a}^L} \mathscr{L} \odot g'(\mathbf{z}^L)$ # Compute $\delta$'s on output layer

For $\ell = L, \ldots, 1$

$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^\ell} = \boldsymbol{\delta}^\ell (\boldsymbol{a}^{l-1})^T$ # Compute weight derivatives

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^\ell} = \boldsymbol{\delta}^\ell$$ # Compute bias derivatives

$\boldsymbol{\delta}^{\ell-1} = \left(W^\ell\right)^T \boldsymbol{\delta}^\ell \odot g'(\mathbf{z}^{\ell-1})$ # Back prop $\delta$'s to previous layer

## Weight initialization

First idea: small random numbers, $W \sim \mathcal{N}(0, 0.01)$

## Weight initialization

$$
\begin{aligned}
Var(z) &= Var(\sum_i w_i x_i) \\
&= nVar(w_i)Var(x_i)
\end{aligned}
$$

## Weight initialization

Xavier initialization [Glorot and Bengio, 2010]

$$W \sim \mathcal{N}(0, \frac{2}{n_{in} + n_{out}})$$

# Weight initialization

Xavier initialization [Glorot and Bengio, 2010]

$$W \sim \mathcal{N}(0, \frac{2}{n_{in} + n_{out}})$$

He initialization [He et al., 2015]

$$W \sim \mathcal{N}(0, \frac{2}{n_{in}})$$

## Weight initialization

Xavier initialization [Glorot and Bengio, 2010]

$$W \sim \mathcal{N}(0, \frac{2}{n_{in} + n_{out}})$$

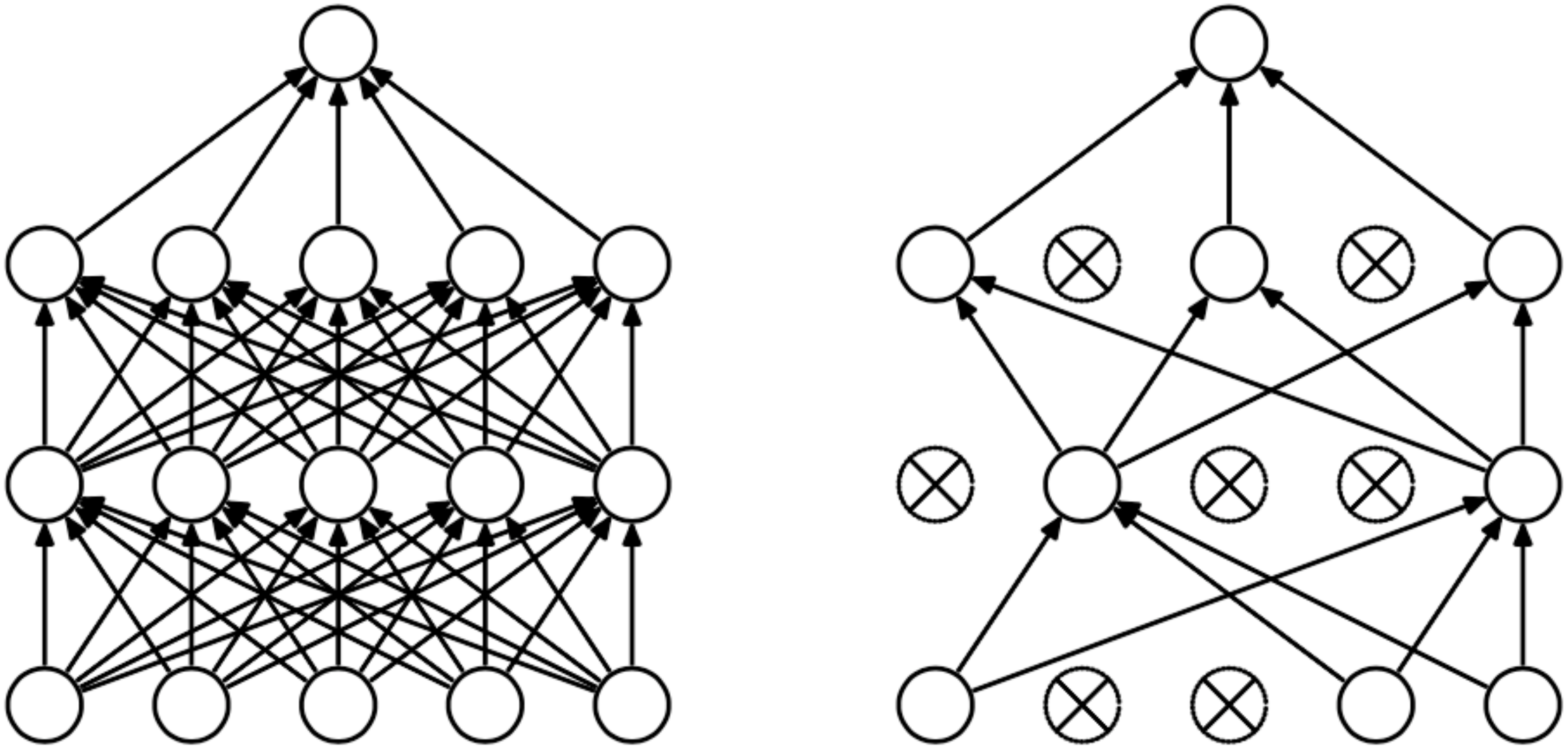
He initialization [He et al., 2015]

$$W \sim \mathcal{N}(0, \frac{2}{n_{in}})$$

This is an actively research area and next great idea may come from you!

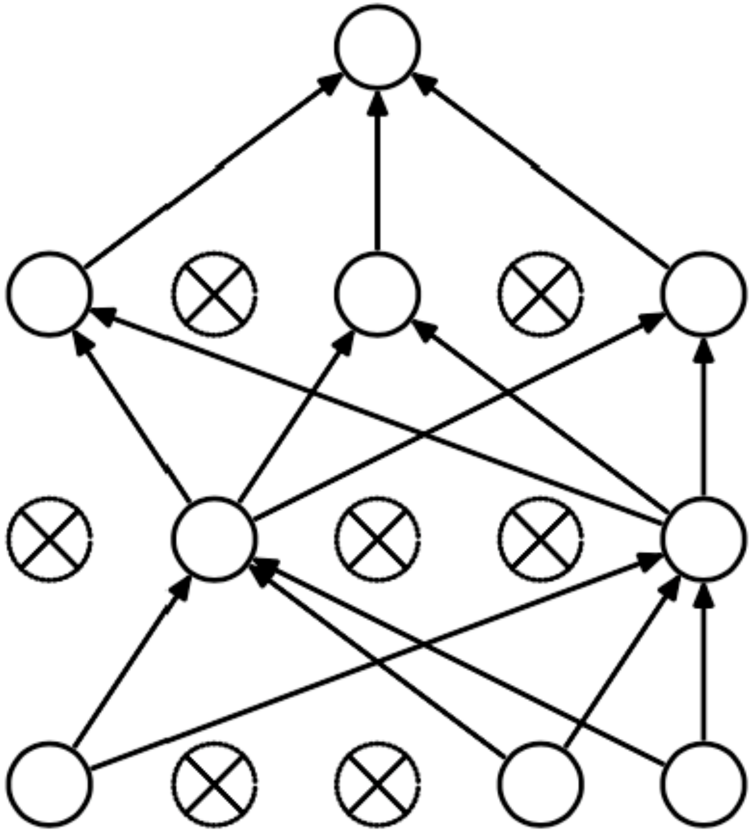
## Dropout layer

"randomly set some neurons to zero in the forward pass" [Srivastava et al., 2014]



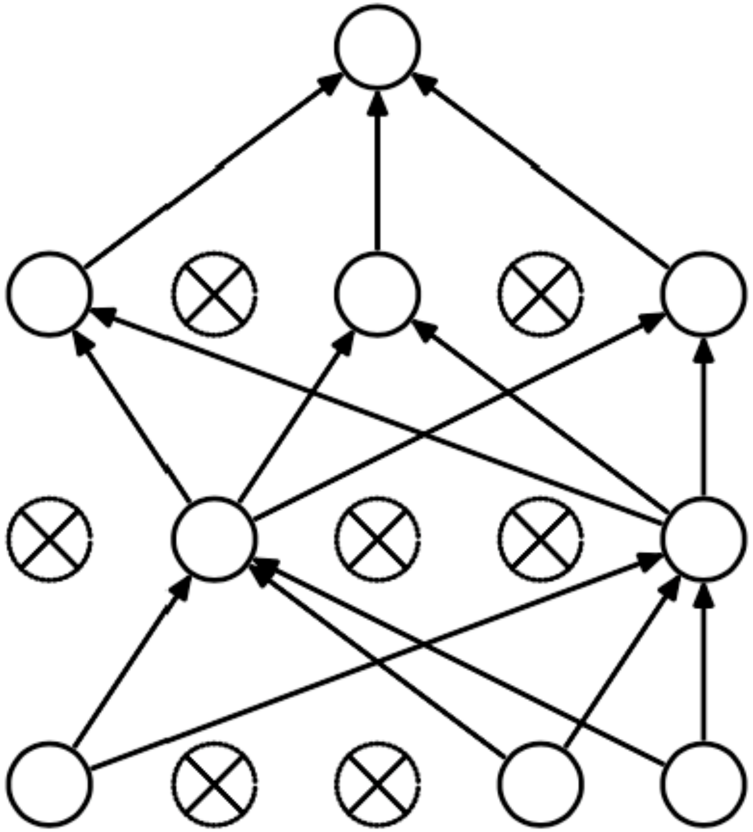
(a) Standard Neural Net    (b) After applying dropout.

## Dropout layer



Forces the network to have a redundant representation.

## Dropout layer



Another interpretation: Dropout is training a large ensemble of models.

## Batch size

We have so far learned gradient descent which uses all training data to compute gradients.

## Batch size

We have so far learned gradient descent which uses all training data to compute gradients.

Alternatively, we use a single instance to compute gradients in stochastic gradient descent.

## Batch size

We have so far learned gradient descent which uses all training data to compute gradients.

Alternatively, we use a single instance to compute gradients in stochastic gradient descent.

In general, we can use a parameter batch size to compute the gradients from a few instances.

- $N$ (the entire training data)
- 1 (a single instance)
- More common values: 16, 32, 64, 128

## Wrap up

Back propagation allows for computing the gradients of the parameters and watch out for unstable gradients!

$\delta^L = \frac{\partial \mathscr{L}}{\partial a_j^L} \odot g'(\mathbf{z}^L)$ # Compute $\delta$'s on output layer

For $\ell = L, \dots, 1$

$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^\ell} = \boldsymbol{\delta}^\ell (\boldsymbol{a}^{l-1})^T$ # Compute weight derivatives

$\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^\ell} = \boldsymbol{\delta}^\ell$ # Compute bias derivatives

$\boldsymbol{\delta}^{\ell-1} = \left(W^\ell\right)^T \boldsymbol{\delta}^\ell \odot g'(\mathbf{z}^{\ell-1})$ # Back prop $\delta$'s to previous layer