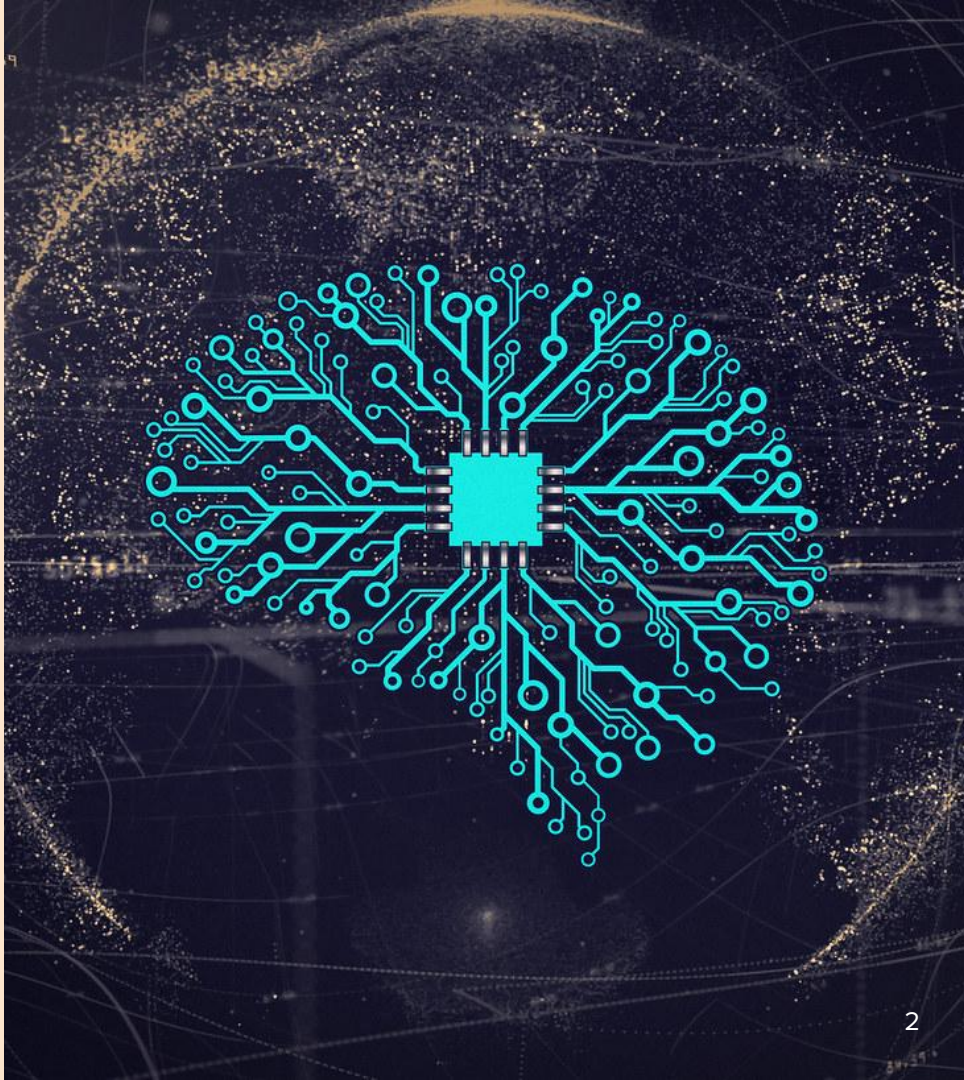
# How do Neural Networks Learn?

Dr. Paisit Khanarsa

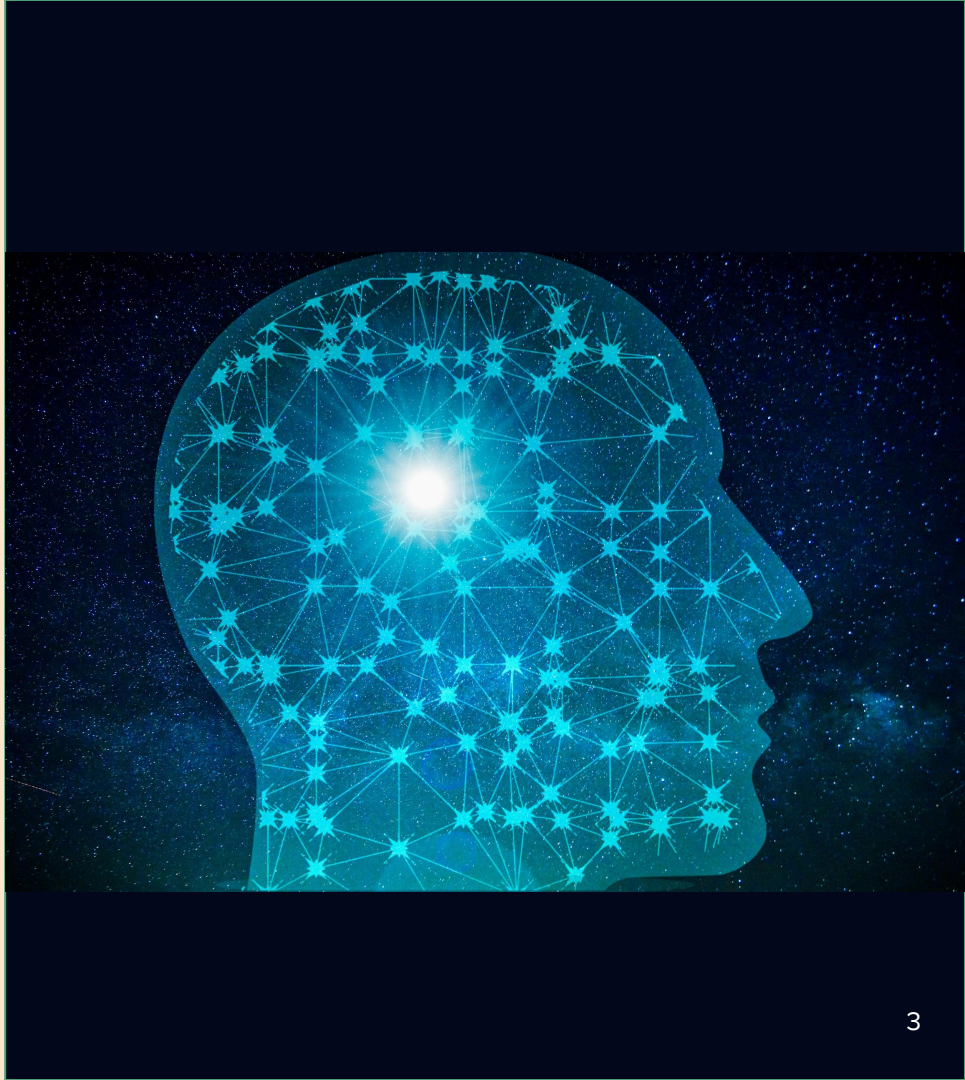
Fibo, Kmutt

# Outline

- Training neural network
- Gradient descent
- Details of neural network training
- Backpropagation algorithm



# Training Neural Network

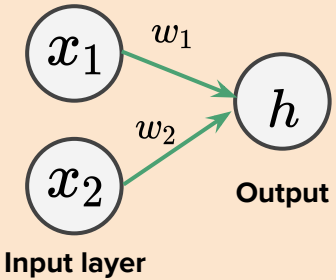


# Training neural network

1. Initialize neural network with number of layers and units we desire. ➔ determine # of w parameters.
2. Initialize the network weights to be random numbers. (Normal Distribution)
3. Measure the goodness of our neural network with a cost function (at first cost function should be high).
4. Adjust the weights with a learning algorithm to minimize cost function.

# Training neural network example

We will start simple by training our neural network to perform regression task with 2D input.




**Neural network**

$$h = f(\sum_j w_j x_j) = f(x_1 w_1 + x_2 w_2)$$

$$Linear\ activation\ function$$

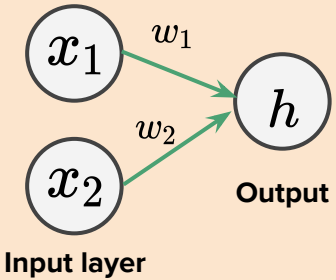
$$f(z) = z$$

$$\vec{w} = [w_1, w_2]$$

- We will pick random numbers for the weights.
- Note that the weight cannot start at zeros.
- Often times, these random initial conditions are constrained to be small.

# Training neural network example

The purpose of regression is to adjust W to minimize regression cost function (sum of square error)




**Neural network**

$$h = f(\sum_j w_j x_j) = f(x_1 w_1 + x_2 w_2)$$

$$\textit{Cost function}$$

$$E(W) = \sum_i^n E^i$$

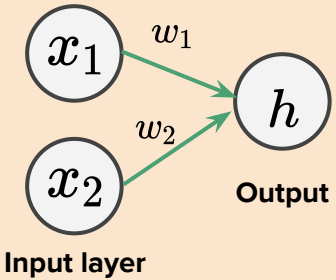
$$= \sum_i^n (h_i - y_i)^2$$

$$= \sum_i^n ((w_1 x_1^i + w_2 x_2^i) - y_i)^2$$

$n$ : Number of samples

# Weight adjustment algorithm

In this example, we will use **gradient descent** algorithm to adjust weights.




**Neural network**

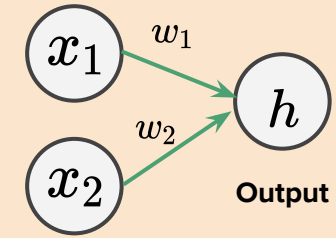
$$w_i := w_i - \alpha \times \frac{\partial Cost}{\partial w_i}$$

$$:= w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$

**New weight** | **Old weight** | **Learning rate** | **Gradient of cost function**

# Weight adjustment algorithm

Adjust weight in vector format.

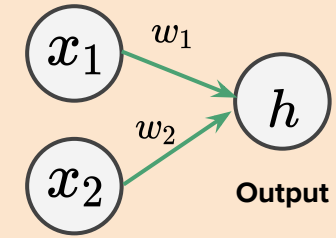



**Neural network**

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longleftarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \bigtriangledown E(w_1, w_2)$$

# Weight adjustment algorithm

Adjust weight in vector format.




**Neural network**

**Weight vector from the last epoch**

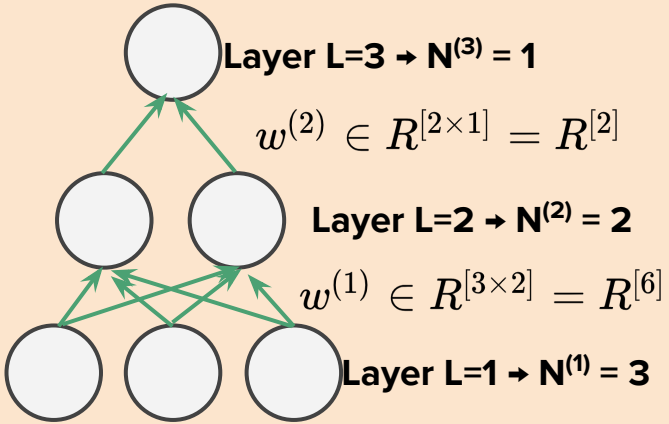
**Gradient vector**

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longleftarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \bigtriangledown E(w_1, w_2)$$

$$\begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$

# Weight adjustment algorithm

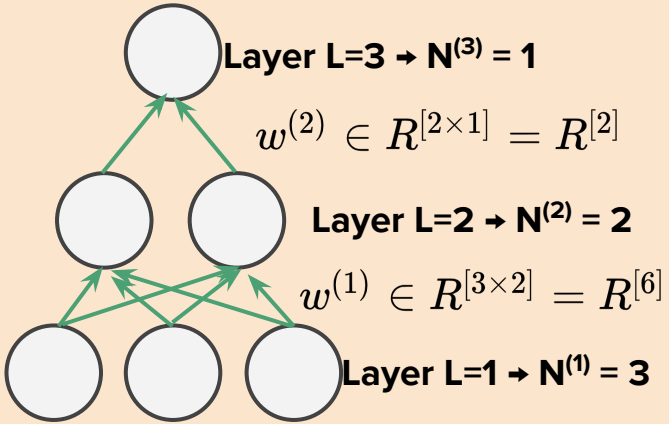
Adjust weight in vector format.




**Neural network**

# Weight adjustment algorithm

Adjust weight in vector format.

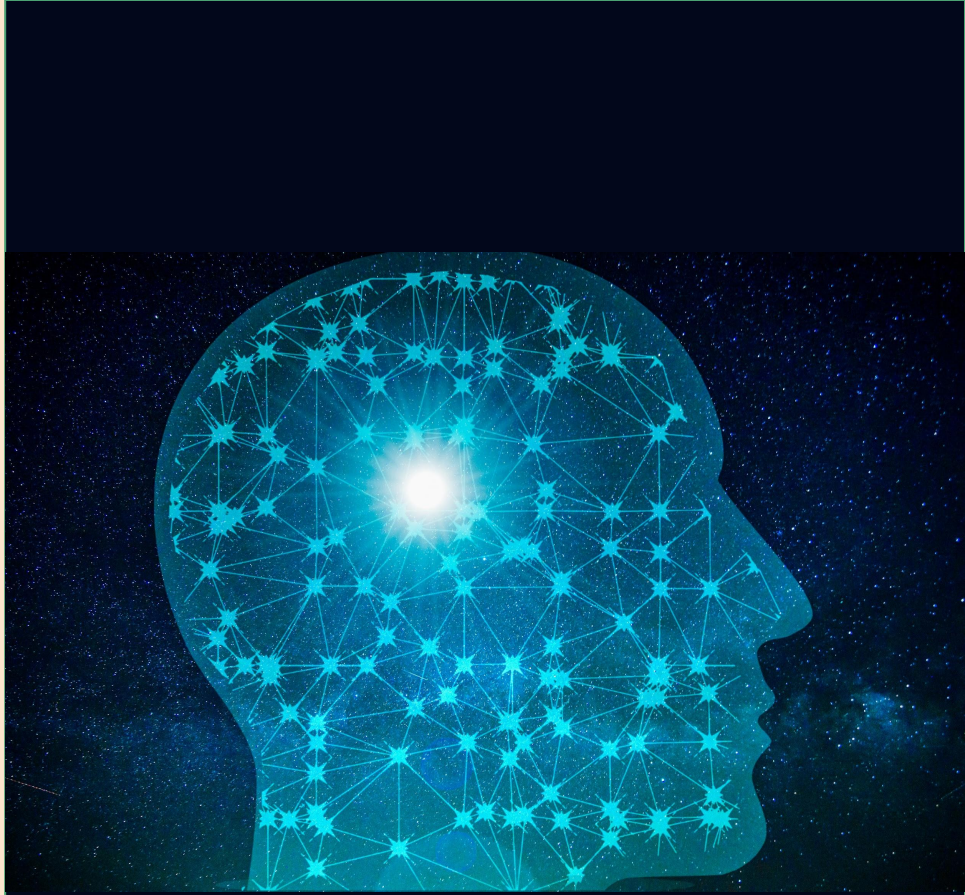



**Neural network**

$$\begin{bmatrix} w_{11}^{(1)} \\ w_{12}^{(1)} \\ w_{21}^{(1)} \\ w_{22}^{(1)} \\ w_{31}^{(1)} \\ w_{32}^{(1)} \\ w_{11}^{(2)} \\ w_{21}^{(2)} \end{bmatrix} \Longrightarrow \begin{bmatrix} w_{11}^{(1)} \\ w_{12}^{(1)} \\ w_{21}^{(1)} \\ w_{22}^{(1)} \\ w_{31}^{(1)} \\ w_{32}^{(1)} \\ w_{11}^{(2)} \\ w_{21}^{(2)} \end{bmatrix} - \alpha \bigtriangledown E(W)$$

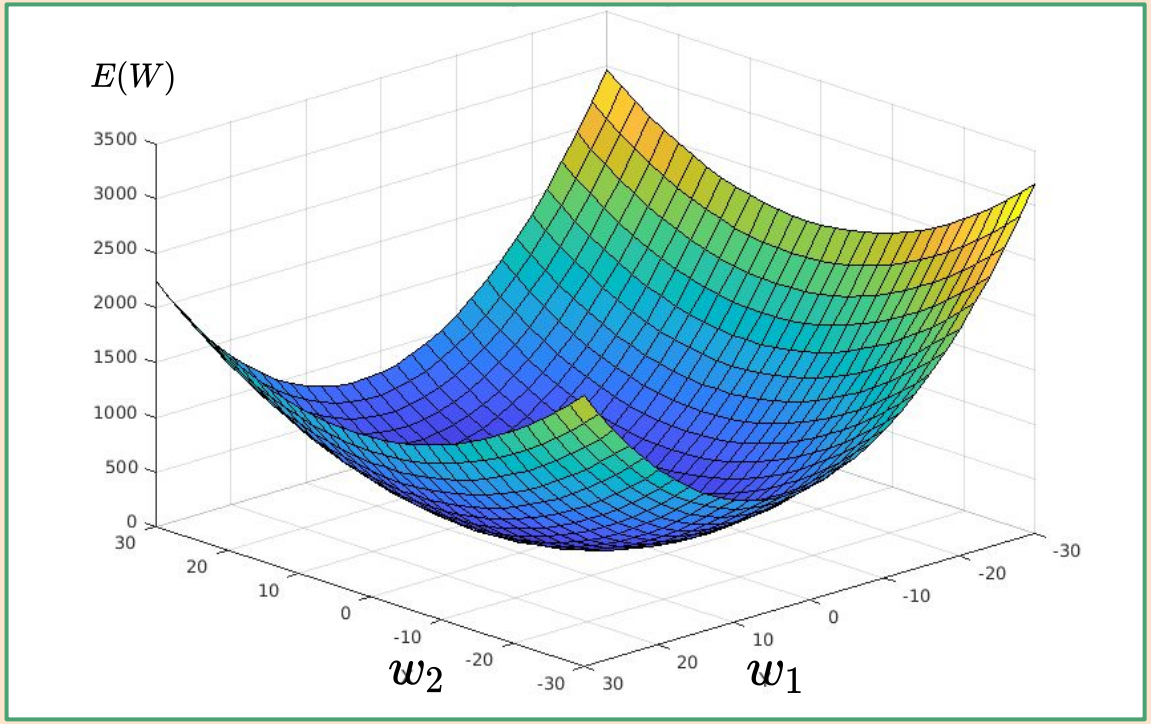
$w_{ij}^{(l)}$ : Connect neuron $i$ in layer $L$ to neuron $j$ in layer $L+1$

# Gradient Descent



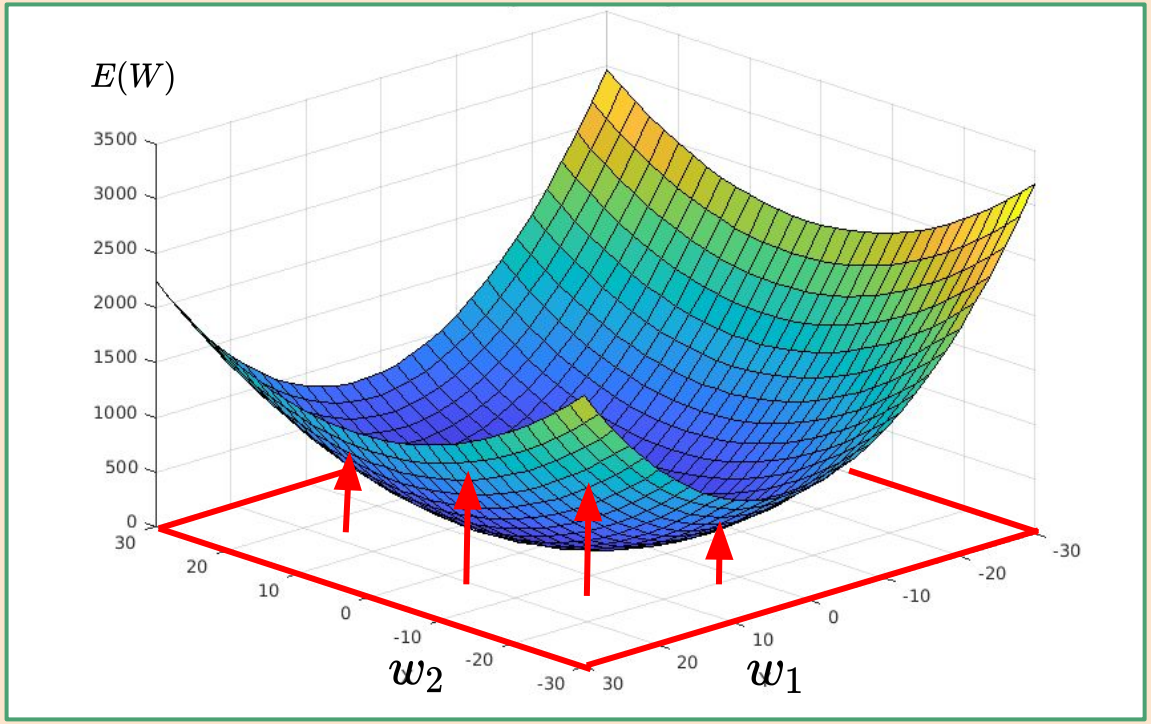
# Understanding gradient descent

- In this example, we use gradient descent algorithm to adjust neural network weights
- To understand gradient descent, let's start with understanding the idea of cost function landscape.
- Given a set of x and y, we can plot cost function as a function of $w_1$ and $w_2$



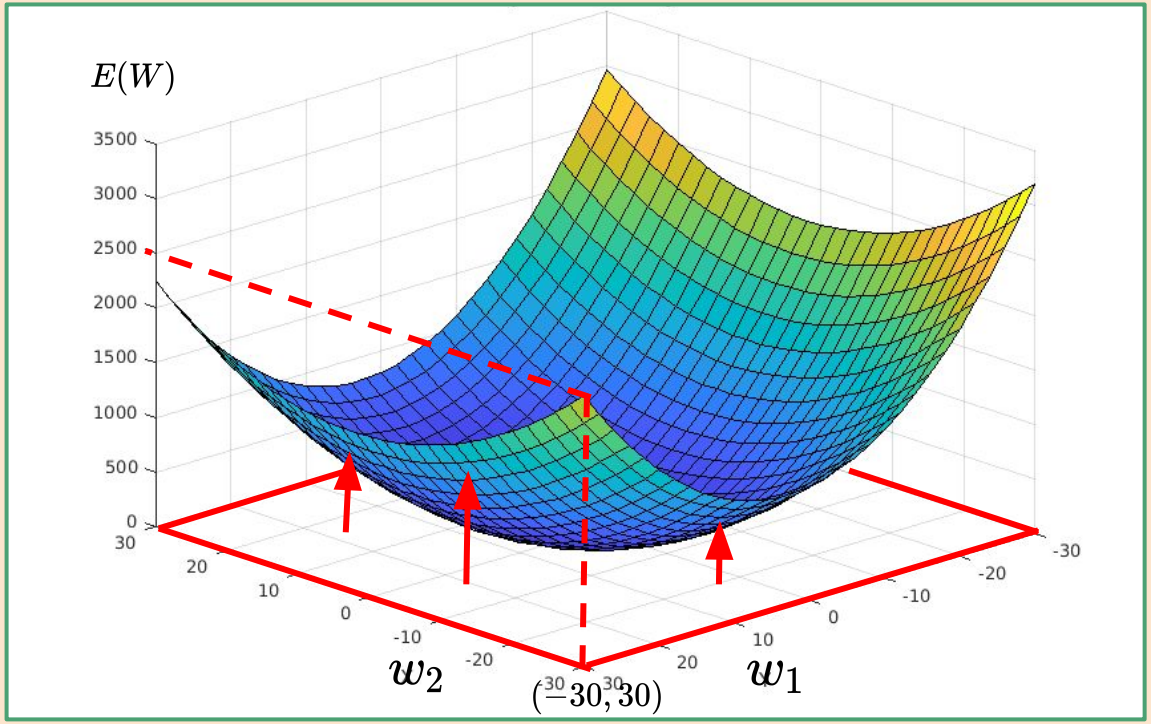
# Understanding gradient descent

- In this example, we use gradient descent algorithm to adjust neural network weights
- To understand gradient descent, let's start with understanding the idea of cost function landscape.
- Given a set of x and y, we can plot cost function as a function of $w_1$ and $w_2$

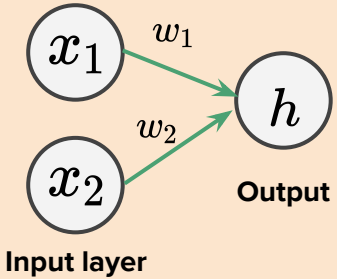


# Understanding gradient descent

- In this example, we use gradient descent algorithm to adjust neural network weights
- To understand gradient descent, let's start with understanding the idea of cost function landscape.
- Given a set of x and y, we can plot cost function as a function of $w_1$ and $w_2$



# Cost function landscape




$$h = f(\sum_j w_j x_j) = f(x_1 w_1 + x_2 w_2)$$

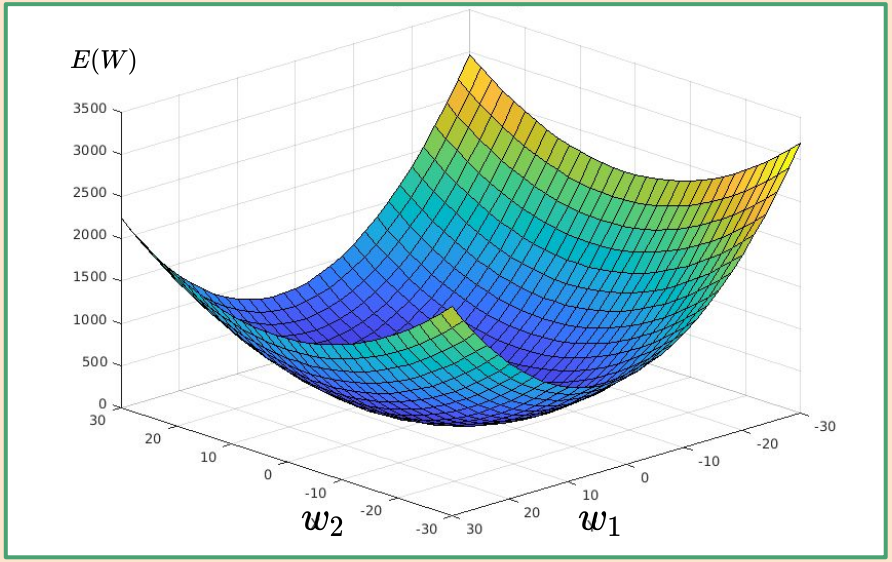
$$\textit{Cost function}$$

$$E(W) = \sum_i^n E^i$$

$$= \sum_i^n (h_i - y_i)^2$$

$$= \sum_i^n ((w_1 x_1^i + w_2 x_2^i) - y_i)^2$$

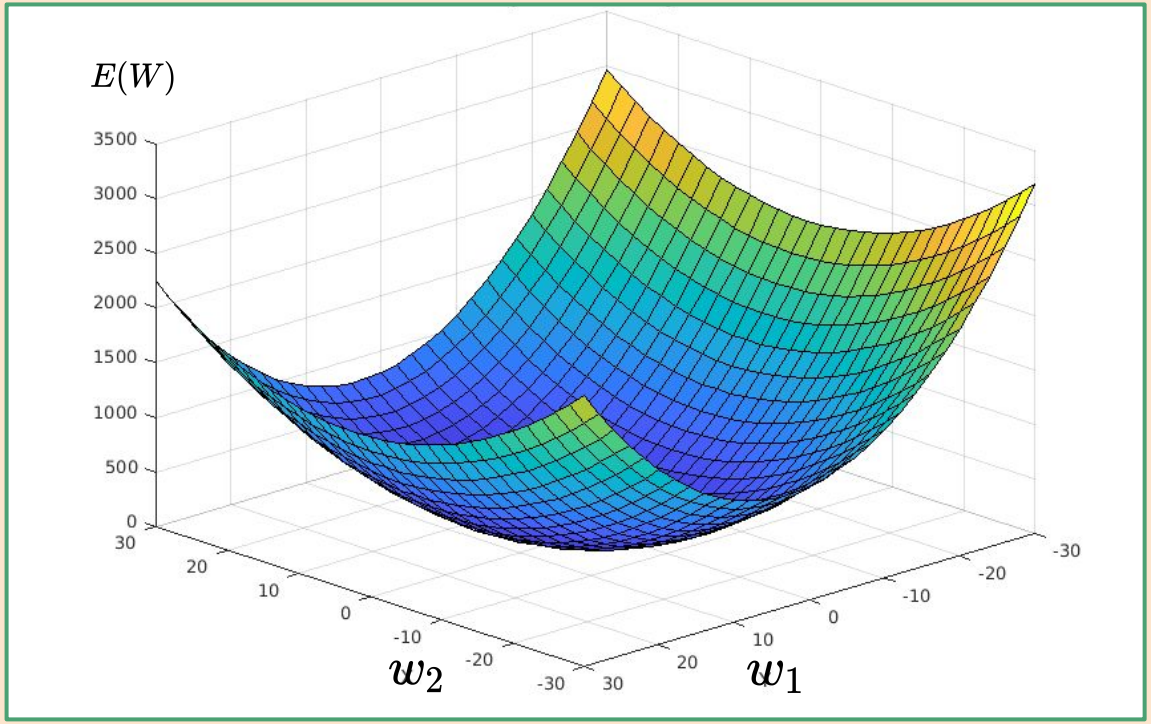
$n$ : Number of samples



# Questions

Answer the following questions about the graph. Just eyeball the graph and give a rough estimate answer.

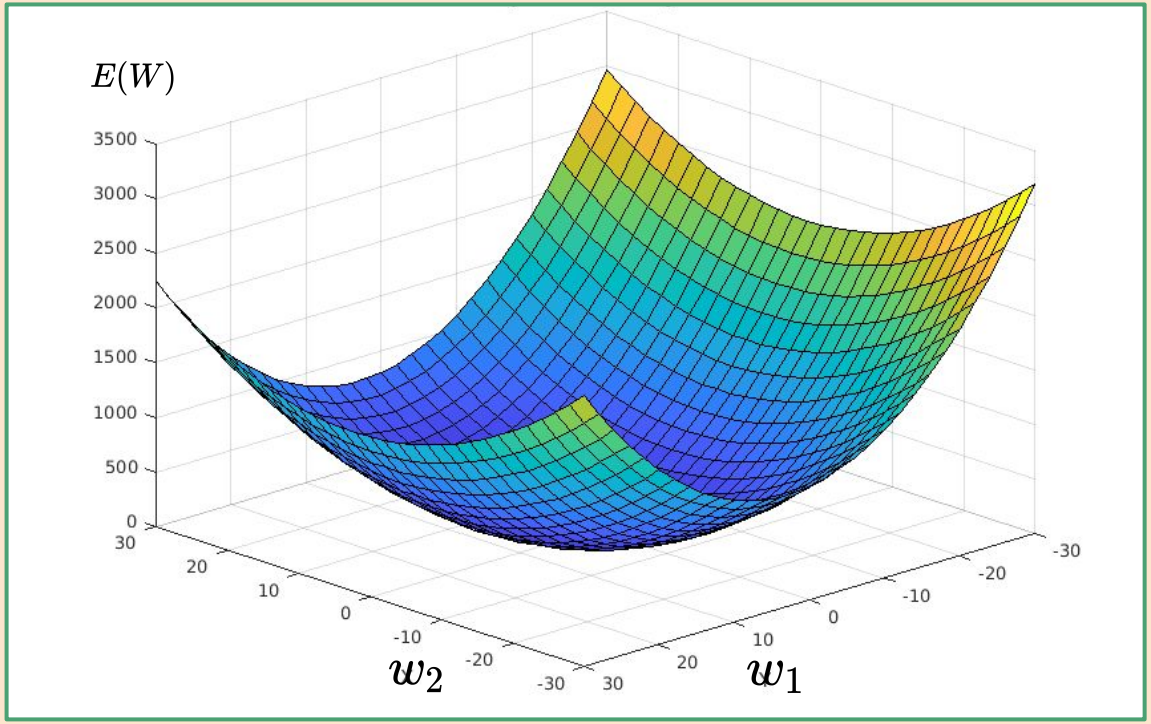
- What $w_1$ and $w_2$ correspond to the bottom of the bowl.
- What value of E will you get at the bottom of the bowl.
- Will the bowl look different if our training set(x and y) are different?



# Questions

Answer the following questions about the graph. Just eyeball the graph and give a rough estimate answer.

- What $w_1$ and $w_2$ correspond to the bottom of the bowl.
- What value of E will you get at the bottom of the bowl.
- Will the bowl look different if our training set(x and y) are different? **Yes!!!**



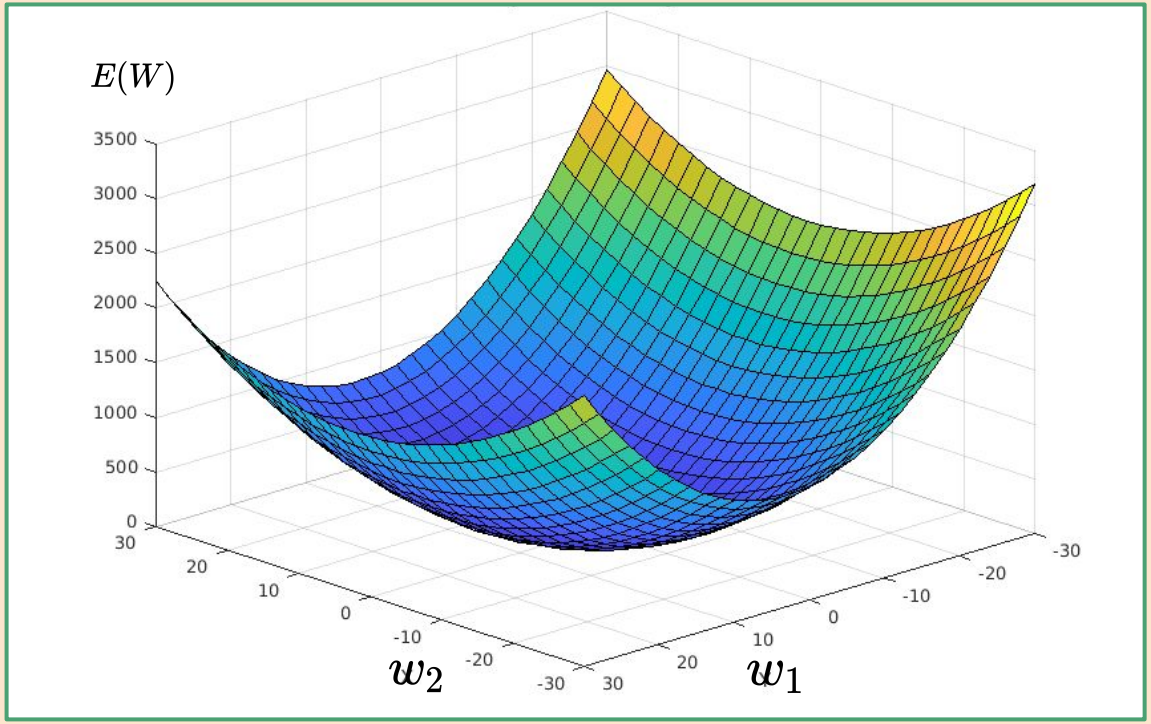

$$Cost\ function$$

$$E(W) = \sum_i^n ((w_1 x_1^i + w_2 x_2^i) - y_i)^2$$

# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

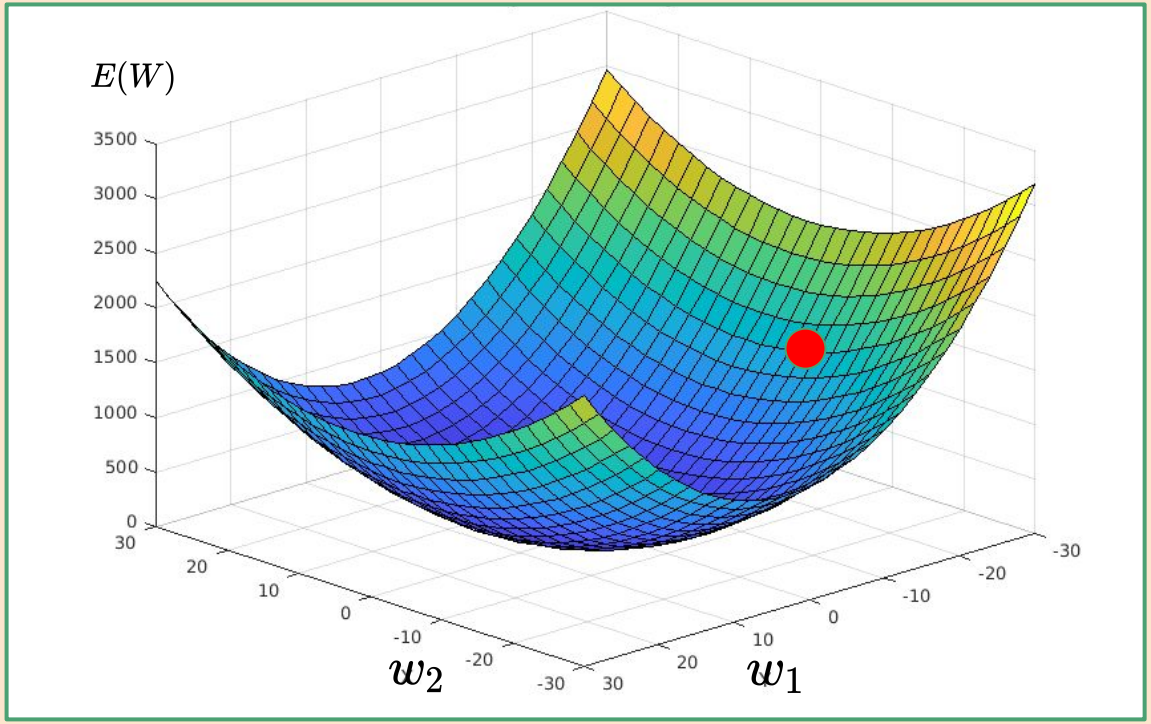
- Pick any pair of w1 and w2, getting dropped at any point in the landscape.
- Find a gradient at that point.
- Gradient $- \bigtriangledown E(w_1, w_2)$ will always point to the steepest direction down the bowl.



# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

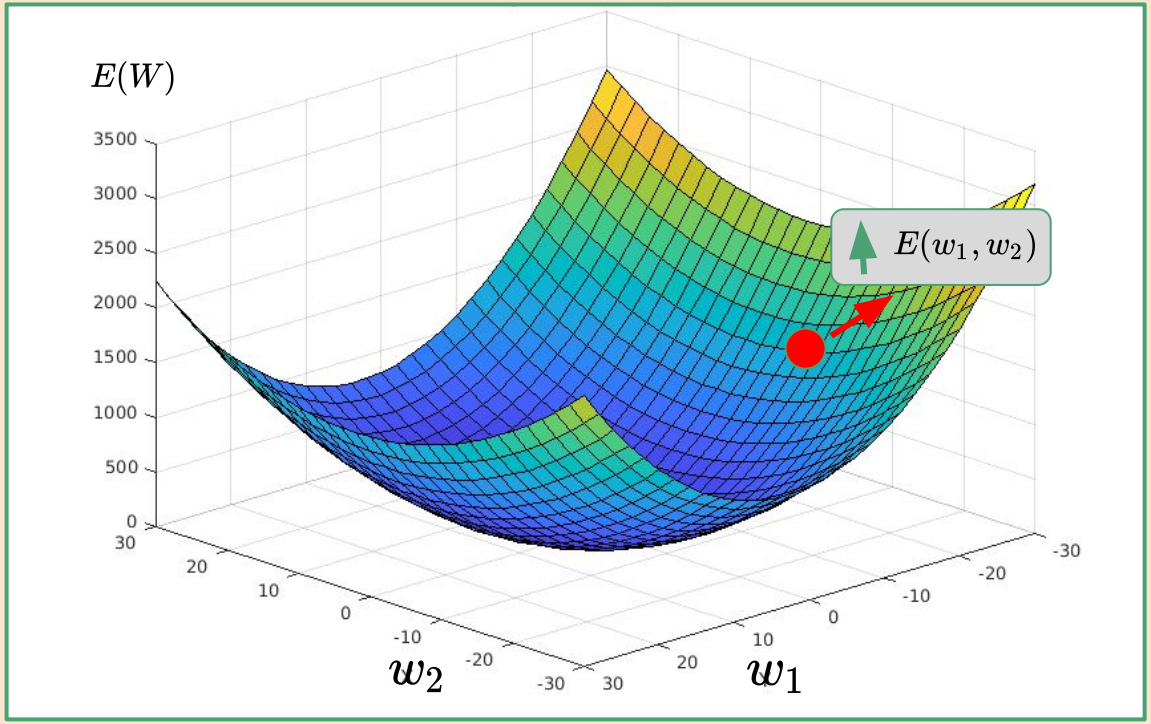
- Pick any pair of w1 and w2, getting dropped at any point in the landscape.
- Find a gradient at that point.
- Gradient $- \bigtriangledown E(w_1, w_2)$ will always point to the steepest direction down the bowl.



# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

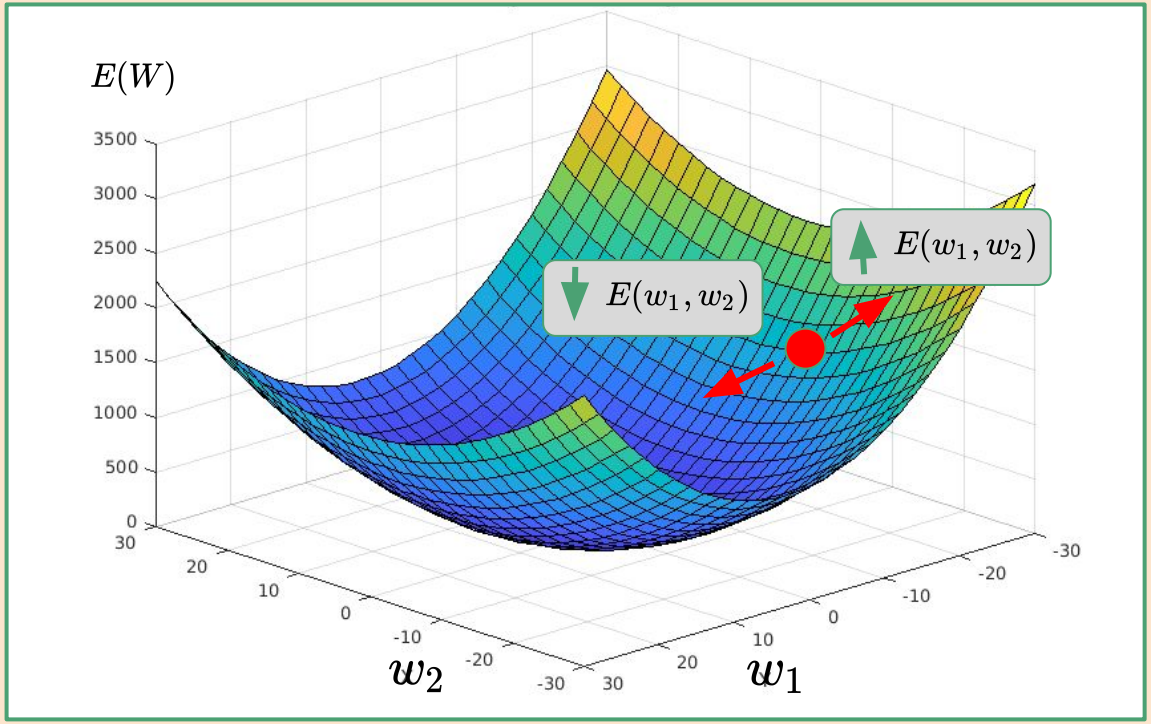
- Pick any pair of w1 and w2, getting dropped at any point in the landscape.
- Find a gradient at that point.
- Gradient $- \bigtriangledown E(w_1, w_2)$ will always point to the steepest direction down the bowl.



# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

- Pick any pair of w1 and w2, getting dropped at any point in the landscape.
- Find a gradient at that point.
- Gradient $- \bigtriangledown E(w_1, w_2)$ will always point to the steepest direction down the bowl.

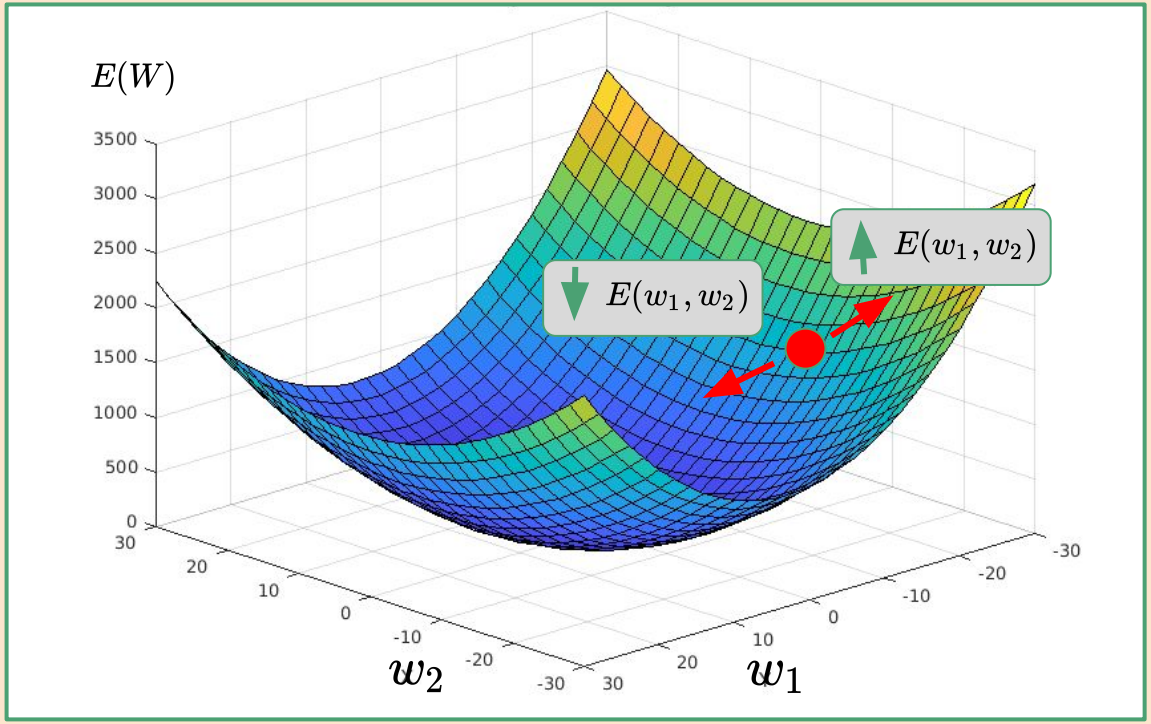


# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

- Pick any pair of w1 and w2, getting dropped at any point in the landscape.
- Find a gradient at that point.
- Gradient $-\bigtriangledown E(w_1, w_2)$ will always point to the steepest direction down the bowl.

$$\begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$

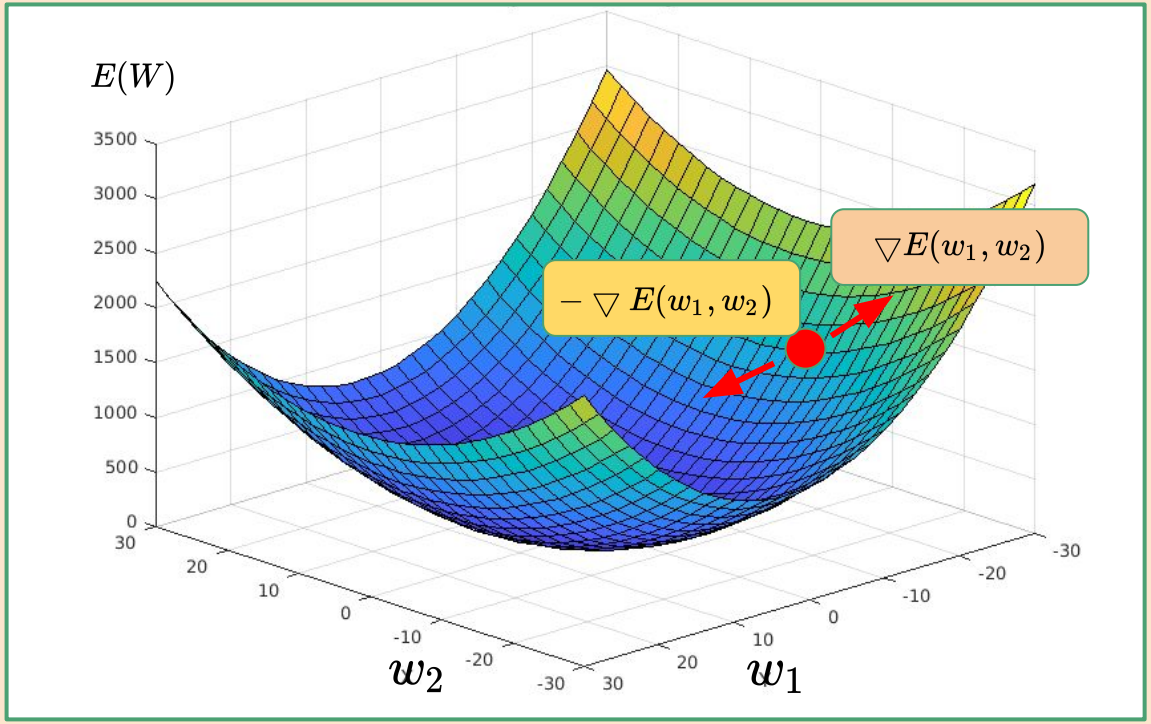


# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

- Pick any pair of w1 and w2, getting dropped at any point in the landscape.
- Find a gradient at that point.
- Gradient $-\bigtriangledown E(w_1, w_2)$ will always point to the steepest direction down the bowl.

$$\begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$

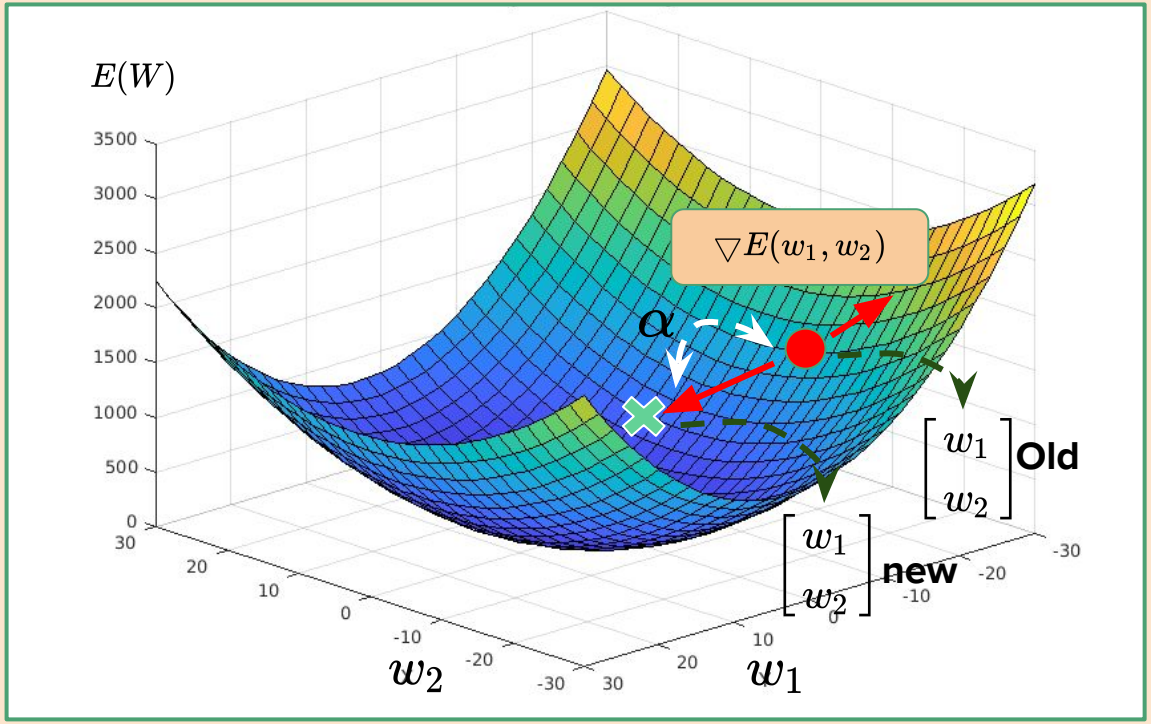


# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

- Take a step down the bowl with the length of the footstep = $\alpha$
- Each step, you will move from one point to another:

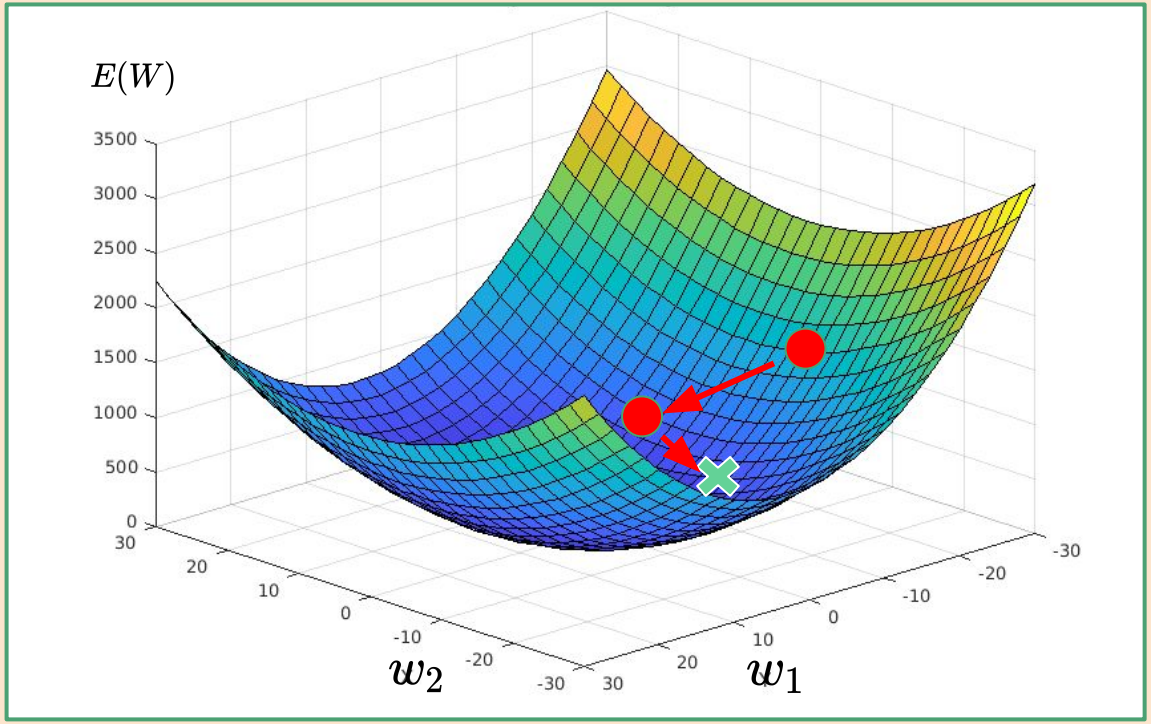
$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longleftarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \bigtriangledown E(w_1, w_2)$$



# Gradient descent algorithm

Getting to the bottom of the bowl with gradient descent.

- Continue walking with the same rule, over and over.
- Eventually, gradient  will be around zero, and your step is tiny.
- No matter where you step at the bottom, no lower points can be found.
- At the point, you have reached the solution.

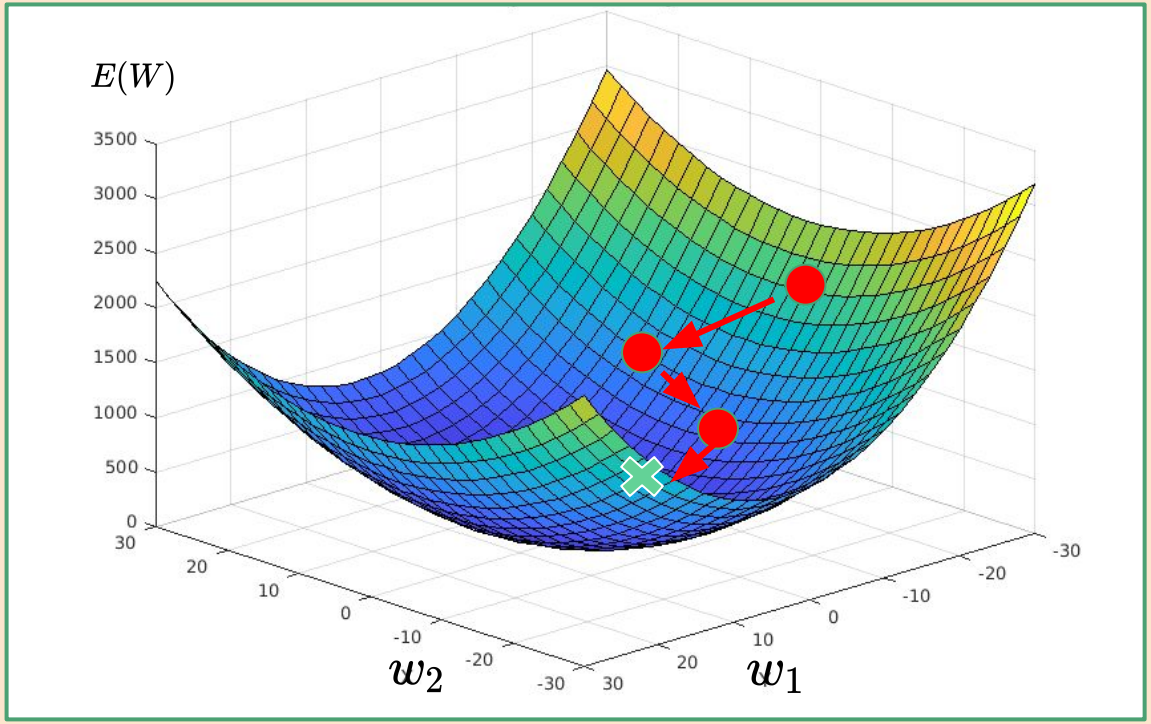


# Gradient descent summary

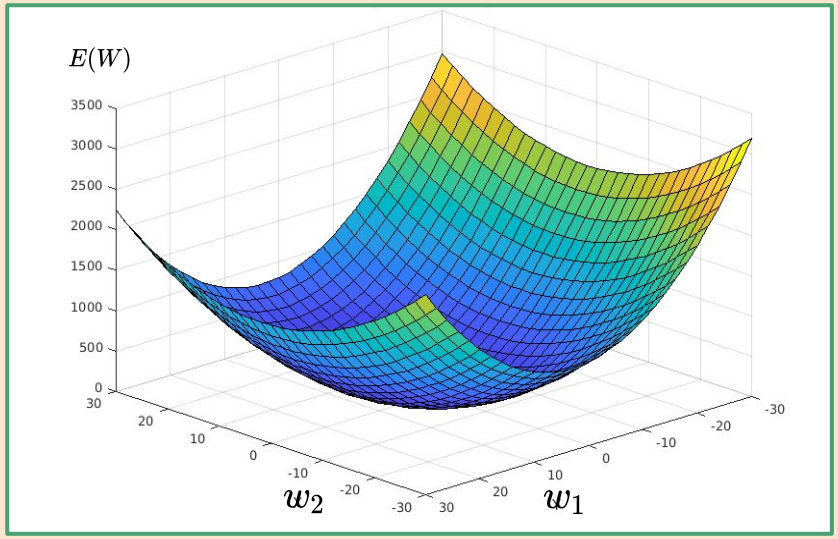
1. Randomly initialize $w_1$ and $w_2$.
2. Calculate the gradient ( $\nabla E(w_1, w_2)$ )
3. Update the algorithm with the following formula:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longrightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \nabla E(w_1, w_2)$$

4. Monitor the cost function. Cost should be lower any time you update weights.
5. When cost function is low enough, stop updating



# Gradient descent quiz




$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longrightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \bigtriangledown E(w_1, w_2)$$

- What value is controlled by $\bigtriangledown E(w_1, w_2)$

1. Direction step
2. Length step
3. Both length and direction

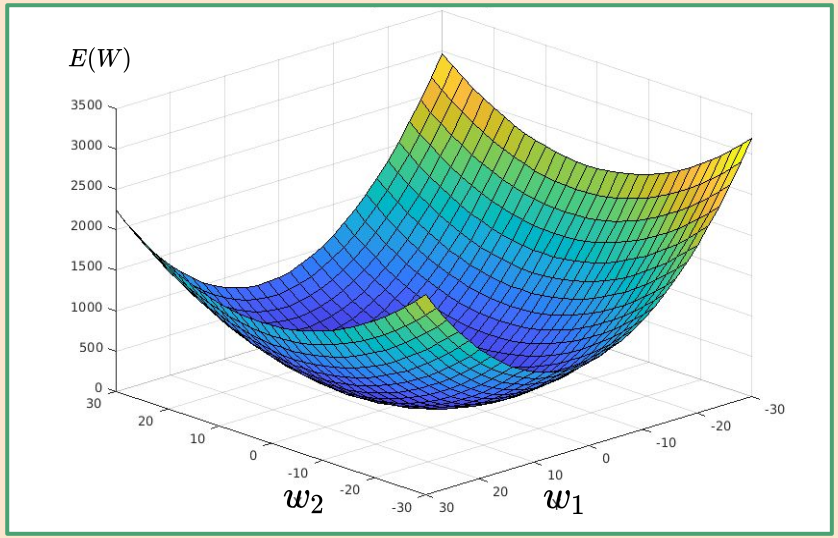
# Gradient descent quiz

Steep slope $\Longrightarrow$ Large $\frac{\partial E}{\partial w_i}$

Shallow slope $\Longrightarrow$ Small $\frac{\partial E}{\partial w_i}$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longrightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \bigtriangledown E(w_1, w_2)$$

$$\text{Slope of landscape} = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$




- What value is controlled by $\bigtriangledown E(w_1, w_2)$

1. Direction step **Yes!!!**
2. Length step
3. Both length and direction

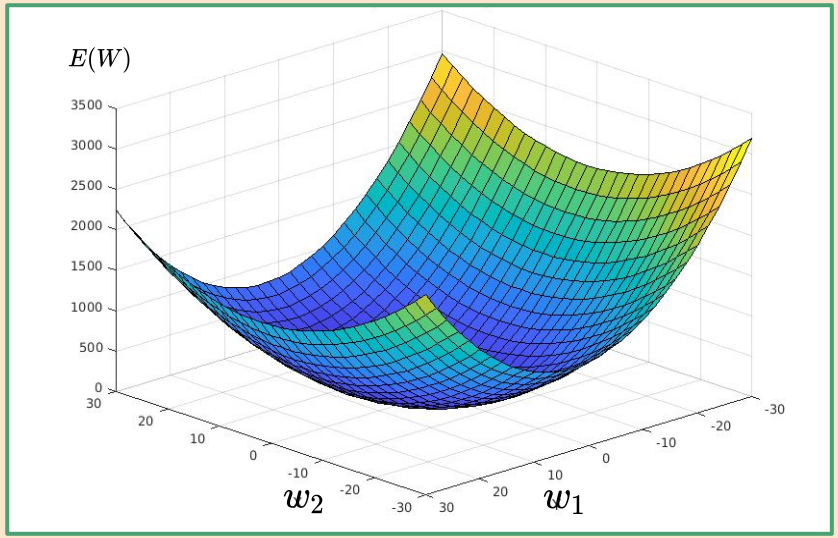
# Gradient descent quiz

Steep slope $\Longrightarrow$ Large $\frac{\partial E}{\partial w_i}$

Shallow slope $\Longrightarrow$ Small $\frac{\partial E}{\partial w_i}$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longrightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \bigtriangledown E(w_1, w_2)$$

$$\text{Slope of landscape} = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$

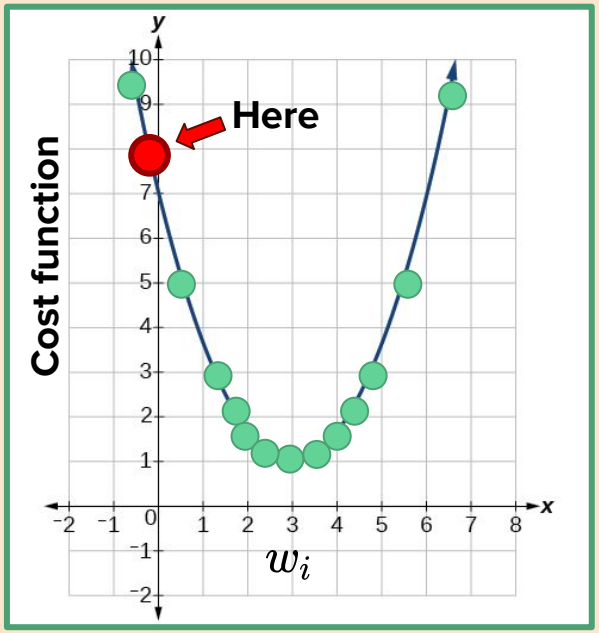



- What value is controlled by $\bigtriangledown E(w_1, w_2)$

1. Direction step **Yes!!!**
2. Length step **Yes!!!**
3. Both length and direction **Yes!!!**

# Gradient descent quiz

$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$

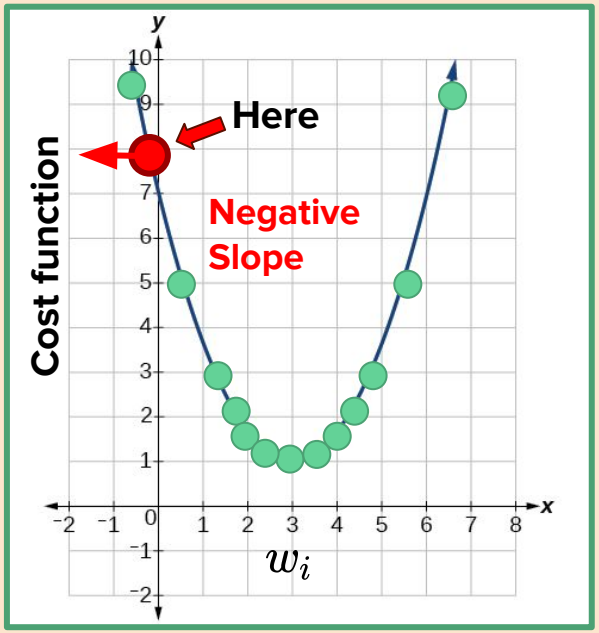



- You are trying to update w by the gradient descent equation above. Your initial condition is at the red point.

1. Will the gradient be positive or negative?
2. Will the algorithm suggests you to increase or decrease the value of w
3. Will your algorithm walks to the right or the left?

# Gradient descent quiz

$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$

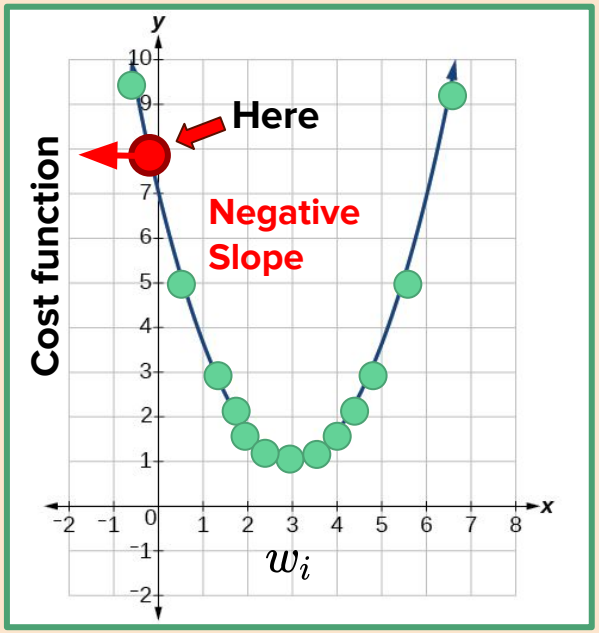



- You are trying to update w by the gradient descent equation above. Your initial condition is at the red point.

1. Will the gradient be positive or negative? **Negative**
2. Will the algorithm suggests you to increase or decrease the value of w
3. Will your algorithm walks to the right or the left?

# Gradient descent quiz

$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$



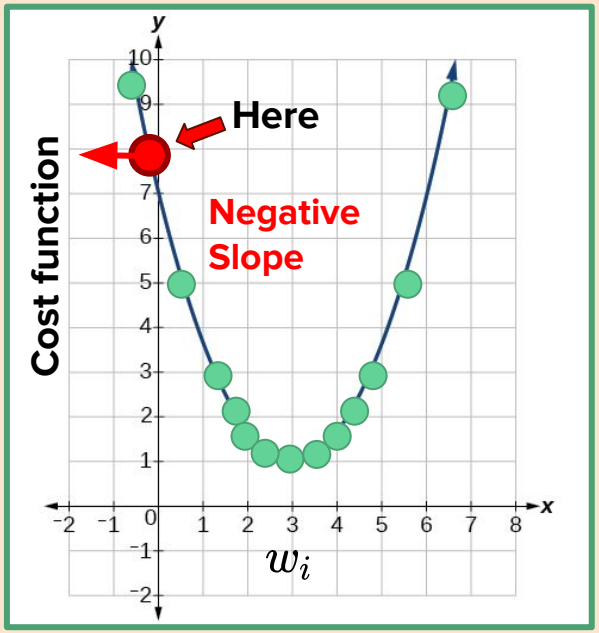

- You are trying to update w by the gradient descent equation above. Your initial condition is at the red point.

1. Will the gradient be positive or negative? **Negative**
2. Will the algorithm suggests you to increase or decrease the value of w
3. Will your algorithm walks to the right or the left?

$$W - \alpha(-G) = W + \alpha(G) \Longrightarrow W \uparrow$$

# Gradient descent quiz

$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$



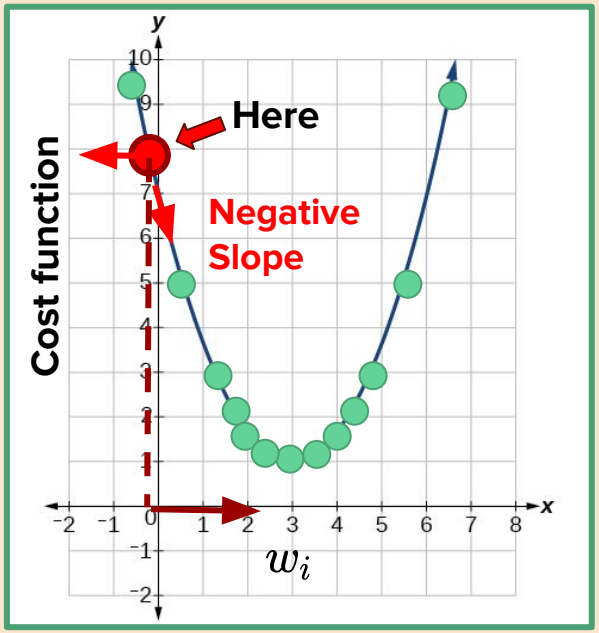

- You are trying to update w by the gradient descent equation above. Your initial condition is at the red point.

1. Will the gradient be positive or negative? **Negative**
2. Will the algorithm suggests you to increase or decrease the value of w **Increase**
3. Will your algorithm walks to the right or the left?

$$W - \alpha(-G) = W + \alpha(G) \Longrightarrow W \uparrow$$

# Gradient descent quiz

$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$

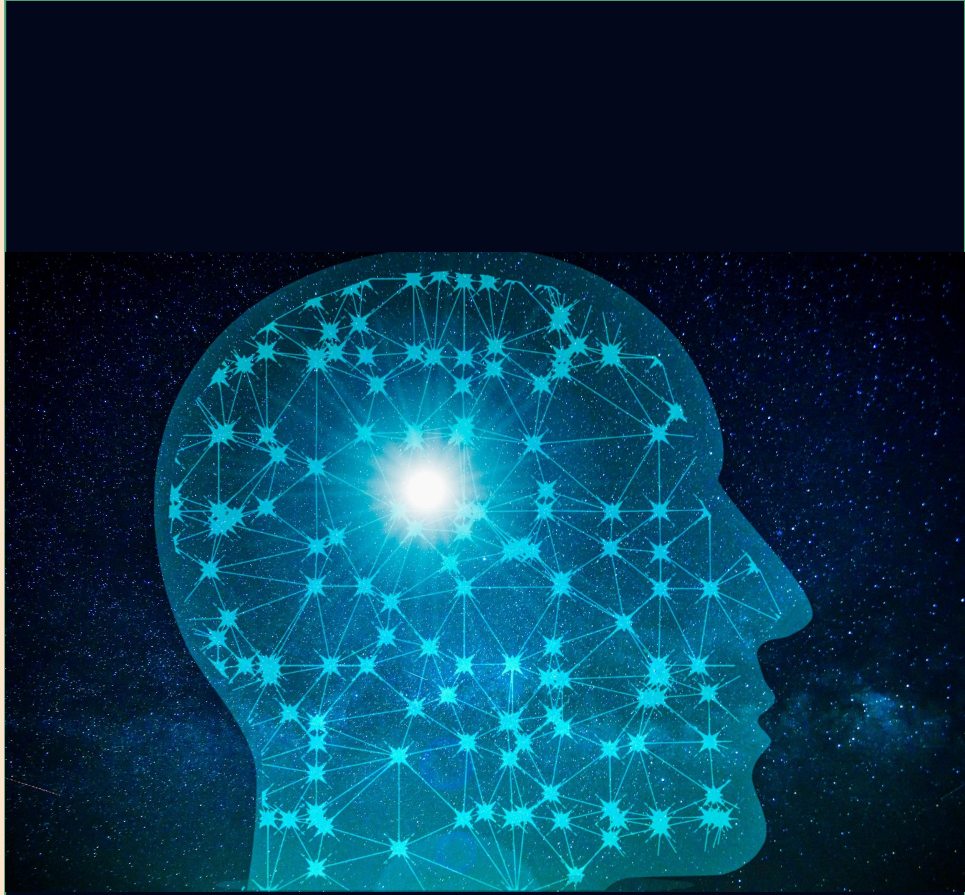



- You are trying to update w by the gradient descent equation above. Your initial condition is at the red point.

1. Will the gradient be positive or negative? **Negative**
2. Will the algorithm suggests you to increase or decrease the value of w **Increase**
3. Will your algorithm walks to the right or the left? **Right**

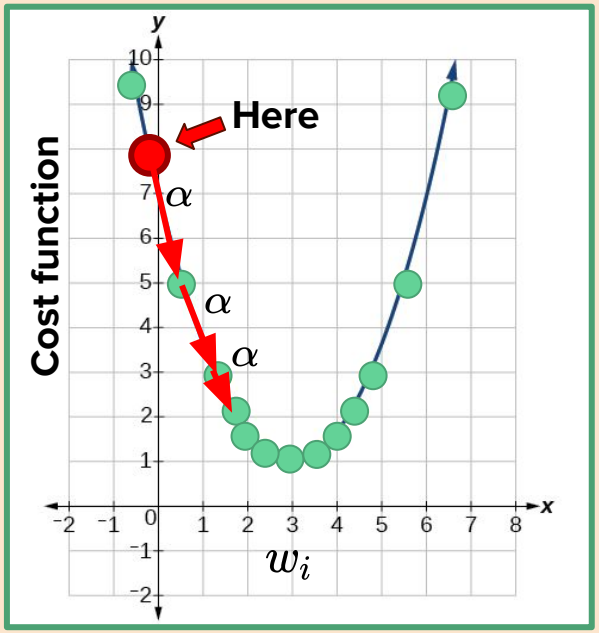
$$W - \alpha(-G) = W + \alpha(G) \Longrightarrow W \uparrow$$

# Details of Neural Network Training



# Picking the right Alpha( $\alpha$ )

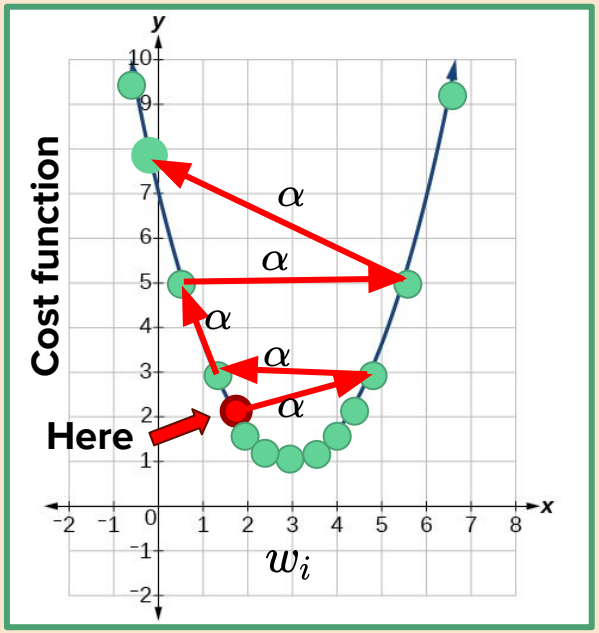
$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$




- Alpha is usually between 0 and 1.
- Large alpha: big step downhill.
- Small alpha: small step.
- You do not want too big or too too small alpha

# Picking the right Alpha( $\alpha$ )

$$w_i := w_i - \alpha \times \frac{\partial E(W)}{\partial w_i}$$




- Alpha is usually between 0 and 1.
- Large alpha: big step downhill.
- Small alpha: small step.
- You do not want too big or too too small alpha

**Big learning rate**

# Different cost function

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

where $m$ is the number of samples

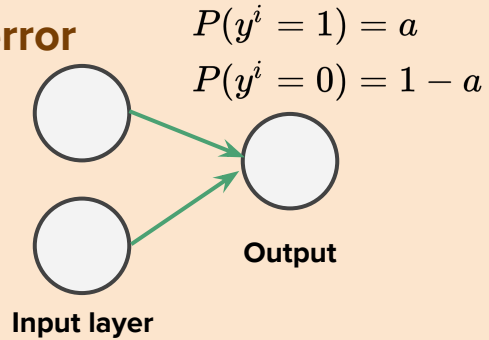
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

where $m$ is the number of samples



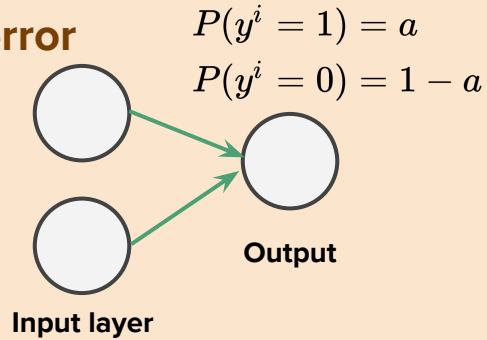
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

where $m$ is the number of samples




**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes

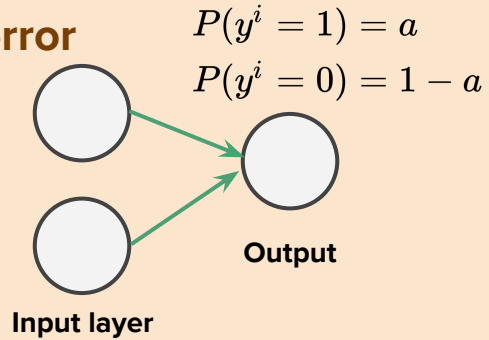
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

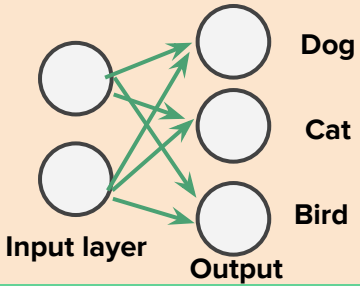
where $m$ is the number of samples




**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes



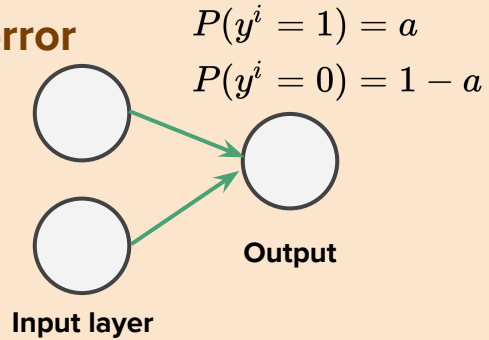
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

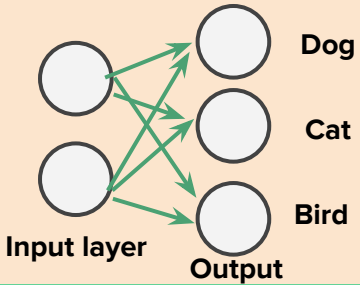
where $m$ is the number of samples




**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes




$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

3 outputs

One-hot-encode categorical variables

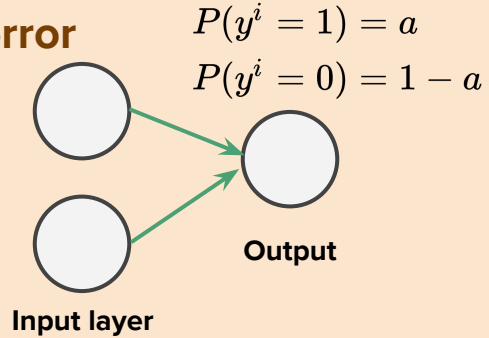
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

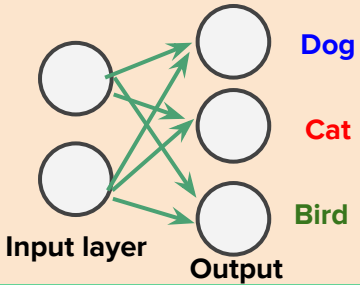
where $m$ is the number of samples




**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes




$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Dog Cat Bird

3 outputs

One-hot-encode categorical variables

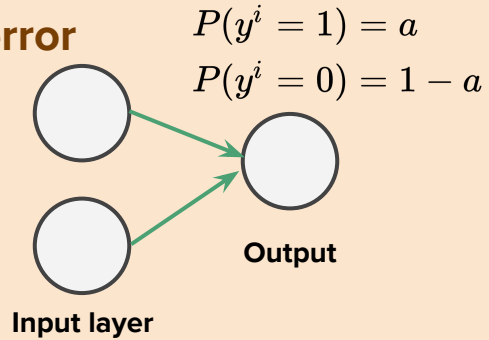
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

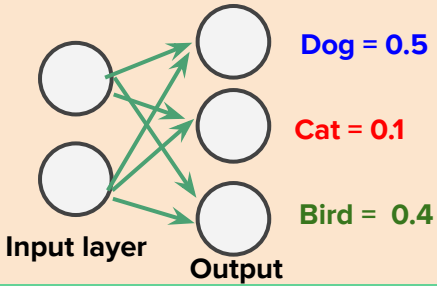
where $m$ is the number of samples




**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes




$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Dog Cat Bird

3 outputs

One-hot-encode categorical variables

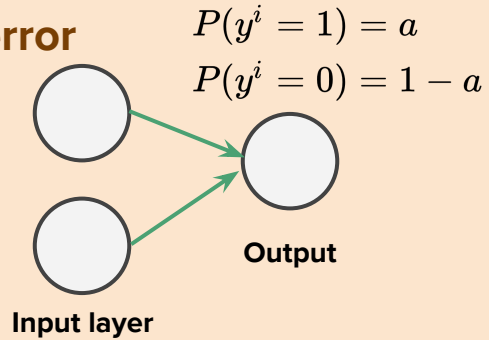
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

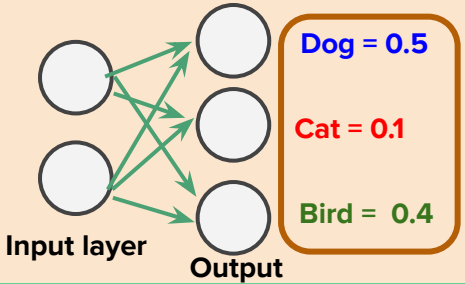
where $m$ is the number of samples

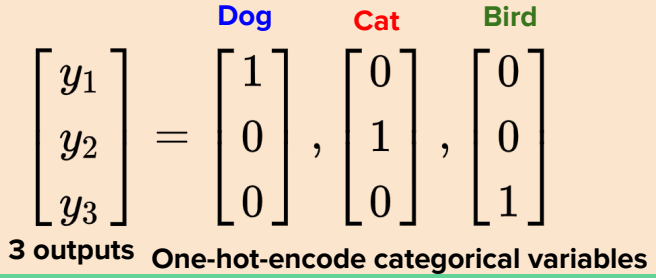



**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes



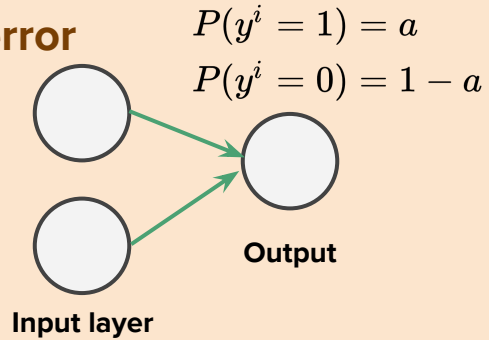
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

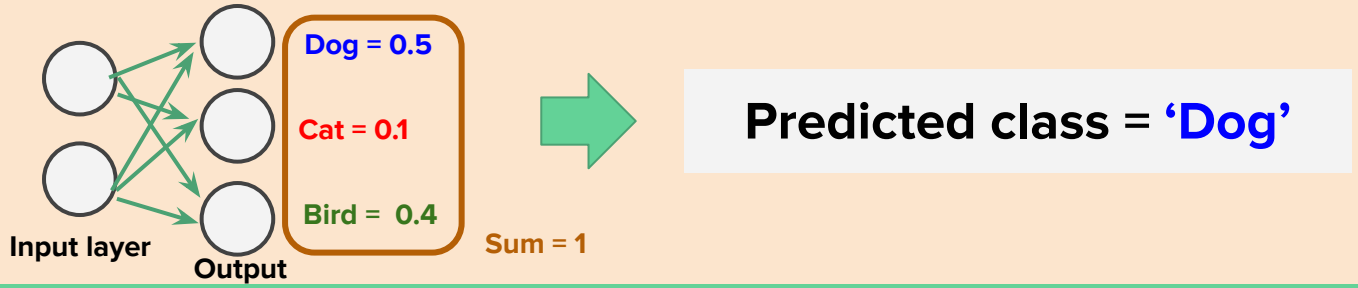
where $m$ is the number of samples




**Classification problem with more than 2 classes: Multinomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

where $m$ is the number of samples and $k$ is the number of outputs or classes



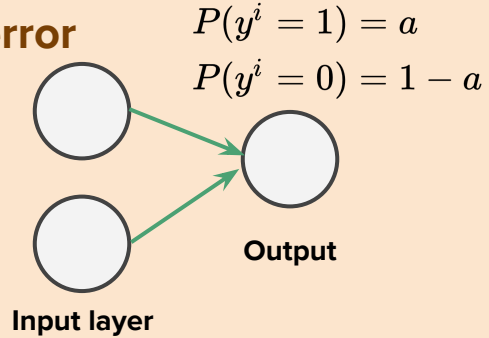
# Different cost function

**Cross entropy error**

**Classification problem with 2 classes: Binomial Log loss**

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

where $m$ is the number of samples



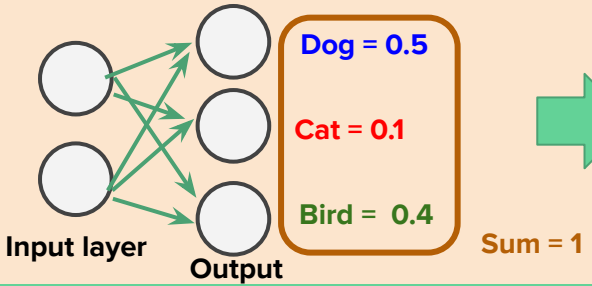

**Classification problem with more than 2 classes: Multinomial Log loss**

Sum # of samples

$$E(W) = -\frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y_c^i log(h_c(x^i)) + (1-y_c^i)log(1-h_c(x^i))]$$

Sum # of classes

where $m$ is the number of samples and $k$ is the number of outputs or classes




**Predicted class = 'Dog'**

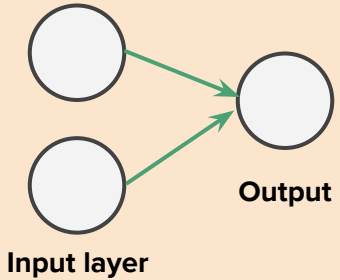
# Different cost function

**Regression**

➢ For single output unit: ➔ Mean square errors (MSE)

$$MSE = E(W) = \frac{1}{m} \sum_{i=1}^{m} [y^i - h(x^i)]^2$$

where $m$ is the number of samples



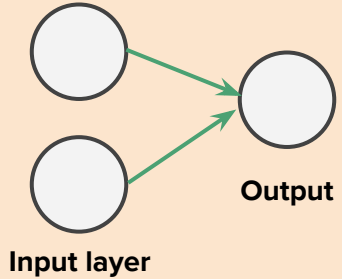
# Different cost function

**Regression**

➢ For single output unit: ➔ Mean square errors (MSE)

$$MSE = E(W) = \frac{1}{m}\sum_{i=1}^{m}[y^i - h(x^i)]^2$$

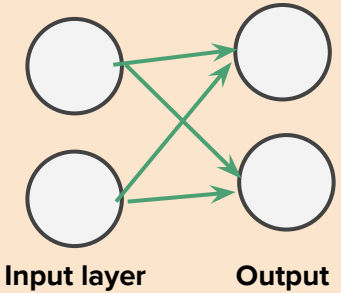
where $m$ is the number of samples




➢ For multiple output units: ➔ Mean square errors (MSE)

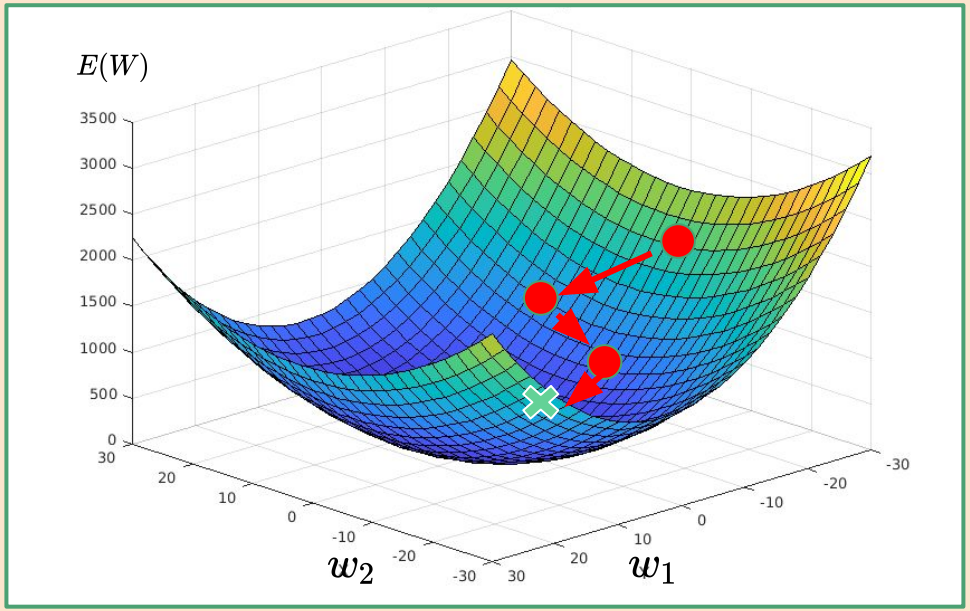
$$MSE = E(W) = \frac{1}{m}\sum_{c=1}^{k}\sum_{i=1}^{m}[y^i - h(x^i)]^2$$

where $m$ is the number of samples and $k$ is the number of outputs



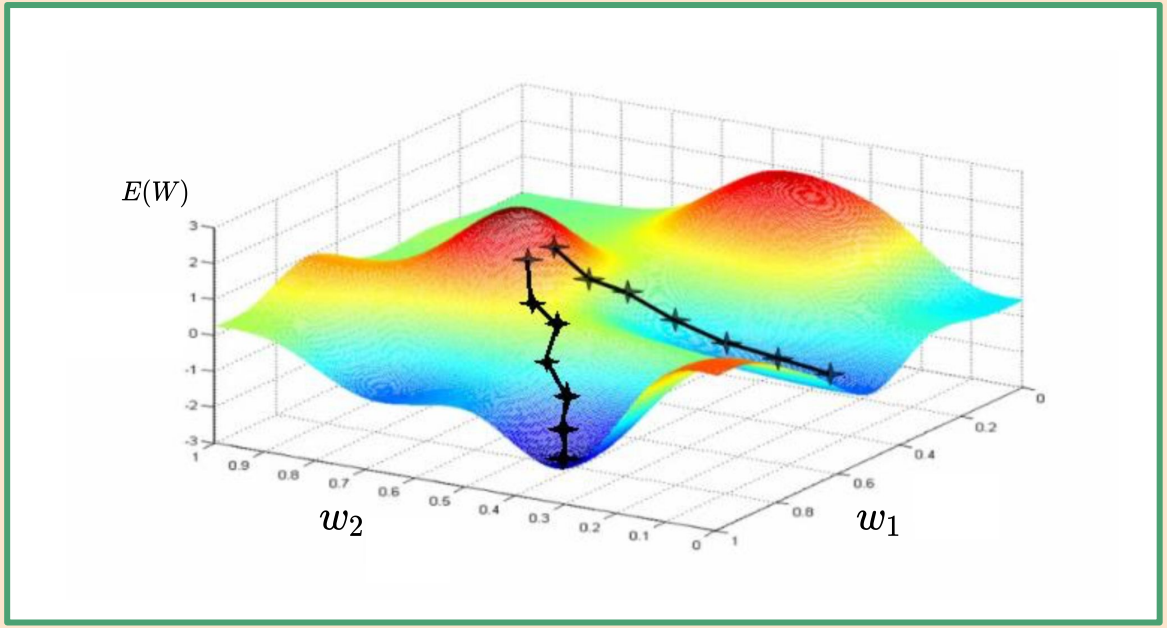
# Single minimum

- For linear regression, you always see convex function like this, which means there's only one lowest point in the bowl.
- And no matter where you start, if you follow the gradient you will definitely reach the bottom.
- Initial condition don't matter.



# Multiple local minimum

- When training neural network, you need to pick different initial conditions and run gradient descent several times.



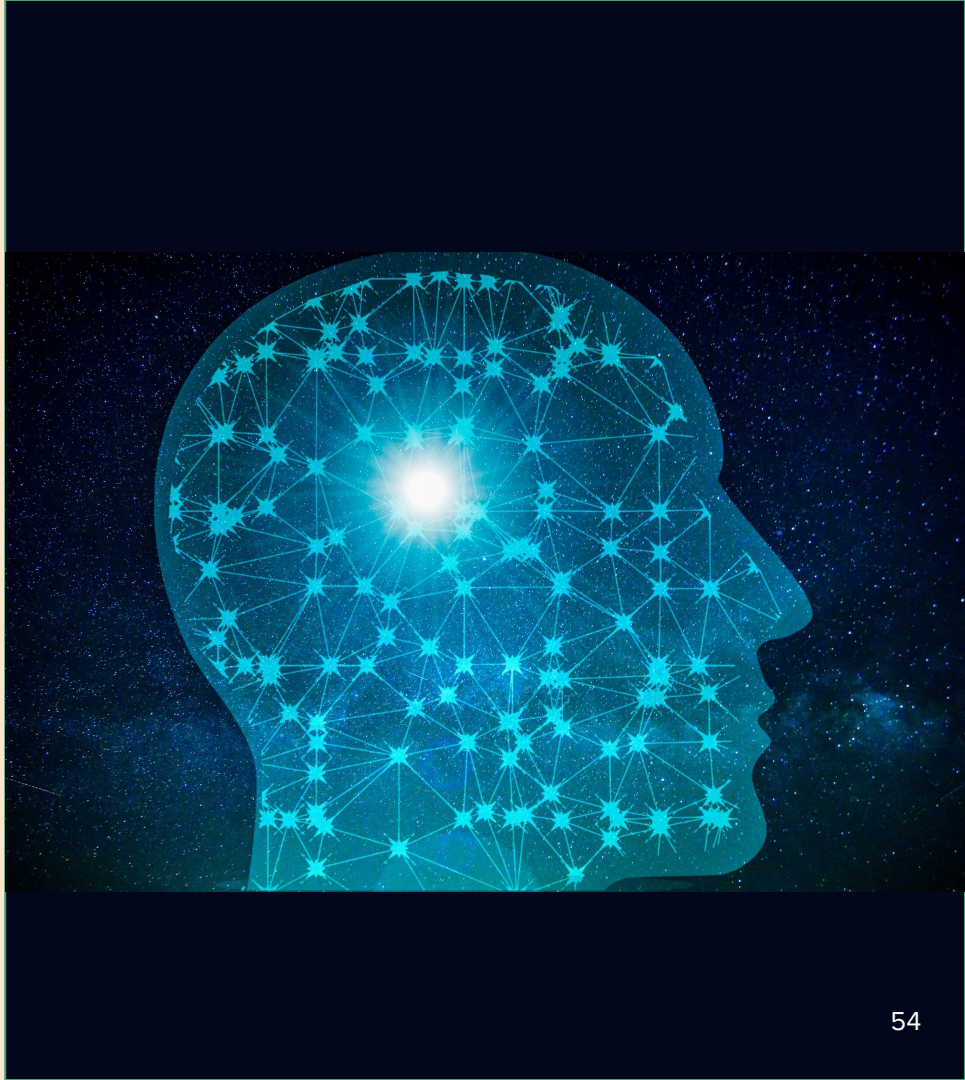
# Stop criteria

- Stop criteria tells the gradient descent algorithm when to stop updating
- We can define stop criteria based on cost function. For example, stop when the difference between the cost in the iteration and last iteration is small.

$$|Cost(i-1) - Cost(i)| < \epsilon$$

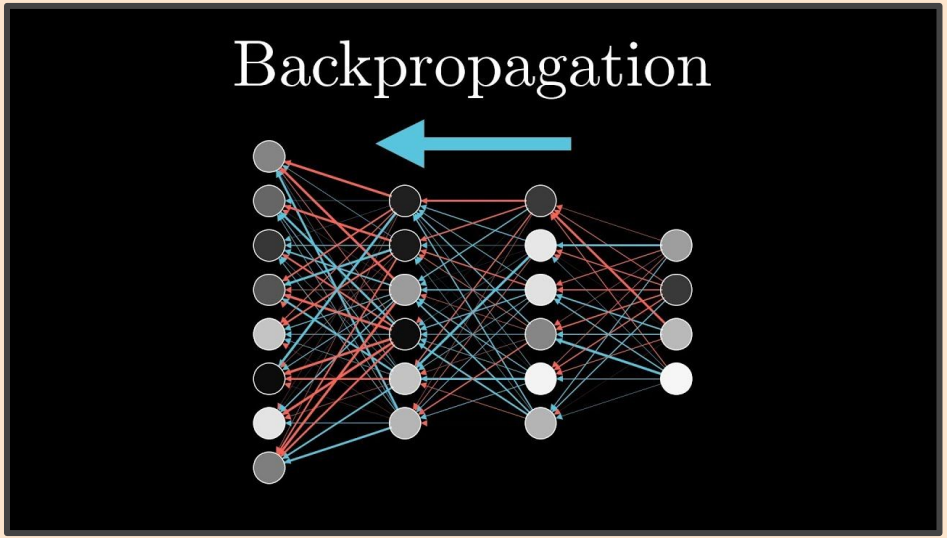
- The value of epsilon is often called tolerance.
- Or we can stop when the gradient is very small.

# Backpropagation Algorithm



# Backpropagation algorithm

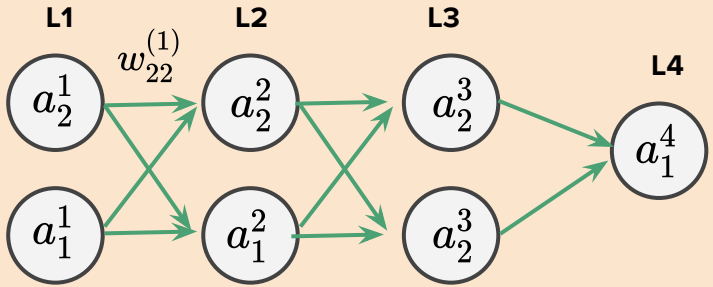
- Back propagation algorithm is created to replace gradient calculation in neural network.
- When you have multiple layers of neural network, gradient calculation becomes super difficult.
- To solve this problem, mathematics modified gradient calculation formula (chain rule) in the form of error passing algorithm.



# Backpropagation algorithm

- Back propagation algorithm is created to replace gradient calculation in neural network.
- When you have multiple layers of neural network, gradient calculation becomes super difficult.
- To solve this problem, mathematics modified gradient calculation formula (chain rule) in the form of error passing algorithm.

**Example**

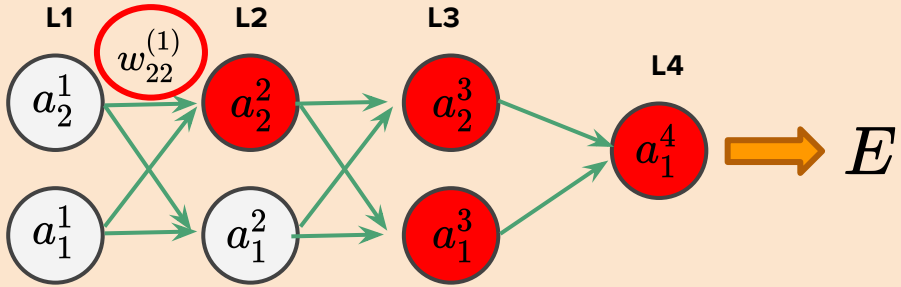



$$\frac{\partial E}{\partial w_{22}^{(1)}} = ?$$

# Backpropagation algorithm

- Back propagation algorithm is created to replace gradient calculation in neural network.
- When you have multiple layers of neural network, gradient calculation becomes super difficult.
- To solve this problem, mathematics modified gradient calculation formula (chain rule) in the form of error passing algorithm.

**Example**

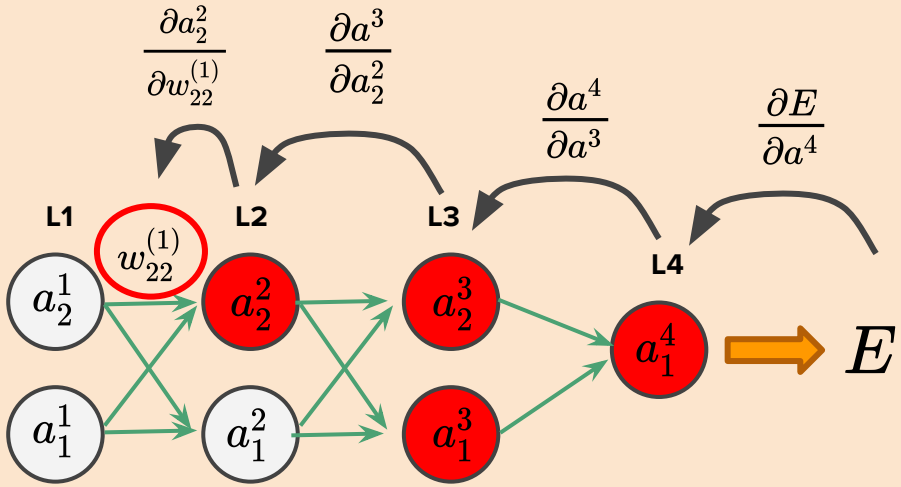



$$\frac{\partial E}{\partial w_{22}^{(1)}} = ?$$

# Backpropagation algorithm

- Back propagation algorithm is created to replace gradient calculation in neural network.
- When you have multiple layers of neural network, gradient calculation becomes super difficult.
- To solve this problem, mathematics modified gradient calculation formula (chain rule) in the form of error passing algorithm.

**Example**




**Chain rule**

$$\frac{\partial E}{\partial w_{22}^{(1)}} = ?$$

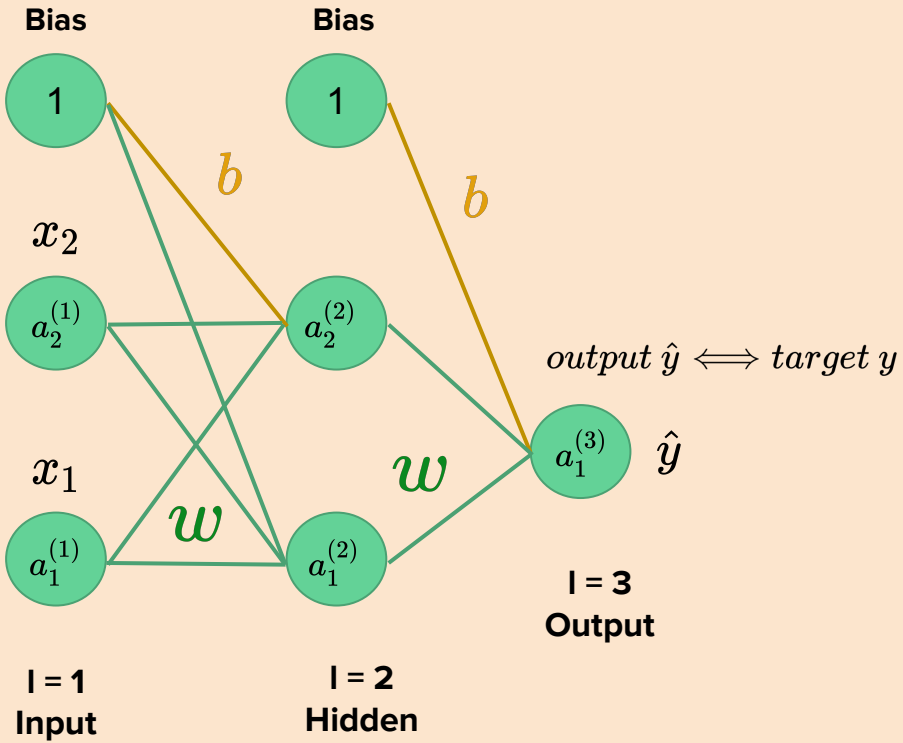
# Backpropagation algorithm example

**Backpropagation concept**

1.Calculate error ($\delta$) at each layer.

2.Assign blames to neurons at previous layer.

3. Use blame to calcutale gradient.



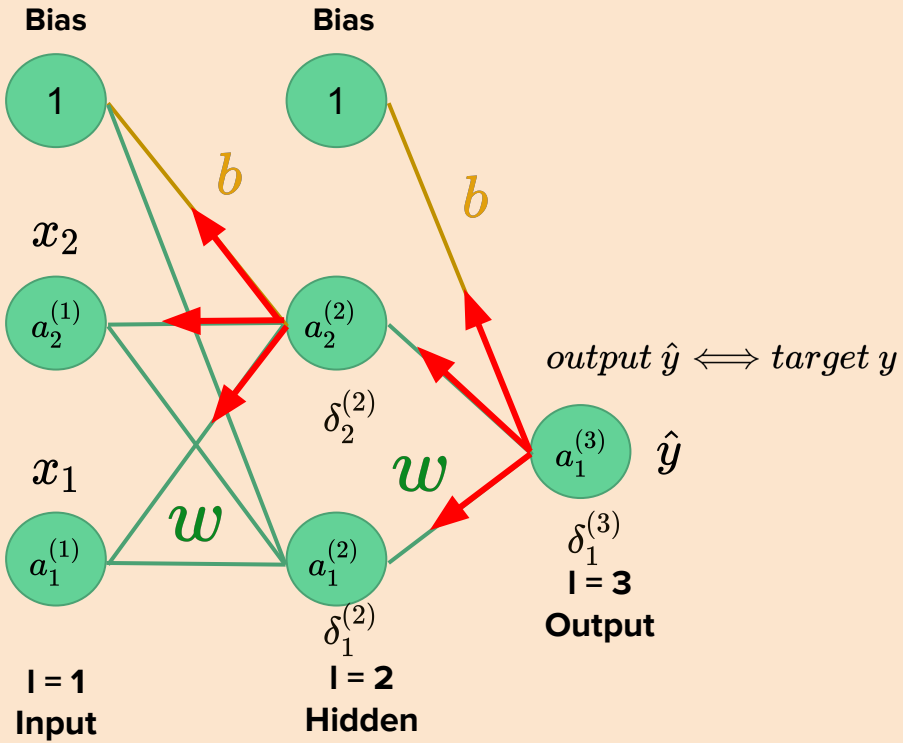
# Backpropagation algorithm example

**Backpropagation concept**

1.Calculate error ($\delta$) at each layer.

2.Assign blames to neurons at previous layer.

3. Use blame to calcutale gradient.



# Backpropagation algorithm example

$$a_j^{(l)} = f(o_j^{(l)}) = f(\sum_i w_{ij}^{(l-1)} a_i^{(l-1)} + b_j^{(l-1)})$$

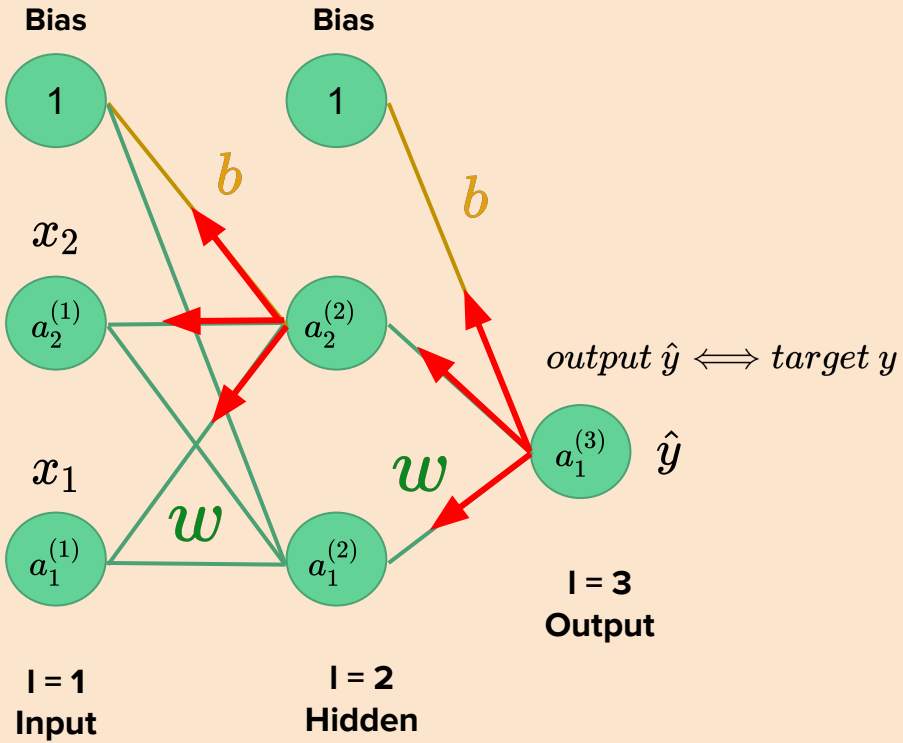
$w_{ij}^{(l)}$ : Connect neuron $i$ in layer $l$

to neuron $j$ in layer $l+1$

$f$ : Activation function

$$MSE = E(W) = \frac{1}{m} \sum_{i=1}^{m} [y^i - \hat{y}^i]^2$$

where $m$ is the number of samples



# Backpropagation algorithm example

$$MSE = E(W) = \frac{1}{m}\sum_{i=1}^{m}[y^i - \hat{y}^i]^2$$

where $m$ is the number of samples

$$Gradient:$$

$$\frac{\partial E(w)}{\partial w_{ij}^{(l)}} = \delta_j^{(l+1)} a_i^{(l)}$$

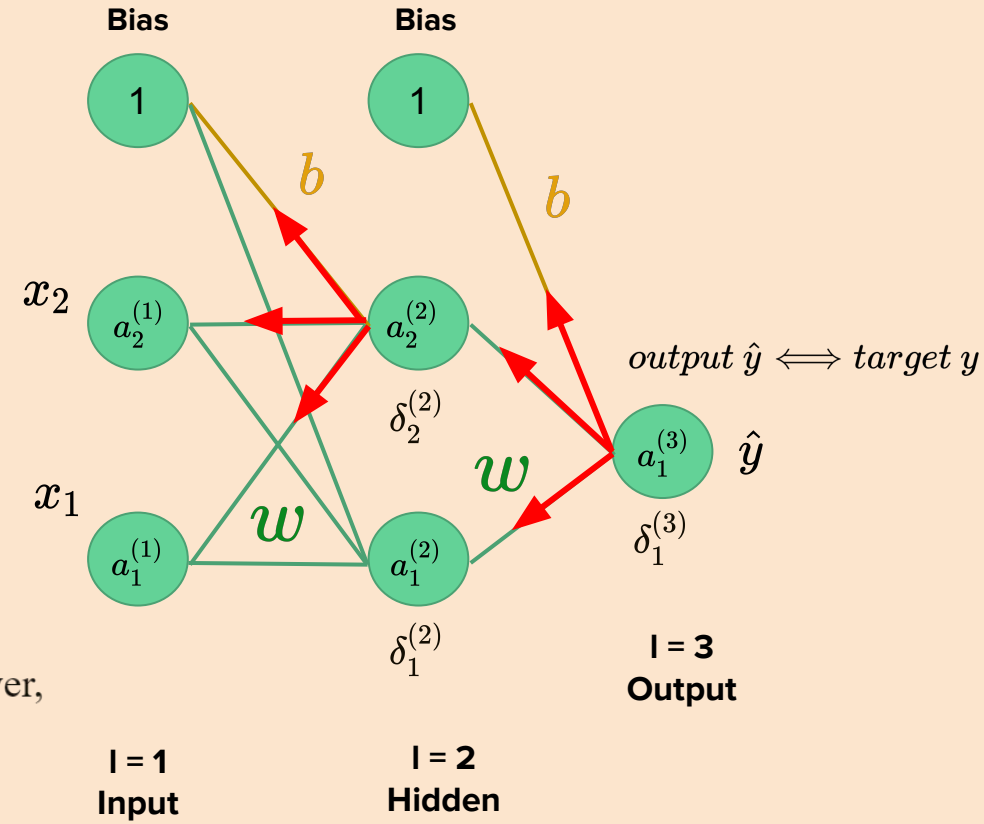
$$\frac{\partial E(w)}{\partial b_j^{(l)}} = \delta_j^{(l+1)}$$

where $\delta_j^{(l)} = \frac{\partial E(w)}{\partial o_j^{(l)}}$ at neuron $j$ in layer,

if $\delta_j^{(l)}$ is in output layer

or $\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \cdot w_{jk}^{(l)} \cdot f'(o_j^{(l)})$ at neuron $j$ in layer,

if $\delta_j^{(l)}$ is in hidden layer

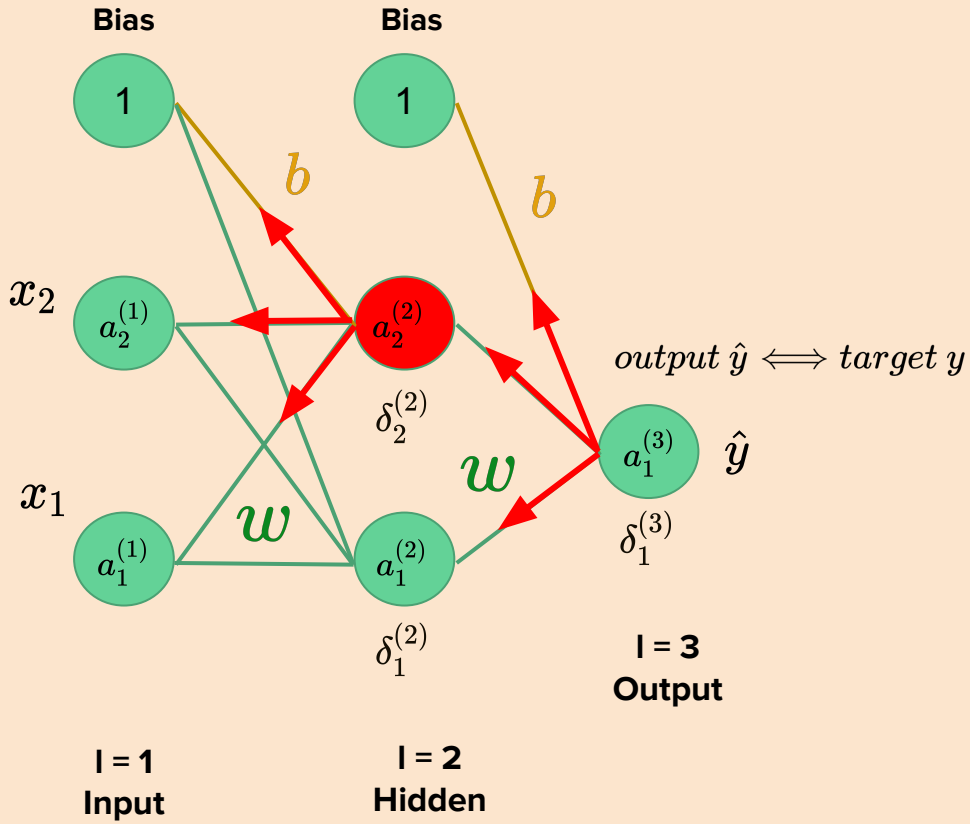


# Backpropagation algorithm example

- **At hidden layer**

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \cdot w_{jk}^{(l)} \cdot f'(o_j^{(l)})$$

$$\delta_2^{(2)} = \sum_k \delta_k^{(3)} \cdot w_{2,k}^{(2)} \cdot f'(o_2^{(2)})$$
$$= \delta_1^{(3)} \cdot w_{2,1}^{(2)} \cdot f'(o_2^{(2)})$$



# Backpropagation algorithm example

- **At output layer**

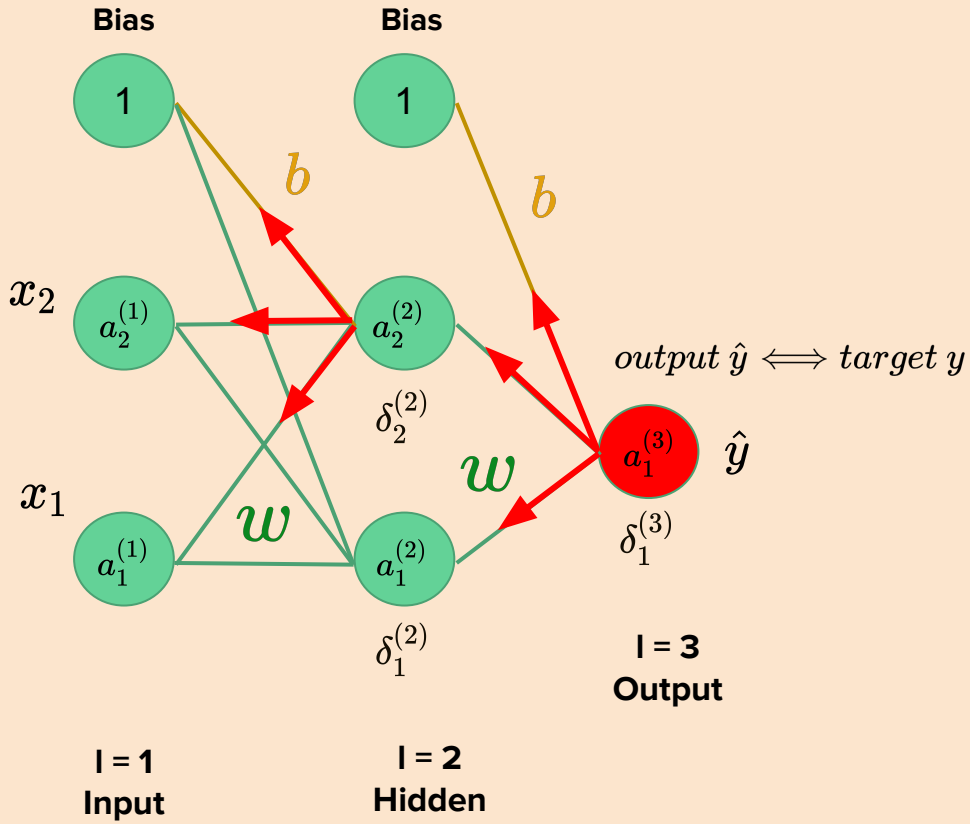
$$\delta_j^{(l)} = \frac{\partial E(w)}{\partial o_j^{(l)}} = \frac{\partial E(w)}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial o_j^{(l)}}$$

$$= \frac{\partial (y - a_j^{(l)})^2}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial o_j^{(l)}}$$

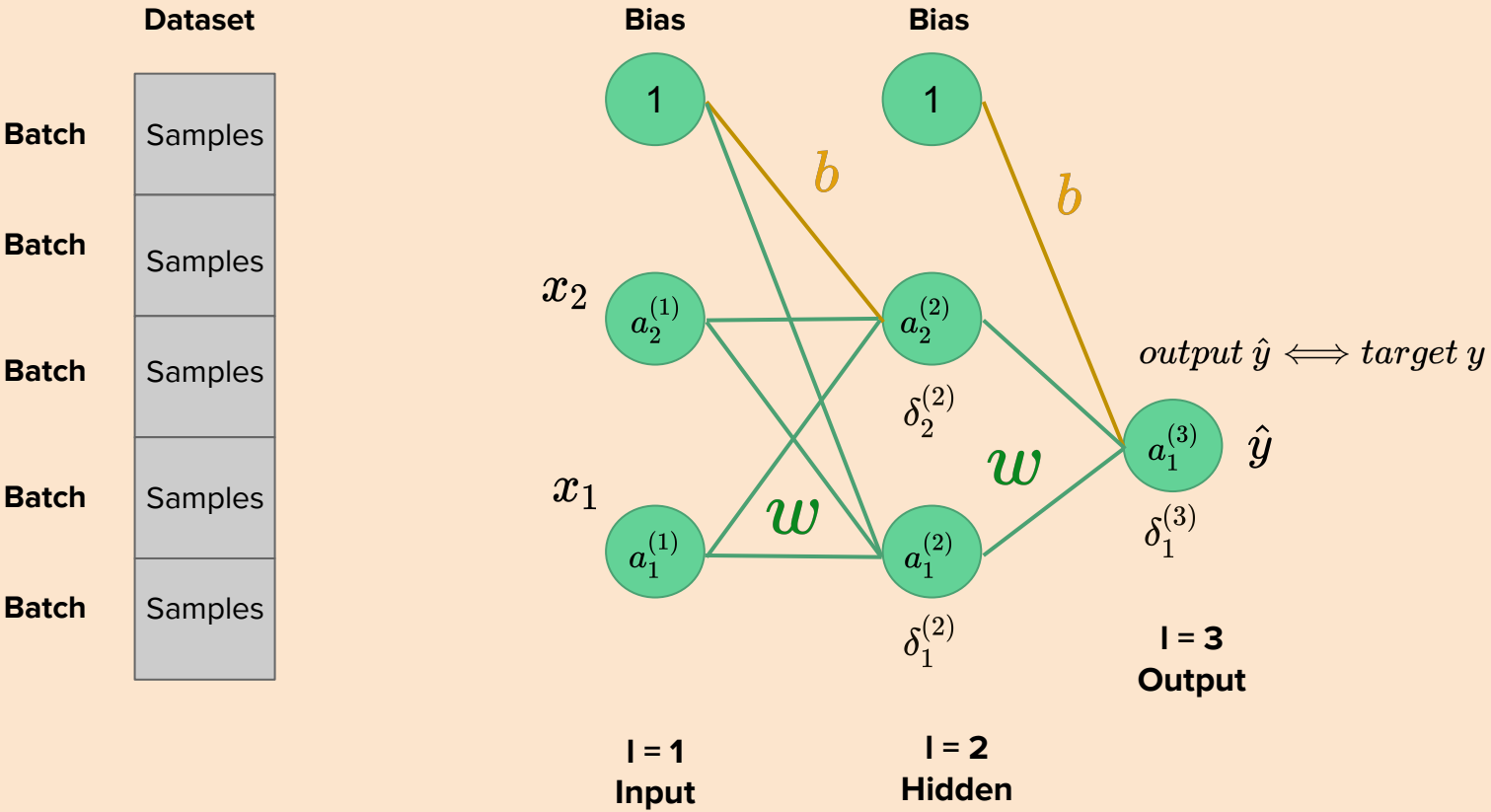
$$= 2(y - a_j^{(l)}) \cdot \frac{\partial f(o_j^{(l)})}{\partial o_j^{(l)}}$$

$$= 2(y - a_j^{(l)}) \cdot f'(o_j^{(l)})$$

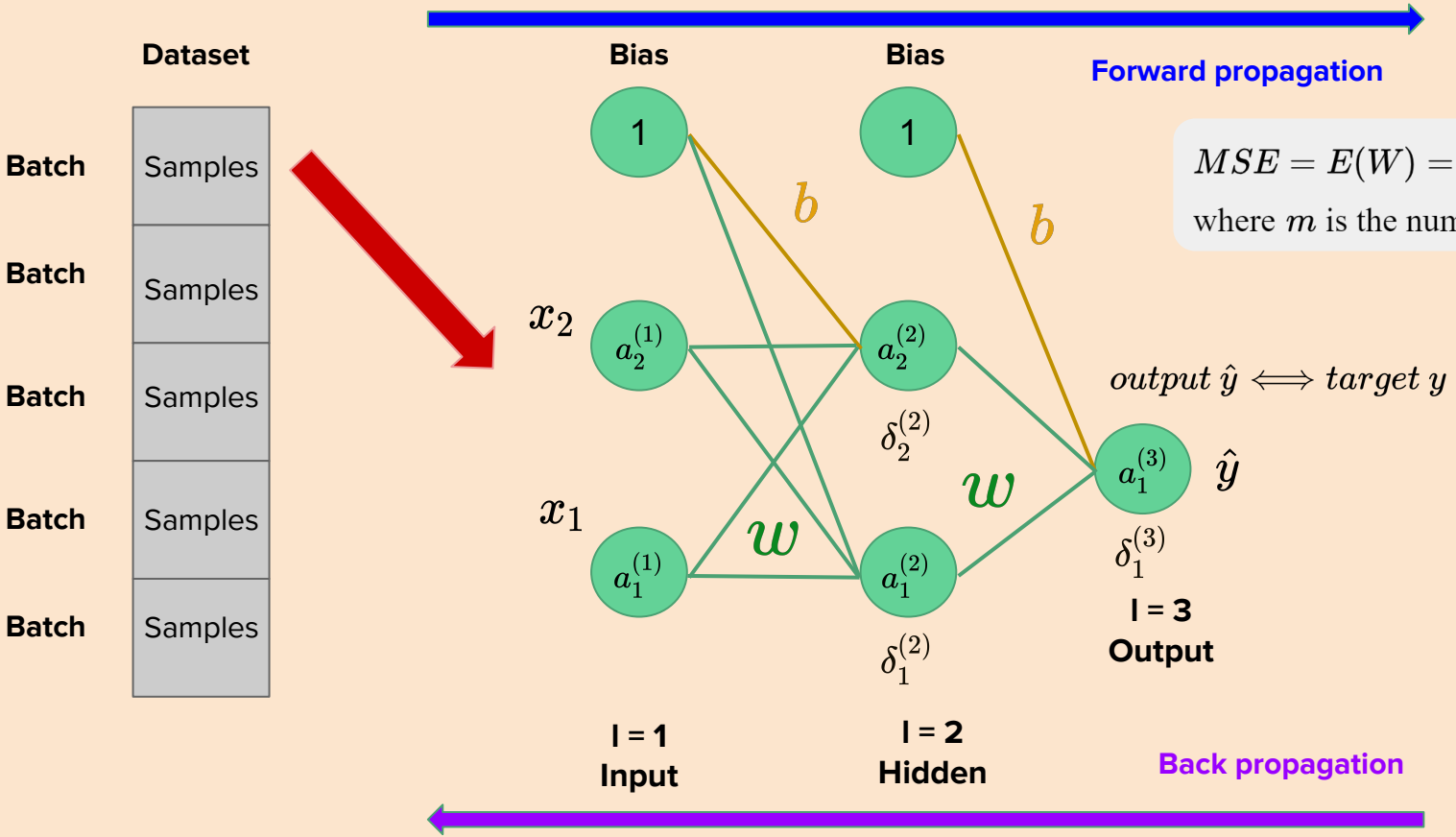
$$\delta_1^{(3)} = 2(y - a_1^{(3)}) \cdot f'(o_1^{(3)})$$



# Backpropagation algorithm example



# Backpropagation algorithm example



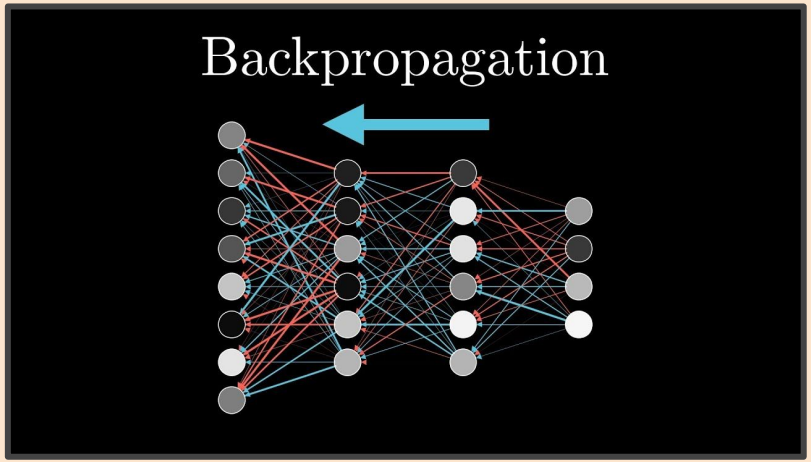

$$MSE = E(W) = \frac{1}{m}\sum_{i=1}^{m}[y^i - \hat{y}^i]^2$$

where $m$ is the number of samples in batch

# Backpropagation summary

- With backpropagation it does not matter how many layers you have. All you have to do is:

1. Back propagate the error
2. Calculate the gradient
3. Update the weights accordingly



# Good luck 😉


Uppsala universitet 1852–1916
Studenten, Staden och Samingen
UPPSALA UNIVERSITE