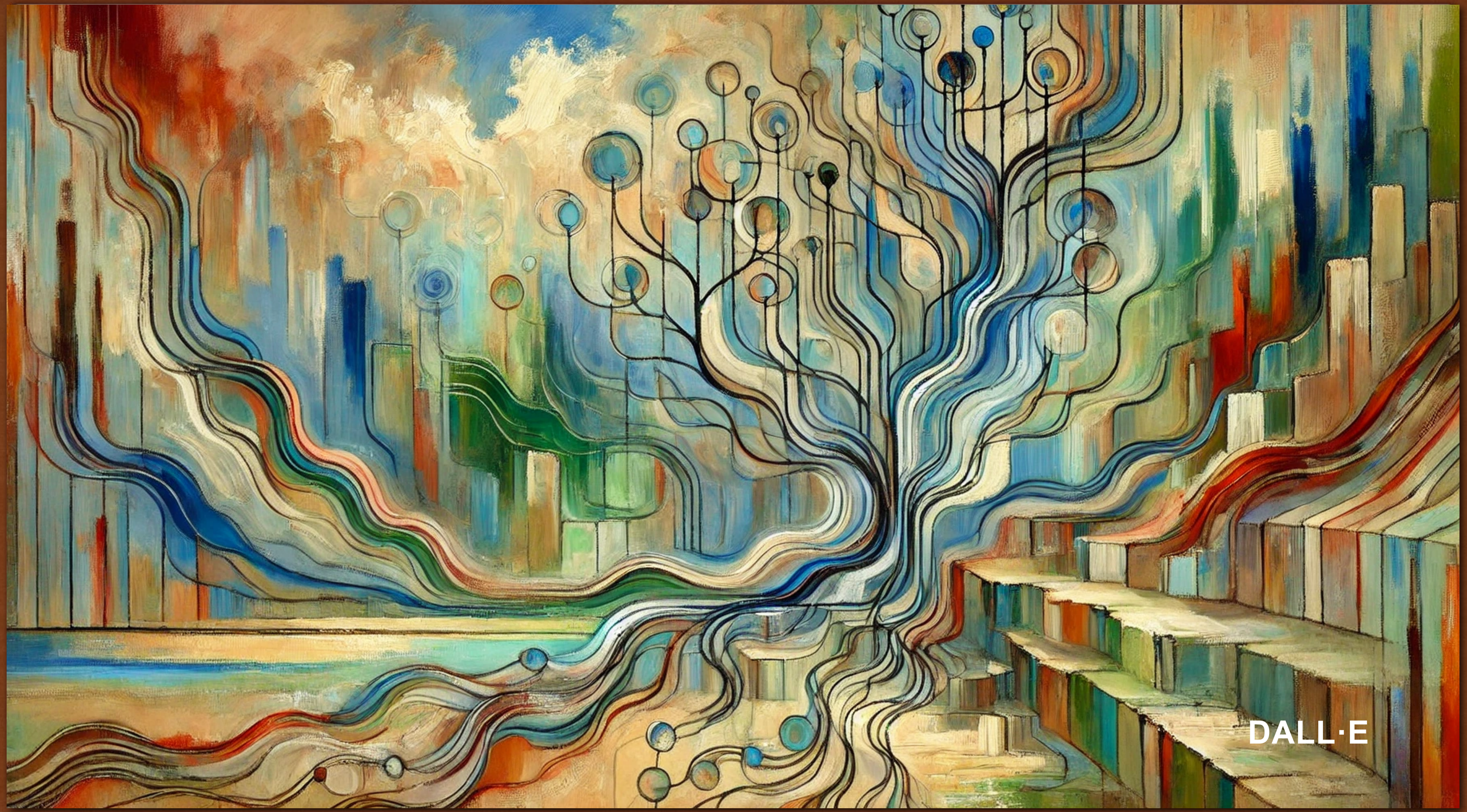
COM 3240

REINFORCEMENT LEARNING


DALL·E

FUTURE REWARDS

THE Q-LEARNING ALGORITHM

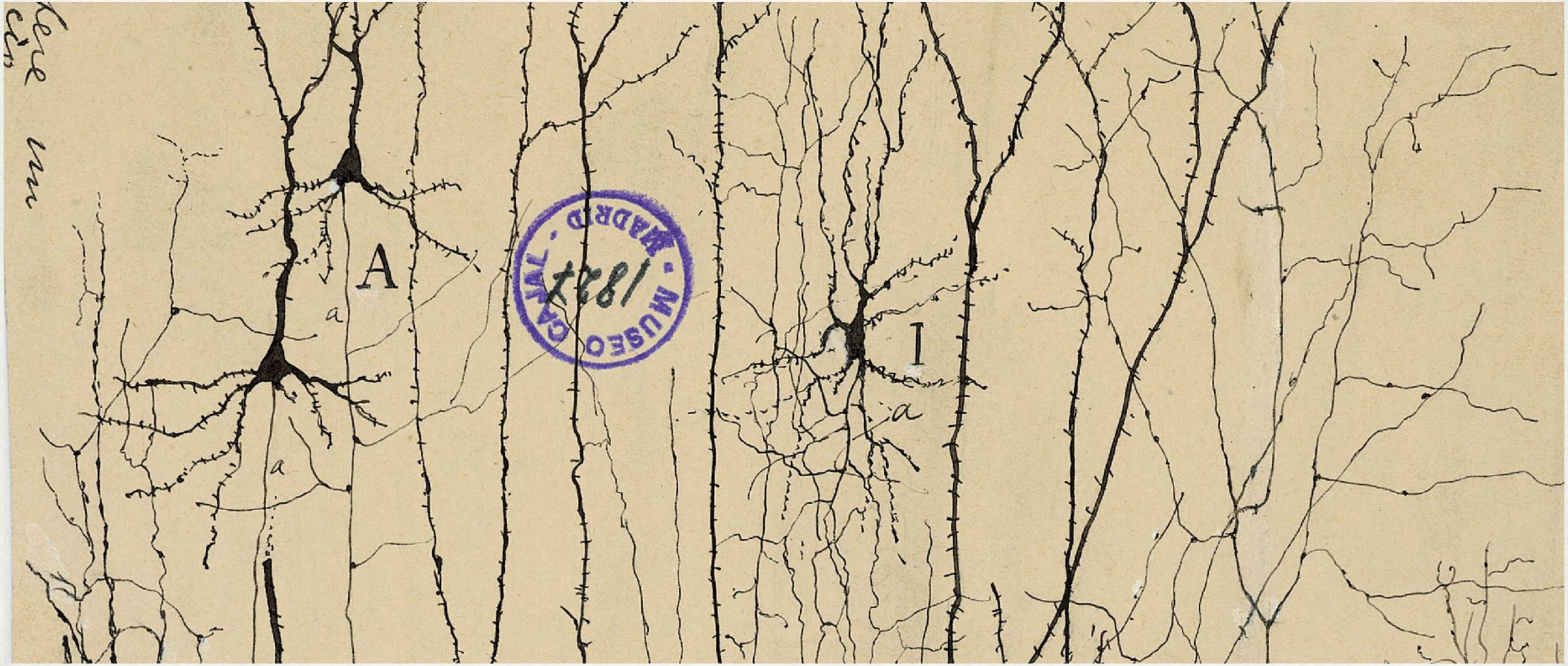
1. Initialise $Q(s, a)$ arbitrarily for all $s \in S$ and $a \in A(s)$.
2. Repeat (for each episode):
   a. Initialise s.
   b. Repeat (for each step of episode):
      i. Choose an action a from s using a policy derived from Q (e.g., ε-greedy).
      ii. Take action a, observe reward r and next state s'.
      iii. $Q(s, a) \leftarrow Q(s, a) + \eta * [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$.
      iv. $s \leftarrow s'$.
   c. until s is terminal.

Off-policy

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

Q-VALUES TABLE

$Q(s_1, a_1)$	$Q(s_2, a_1)$		
$Q(s_1, a_2)$			



Ramon y Cahal

1mm³ : 10,000 neurons 3km wires

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

SINGLE NEURON

$$\mathbf{y}^* = F^*(\mathbf{x})$$

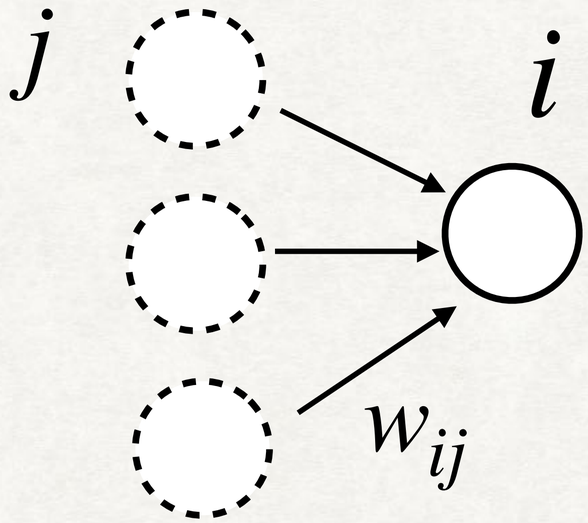
$$\mathbf{y} = F(\mathbf{x})$$

$$L = \frac{1}{2n} \sum_i \sum_{\mathbf{x}} \left(y_i^t(\mathbf{x}) - y_i(\mathbf{x}; \mathbf{W}) \right)^2$$

Samples from the input and output space of the function.

$$\mathbf{x}^{(1)}\ \mathbf{x}^{(2)}\ \mathbf{x}^{(3)} \ldots \mathbf{x}^{(n)}$$

$$\mathbf{y}^{(1)}\ \mathbf{y}^{(2)}\ \mathbf{y}^{(3)} \ldots \mathbf{y}^{(n)}$$



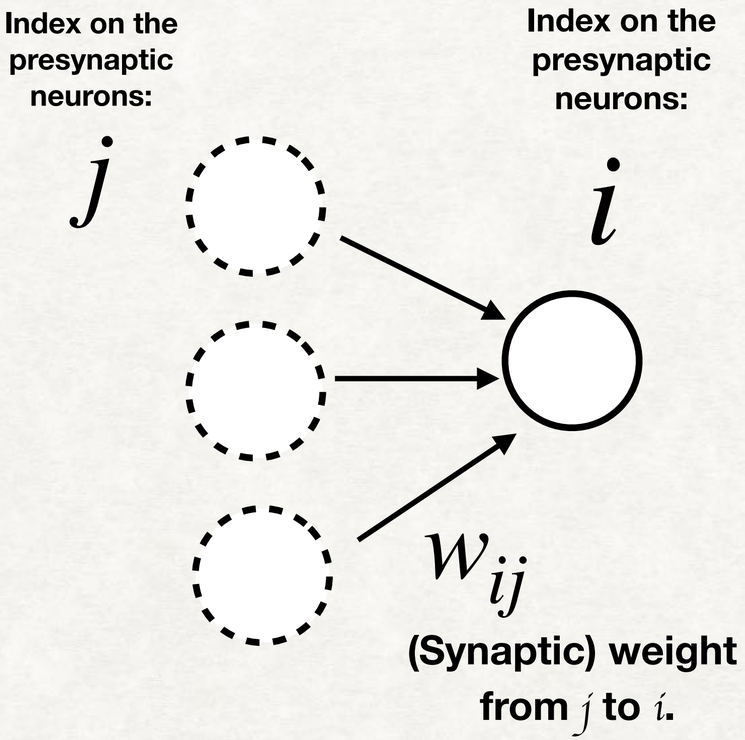
Find parameters (weights) that minimise the Loss function.

$$\Delta \mathbf{W}_{lk} \propto - \frac{\partial E}{\partial \mathbf{W}_{lk}}$$

$$\Delta \mathbf{W} \propto - \nabla_{\mathbf{W}} E$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

SINGLE NEURON



$$y = ax + b$$

$$h_i = \sum_j w_{ij} x_j + b_i$$

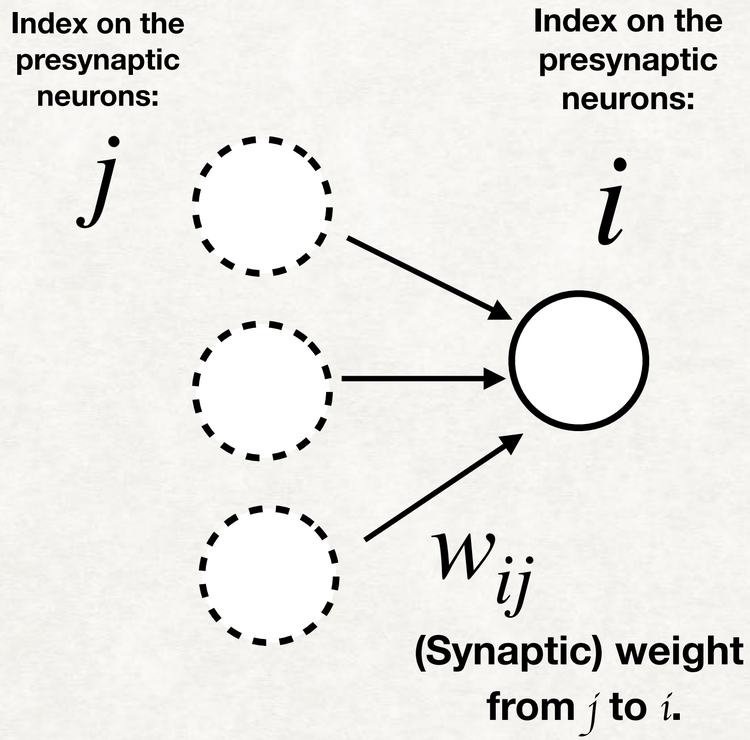
Bias

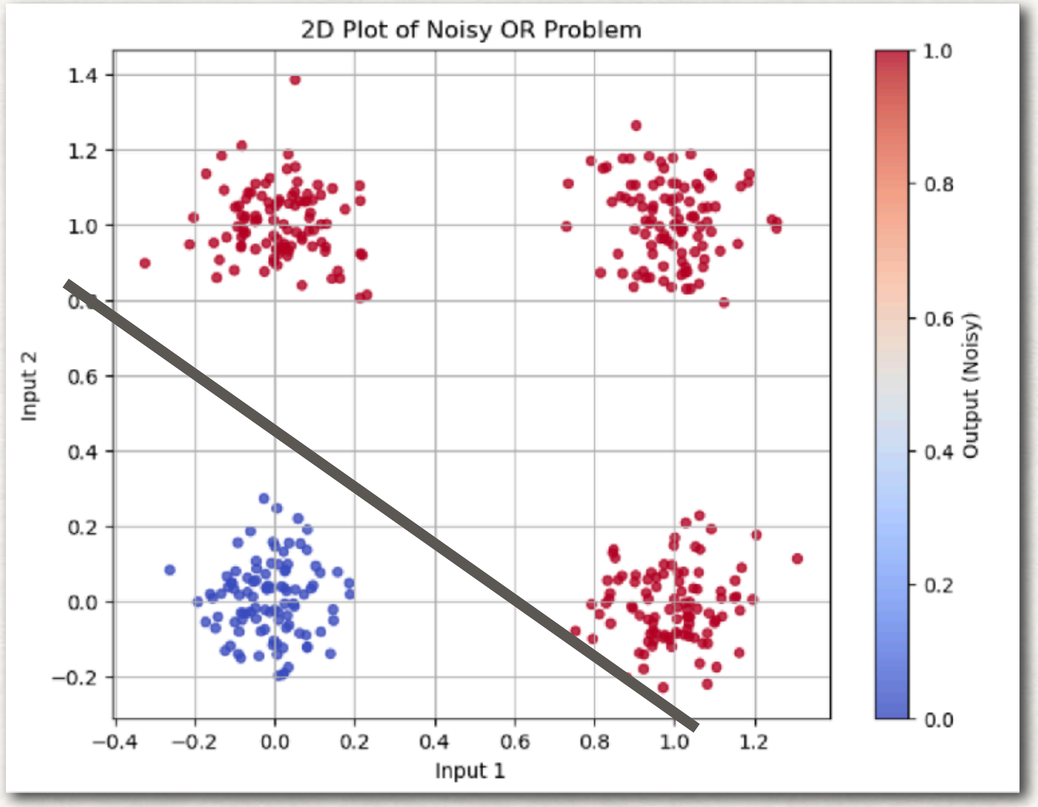
$$x_i = f(h_i)$$

Activation function *f*, sigmoidal or ReLU

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

NON-LINEARLY SEPARABLE PROBLEMS

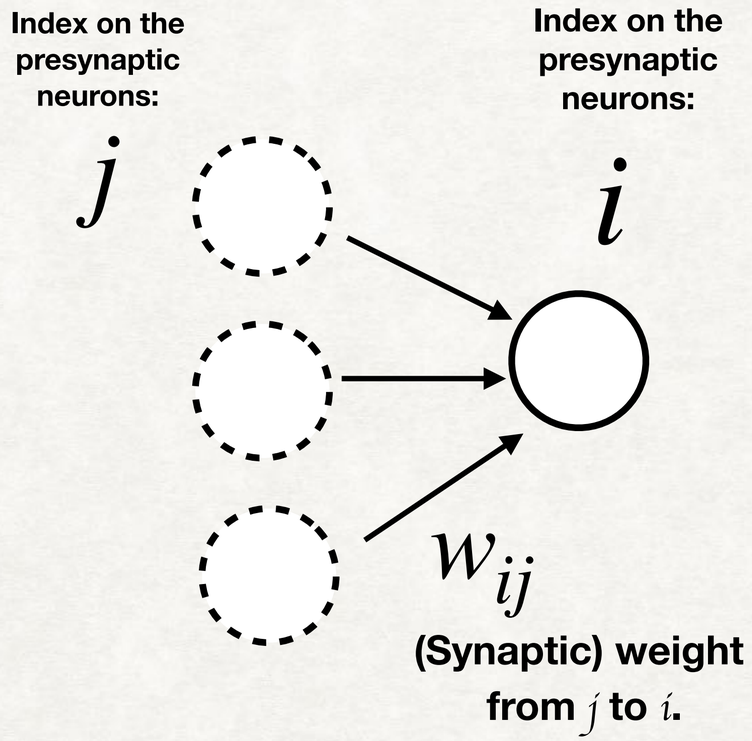


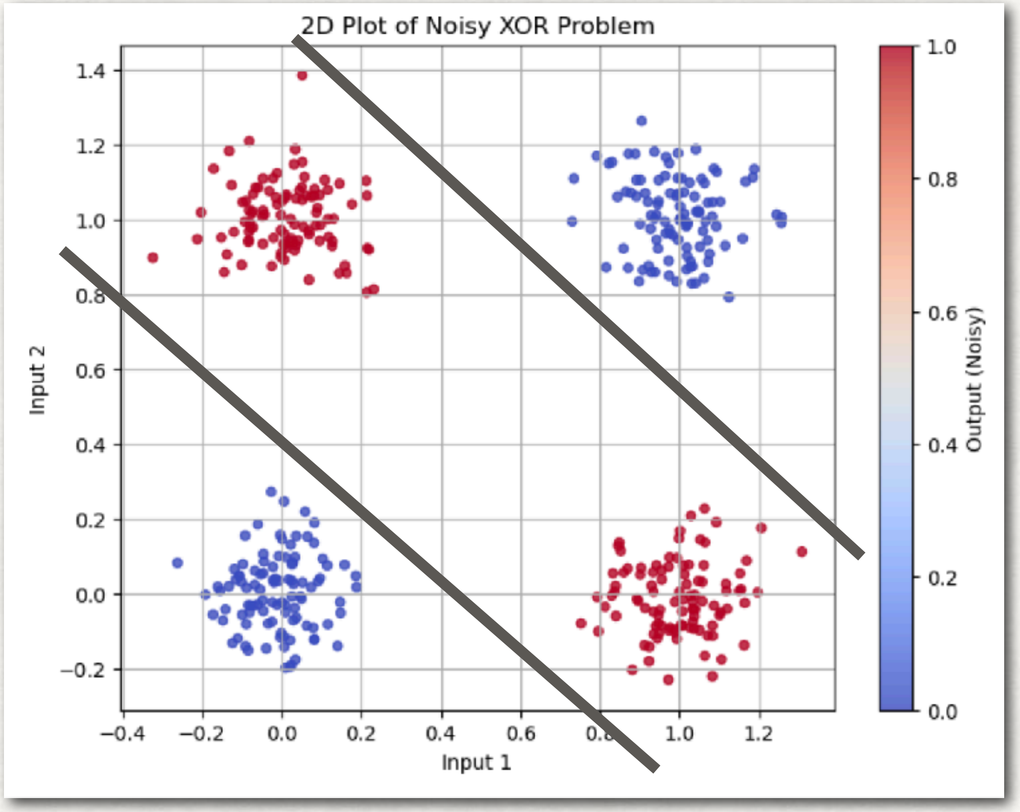


A single layer can solve very easy problems.

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

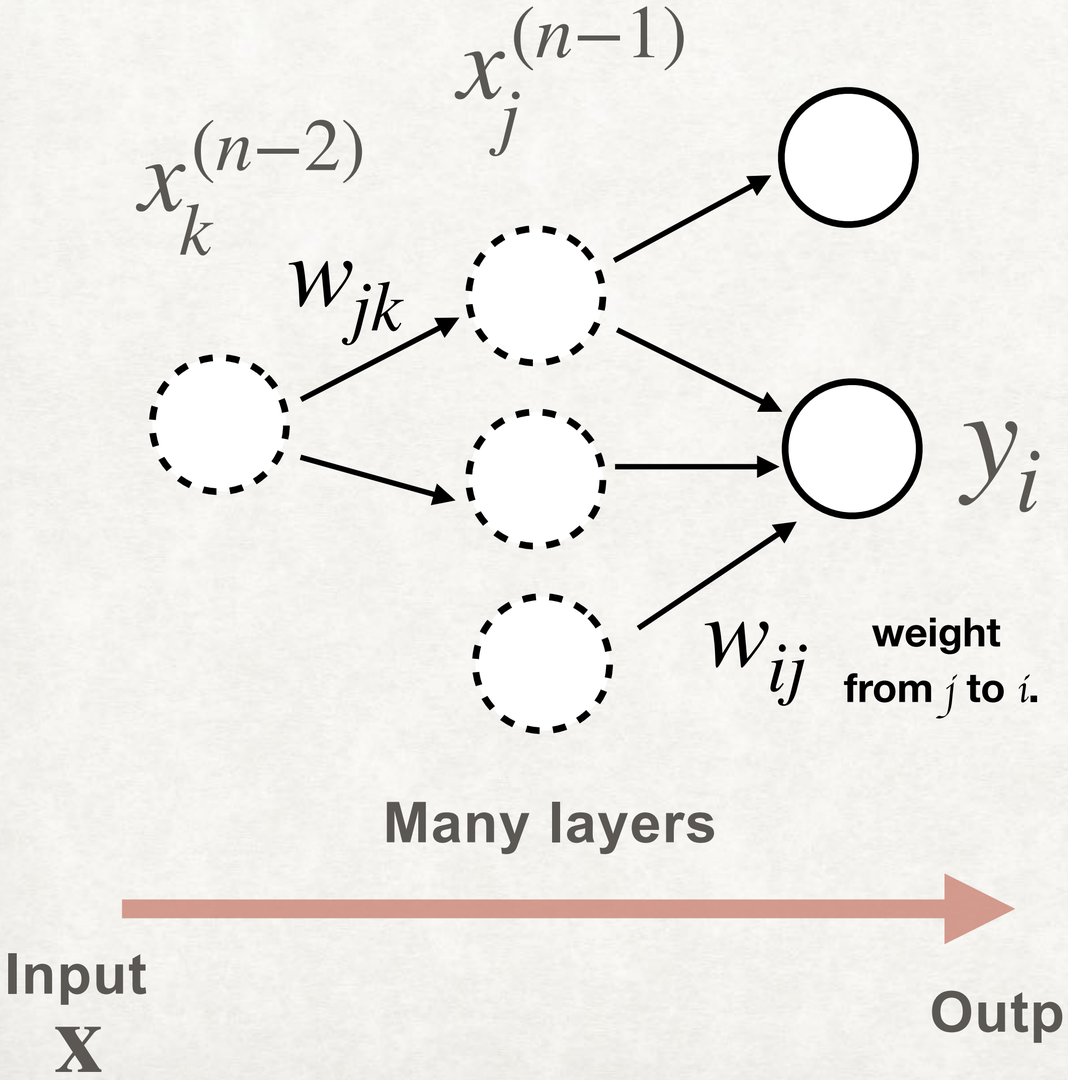
NON-LINEARLY SEPARABLE PROBLEMS





A single layer of neurons cannot learn such problems

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS



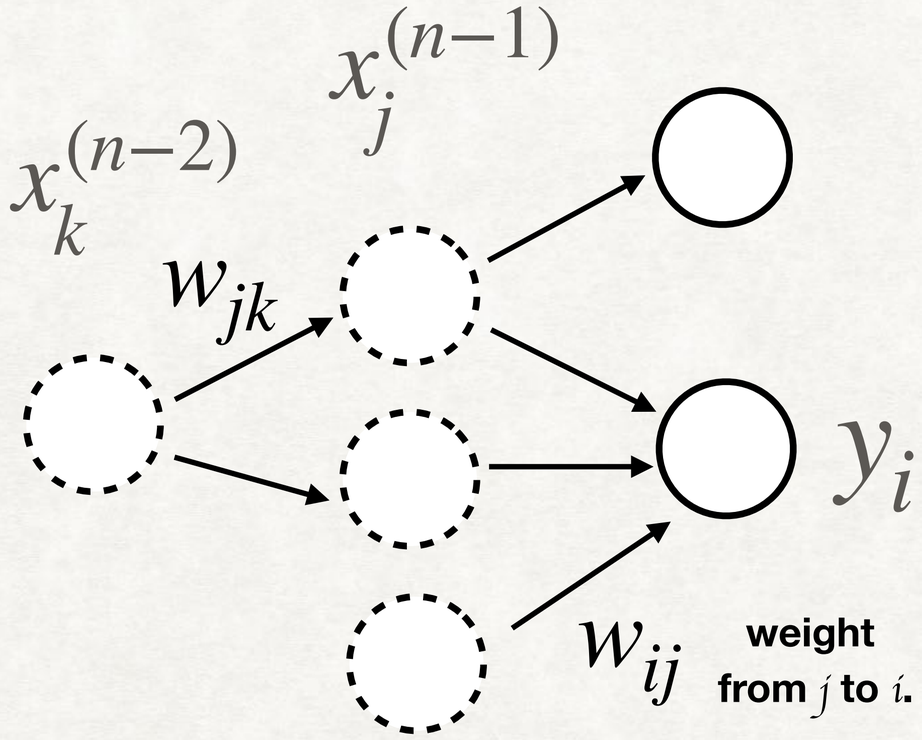
Forward Pass

Calculates output $\mathbf{y}$

$$h_i = \sum_j w_{ij} y_j + b_i$$

$$y_i = f(h_i)$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS



Forward Pass

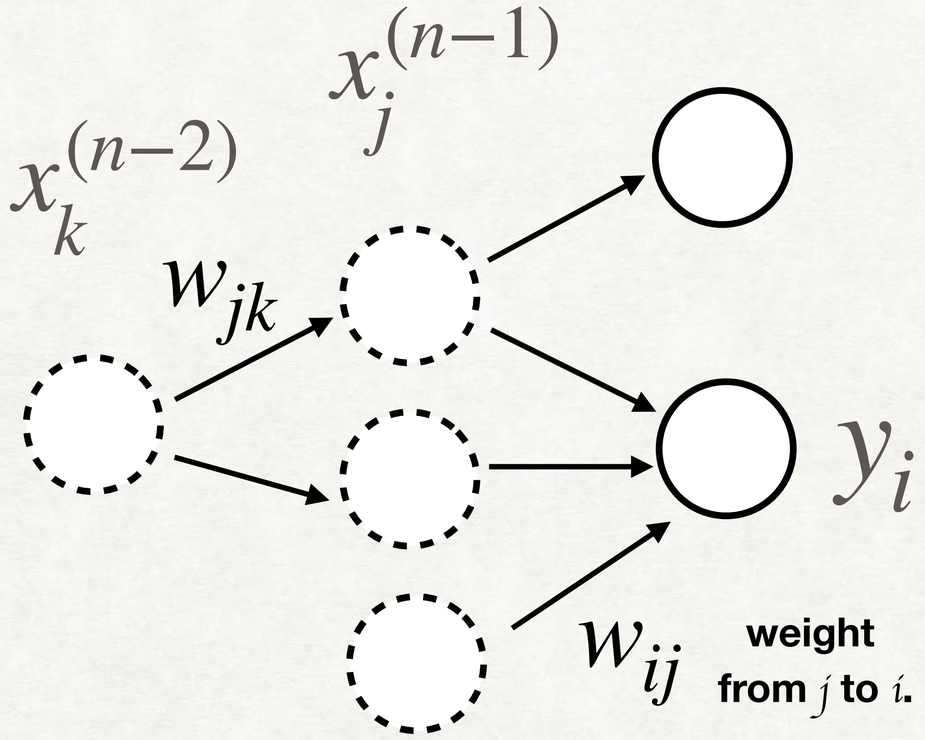
Calculates output $\mathbf{y}$

Calculates error

SSE

$$L = \frac{1}{2} \sum_{\mathbf{x}} \sum_{i} \left(y_i^*(\mathbf{x}) - y_i(\mathbf{x}; \mathbf{W}) \right)^2$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS



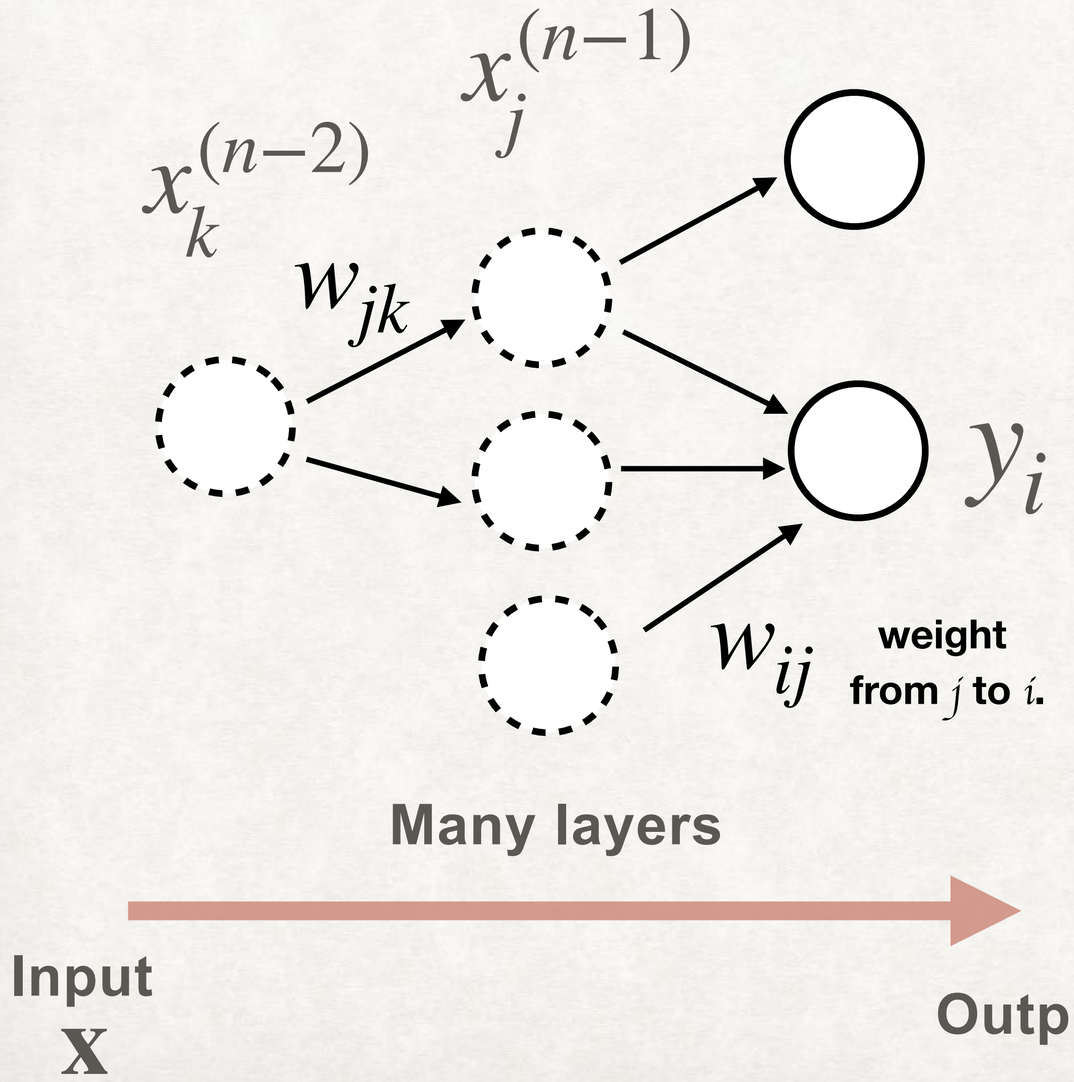
SSE

$$L = \sum_{\mathbf{x}} L_o$$

$$L_o = \frac{1}{2} \sum_i \left(y_i^*(\mathbf{x}) - y_i(\mathbf{x}; \mathbf{W}) \right)^2$$

We suppress the sum to operate online.

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS



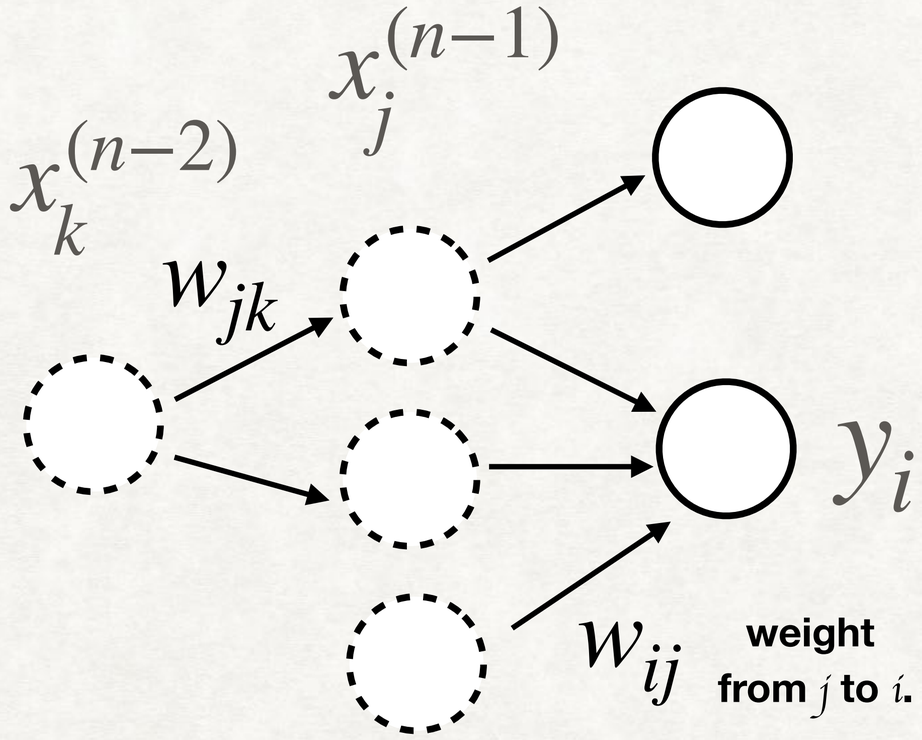
Backward pass

Updates weights and biases to decrease the error.

To calculate weight updates take the gradient on a Loss function.

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$L = \frac{1}{2}\sum_{\mathbf{x}}\sum_{i}\left(y_i^*(\mathbf{x}) - y_i(\mathbf{x};\mathbf{W})\right)^2 = \sum_{\mathbf{x}} L_o$$

$$\Delta w_{ij} = -\eta\frac{\partial L}{\partial w_{ij}} = -\eta\sum_{\mathbf{x}}\frac{\partial L_o}{\partial w_{ij}}$$

$$\Delta w_{jk} = -\eta\frac{\partial L}{\partial w_{jk}} = -\eta\sum_{\mathbf{x}}\frac{\partial L_o}{\partial w_{jk}}$$

Many layers

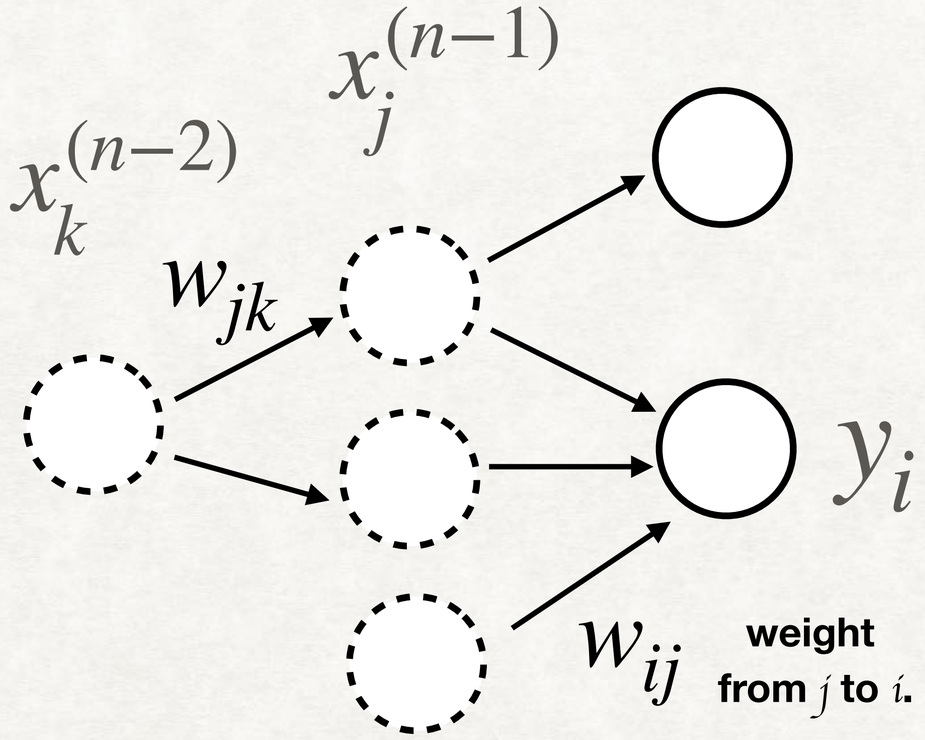
Input
$\mathbf{x}$

Output

Similarly for biases

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$L = \frac{1}{2}\sum_{\mathbf{x}}\sum_{i}\left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

$$\Delta w_{ij} = -\eta\frac{\partial L}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\eta\sum_{\mathbf{x}}\frac{\partial L_o}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}}$$

Many layers

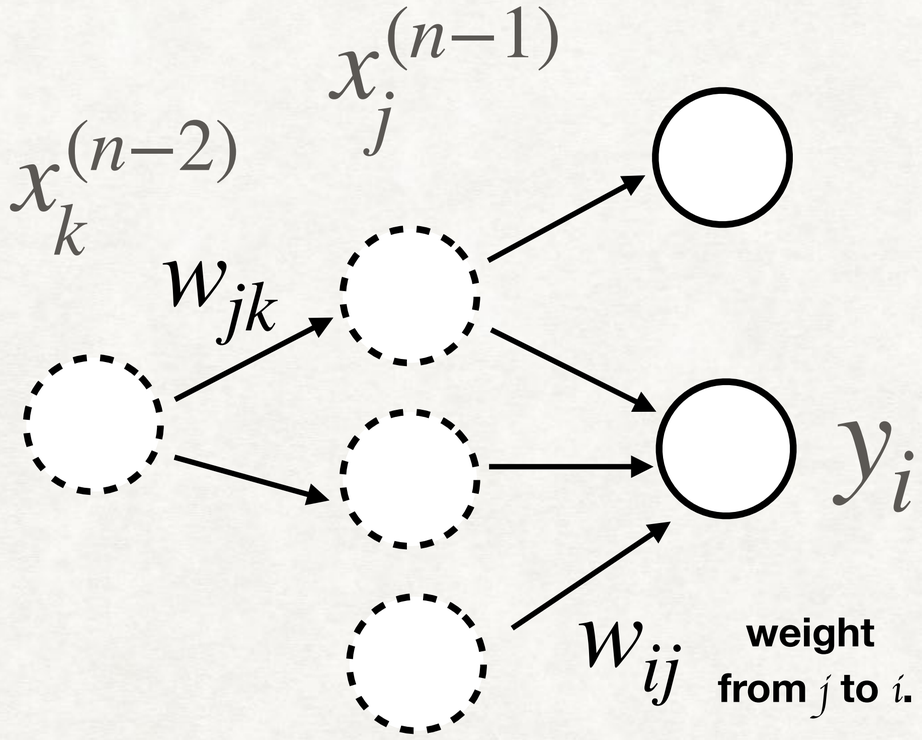
Input
x

Output

We have done it previously (delta rule).

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



Many layers

Input
x

Output

$$L = \frac{1}{2}\sum_{\mathbf{x}}\sum_{i}\left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

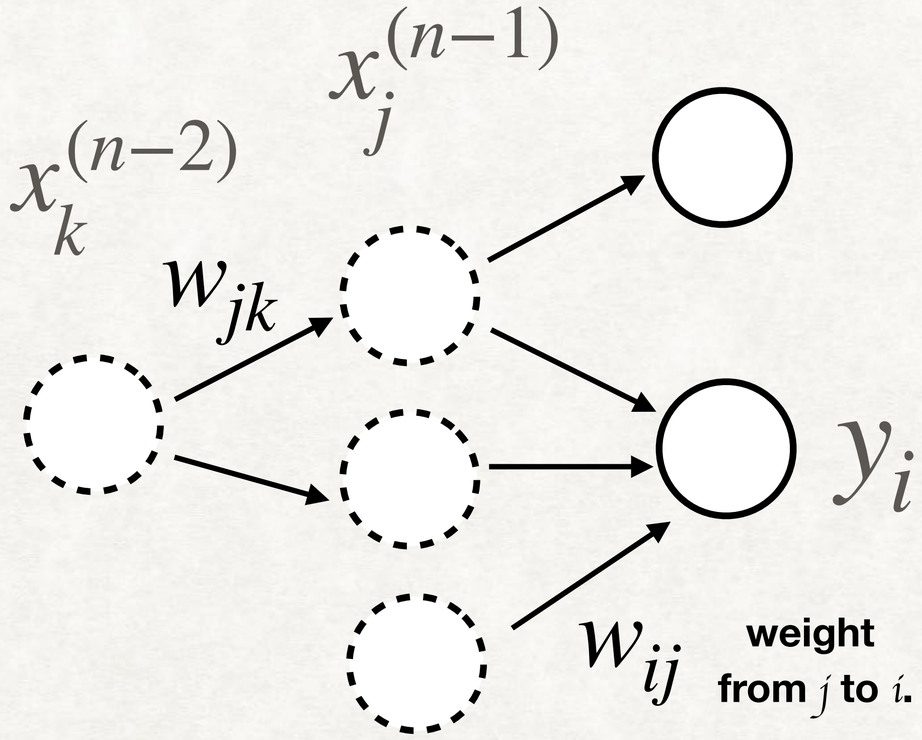
$$\Delta w_{ij} = -\eta\frac{\partial L}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\eta\sum_{\mathbf{x}}\frac{\partial L_o}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}}$$

$$\Delta w_{ij} = \eta\sum_{\mathbf{x}}\left(y_i^* - y_i\right)f_i'^{(n)}x_j^{(n-1)}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



Many layers

Input
$\mathbf{x}$

Output

$$L = \frac{1}{2}\sum_{\mathbf{x}}\sum_{i}\left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

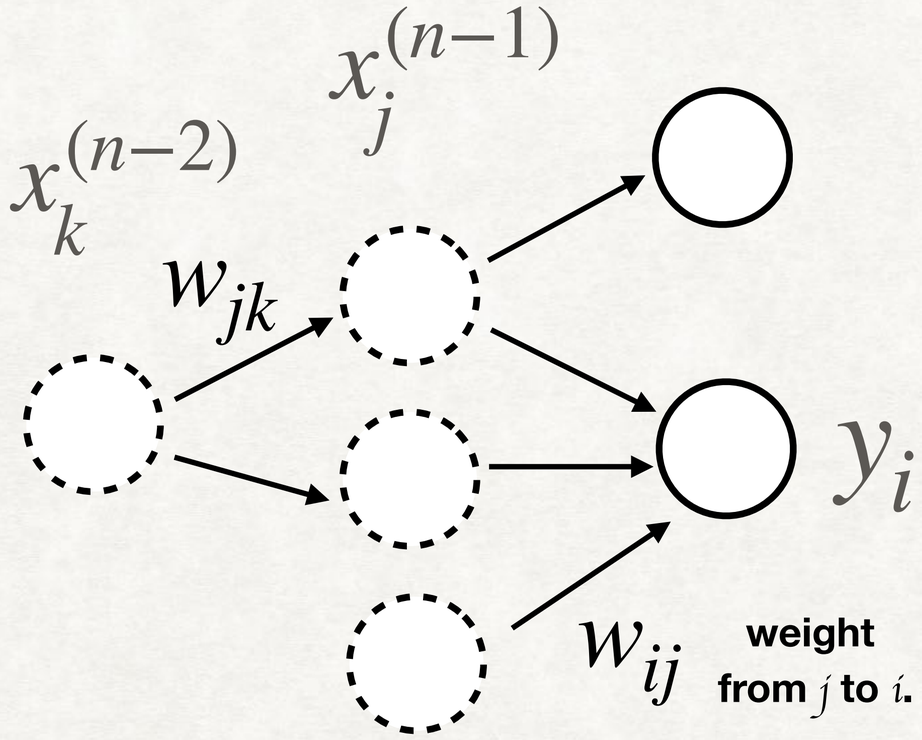
$$\Delta w_{jk} = -\eta\frac{\partial L}{\partial w_{jk}} = -\eta\sum_{\mathbf{x}}\frac{\partial L_o}{\partial w_{jk}}$$

$$\Delta w_{jk} = -\eta\sum_{\mathbf{x}}\frac{\partial L_o}{\partial x_j^{(n-1)}}\frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

It is safe to write like this because the weight affects just this specific neuron directly.

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



Many layers

Input
x

Output

$$L = \frac{1}{2}\sum_{\mathbf{x}}\sum_{i}\left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

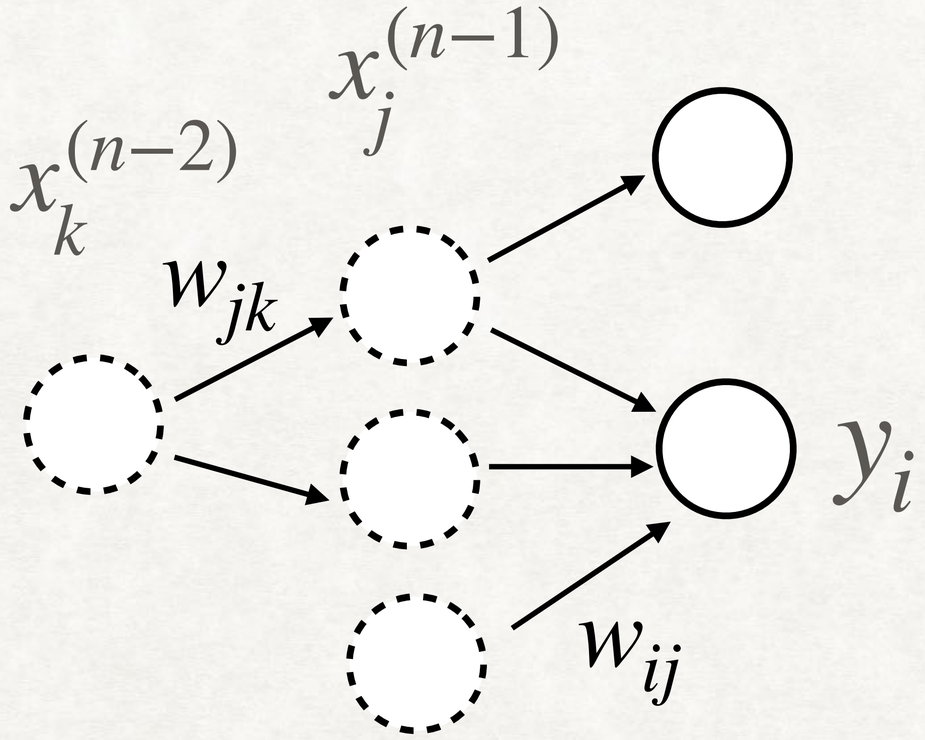
$$\Delta w_{jk} = -\eta \sum_{\mathbf{x}} \frac{\partial L_o}{\partial x_j^{(n-1)}} \frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

We calculate the two terms separately:

$$\frac{\partial L_o}{\partial x_j^{(n-1)}} \qquad \frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



Many layers

Input
X

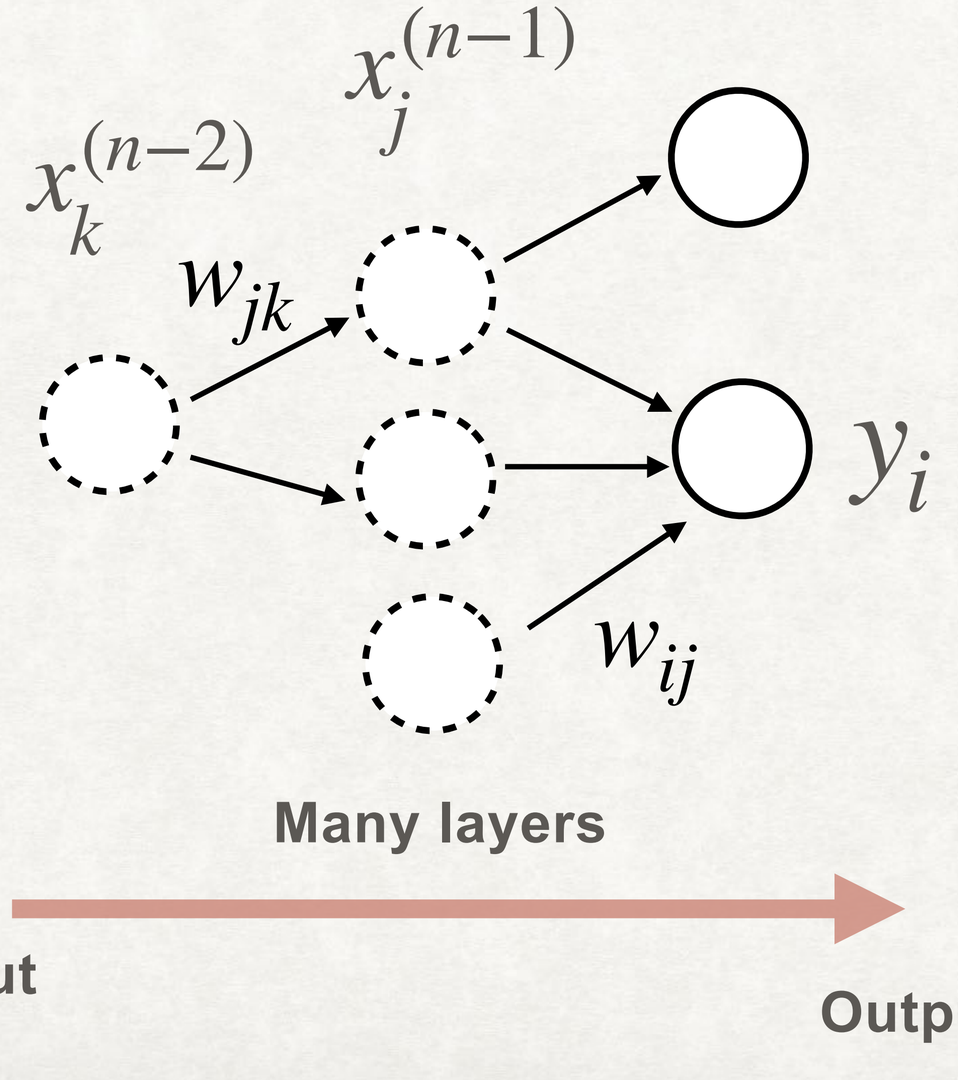
Output

$$L_o = \frac{1}{2}\sum_i \left(y_i^* - y_i\right)^2$$

$$\frac{\partial L_o}{\partial x_j^{(n-1)}} = \frac{1}{2}\sum_i \frac{\partial \left(y_i^* - y_i\right)^2}{\partial x_j^{(n-1)}}$$

$$= \frac{1}{2}\sum_i \frac{\partial \left(y_i^* - y_i\right)^2}{\partial \left(y_i^* - y_i\right)} \frac{\partial \left(y_i^* - y_i\right)}{\partial x_j^{(n-1)}}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

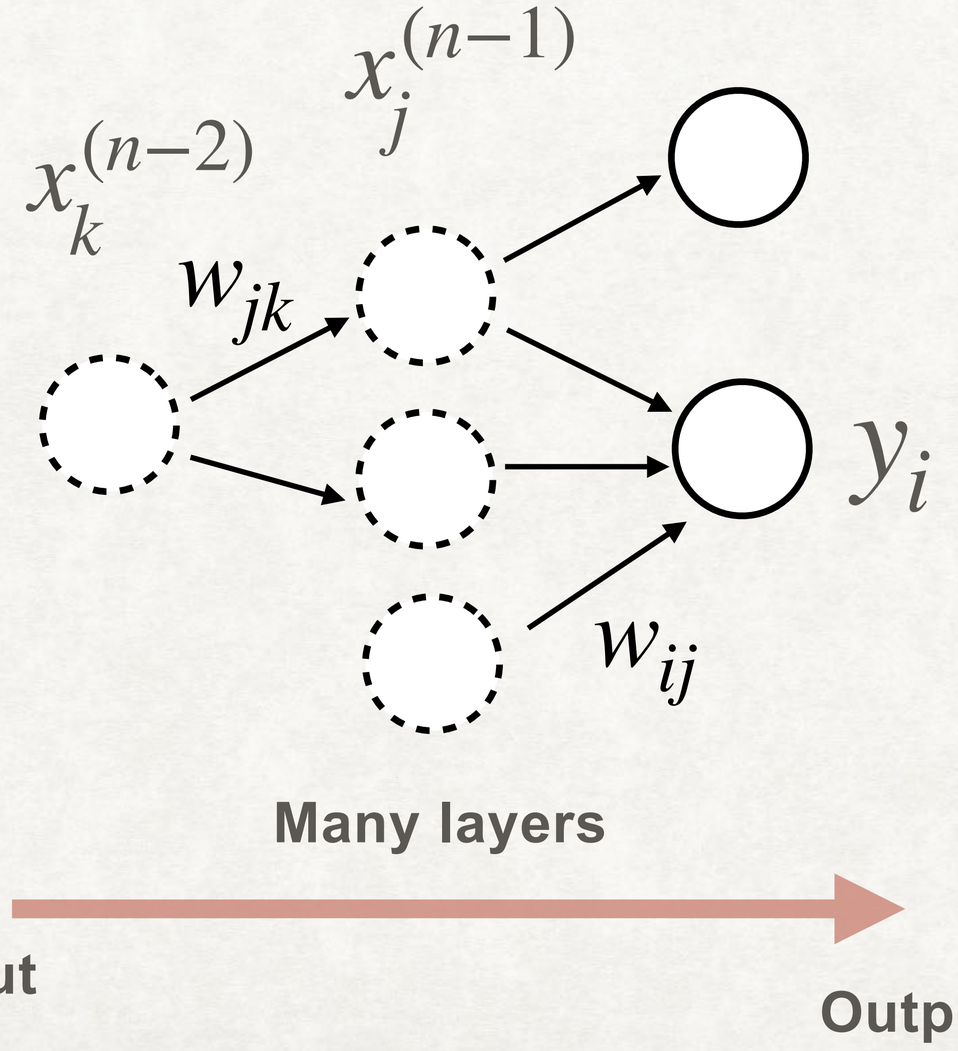


$$L_o = \frac{1}{2}\sum_i \left(y_i^* - y_i\right)^2$$

$$\frac{\partial L_o}{\partial x_j^{(n-1)}} = \frac{1}{2}\sum_i \frac{\partial \left(y_i^* - y_i\right)^2}{\partial x_j^{(n-1)}}$$

$$= \sum_i \left(y_i^* - y_i\right) \frac{\partial \left(y_i^* - y_i\right)}{\partial x_j^{(n-1)}}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

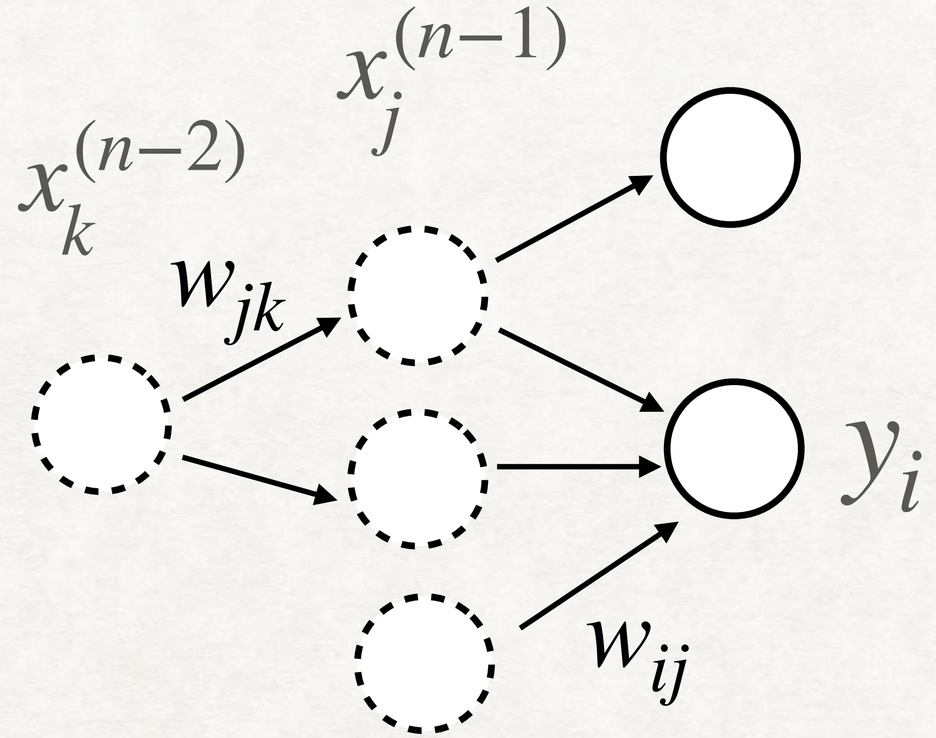


$$L_o = \frac{1}{2}\sum_i \left(y_i^* - y_i\right)^2$$

$$\frac{\partial L_o}{\partial x_j^{(n-1)}} = \frac{1}{2}\sum_i \frac{\partial \left(y_i^* - y_i\right)^2}{\partial x_j^{(n-1)}}$$

$$= -\sum_i \left(y_i^* - y_i\right)\frac{\partial y_i}{\partial x_j^{(n-1)}}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS



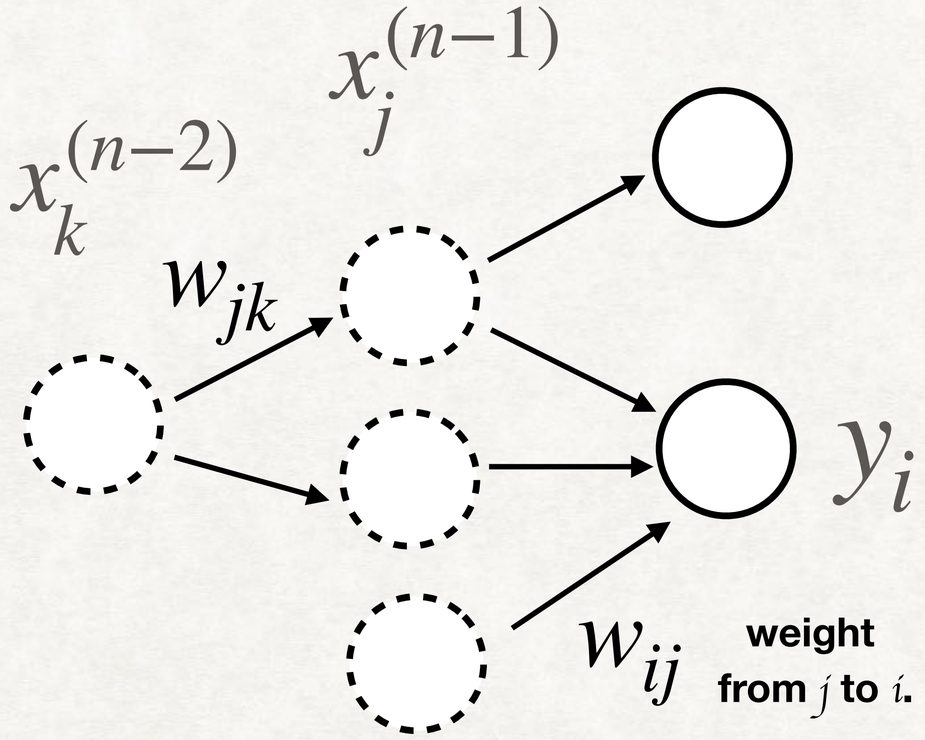
$$\frac{\partial y_i}{\partial x_j^{(n-1)}} \qquad y_i = f\left(\sum_j w_{ij} x_j^{(n-1)}\right)$$

$$\frac{\partial y_i}{\partial x_j^{(n-1)}} = \frac{\partial f\left(\sum_j w_{ij} x_j^{(n-1)}\right)}{\partial x_j^{(n-1)}} = f_i'^{(n)} w_{ij}$$

$$\frac{\partial L}{\partial x_j^{(n-1)}} = -\sum_i \sum_{\mathbf{x}} \left(y_i^* - y_i\right) f_i'^{(n)} w_{ij}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$L = \frac{1}{2}\sum_{\mathbf{x}}\sum_{i}\left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

$$\Delta w_{jk} = -\eta \sum_{x} \frac{\partial L_o}{\partial x_j^{(n-1)}} \frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

We calculate the two terms separately:

$$\frac{\partial L}{\partial x_j^{(n-1)}}$$

$$\frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

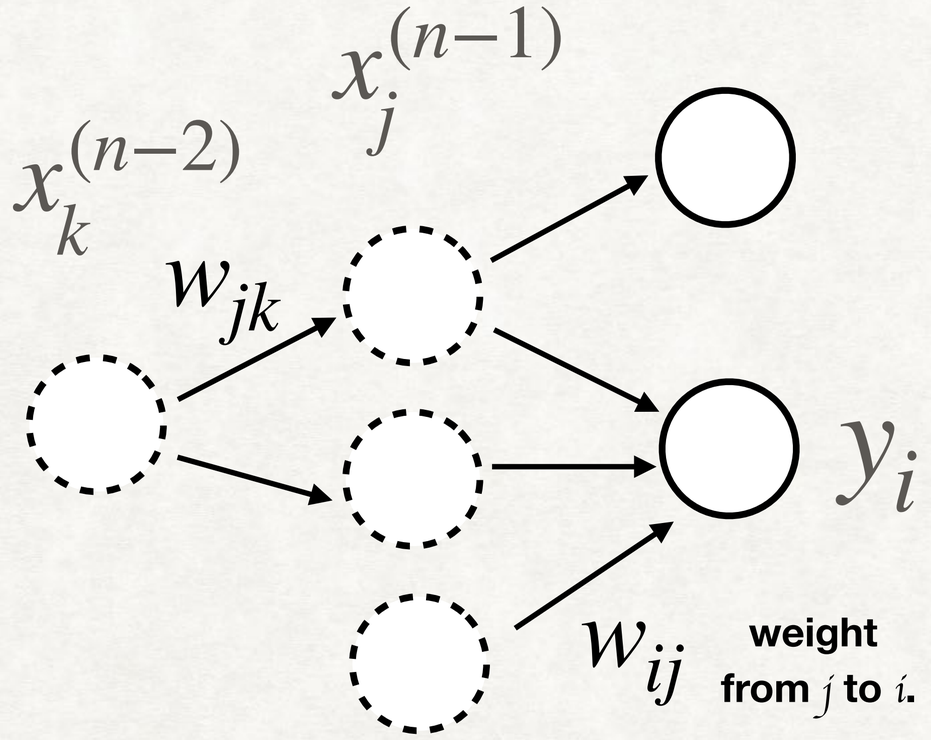
Many layers

Input
x

Output

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$\frac{\partial x_j^{(n-1)}}{\partial w_{jk}} \qquad x_j^{(n-1)} = f\left(\sum_j w_{jk} x_k^{(n-2)}\right)$$

$$\frac{\partial x_j^{(n-1)}}{\partial w_{jk}} = \frac{\partial f\left(\sum_k w_{jk} x_k^{(n-2)}\right)}{\partial w_{jk}} = f_j'^{(n-1)} x_k^{(n-2)}$$

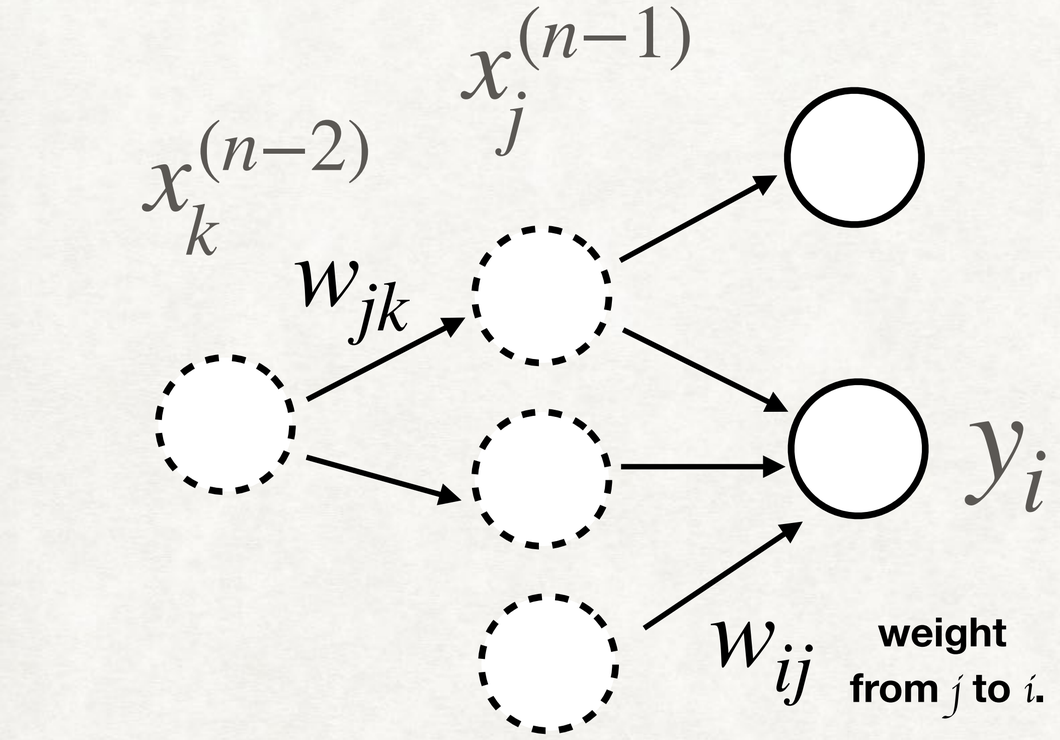
Many layers

Let us combine the results.

Input
X

Output

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS



$$L = \frac{1}{2}\sum_i \sum_{\mathbf{x}} \left(y_i^* - y_i\right)^2$$

$$\Delta w_{jk} = -\eta \frac{\partial L}{\partial x_j^{(n-1)}} \frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

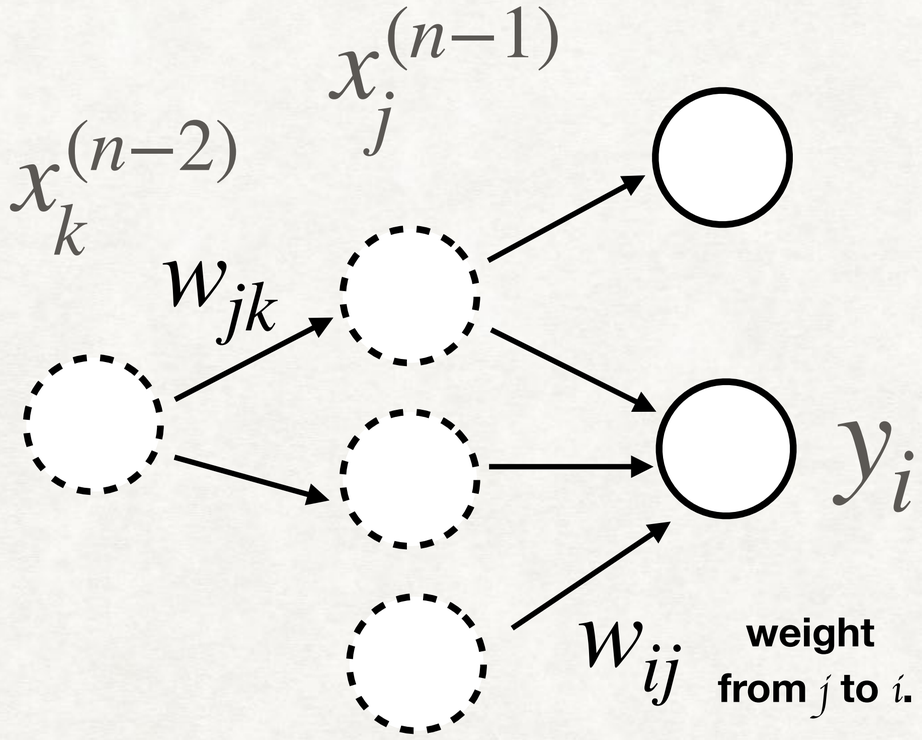
We will calculate the two terms separately:

$$\frac{\partial L}{\partial x_j^{(n-1)}}$$

$$\frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$L = \frac{1}{2}\sum_i \sum_{\mathbf{x}} \left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

$$\Delta w_{jk} = -\eta \sum_{\mathbf{x}} \frac{\partial L_o}{\partial x_j^{(n-1)}} \frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

$$\Delta w_{jk} = \eta \sum_{\mathbf{x}} \sum_i \Bigg(\underbrace{\left(y_i^* - y_i\right) f_i'^{(n)} w_{ij}}_{\delta_i^{(n)}} \Bigg) f_j'^{(n-1)} x_k^{(n-2)}$$

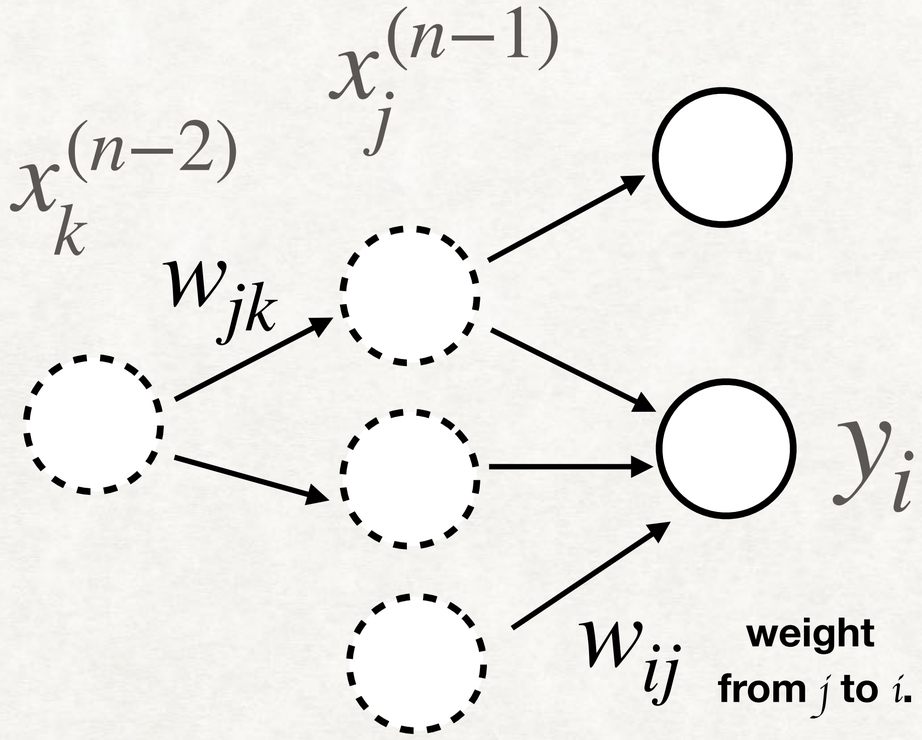
Many layers

Input
$\mathbf{x}$

Output

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$L = \frac{1}{2}\sum_i \sum_{\mathbf{x}} \left(y_i^* - y_i\right)^2 = \sum_{\mathbf{x}} L_o$$

$$\Delta w_{jk} = -\eta \sum_{\mathbf{x}} \frac{\partial L_o}{\partial x_j^{(n-1)}} \frac{\partial x_j^{(n-1)}}{\partial w_{jk}}$$

$$\Delta w_{jk} = \eta \sum_{\mathbf{x}} \frac{\sum_i \delta_i^{(n)} w_{ij} f_j'^{(n-1)} x_k^{(n-2)}}{\delta_j^{(n-1)}}$$

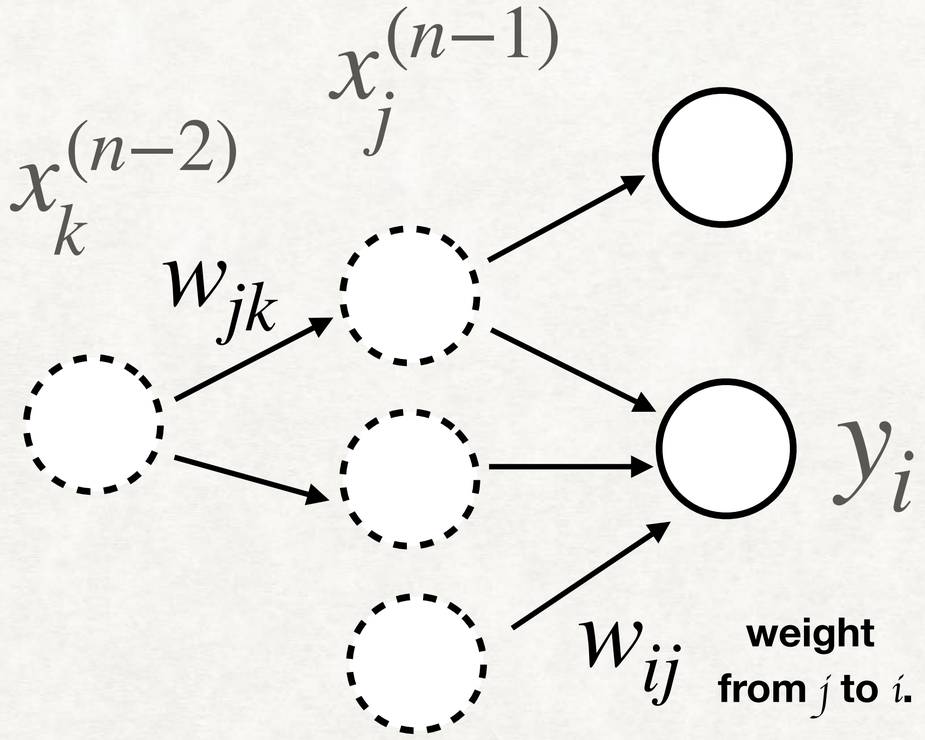
Many layers

Input
$\mathbf{x}$

Output

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

HIDDEN LAYER



$$L = \frac{1}{2} \sum_{\mathbf{x}} \sum_{i} \left(y_i^* - y_i \right)^2$$

$$\delta_i^{(n)} = \left(y_i^* - y_i \right) f_i'^{(n)}$$

$$\delta_j^{(n-1)} = \sum_i \delta_i^{(n)} w_{ij} f_j'^{(n-1)}$$

$$\Delta w_{ij} = \eta \sum_{\mathbf{x}} \delta_i^{(n)} x_j^{(n-1)}$$

$$\Delta w_{jk} = \eta \sum_{\mathbf{x}} \delta_j^{(n-1)} x_k^{(n-2)}$$

Many layers

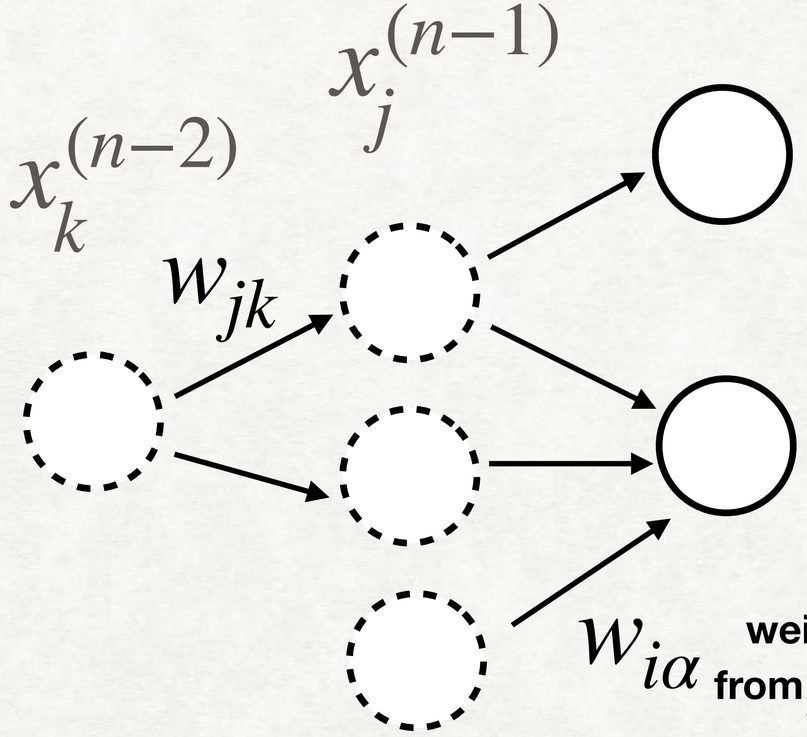
Input
$\mathbf{x}$

Output

For arbitrary number of layers.

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

REPLACING THE Q-VALUE TABLE



$$Q(s, \alpha)$$

One output neuron per action

$$L = \frac{1}{2}\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2$$

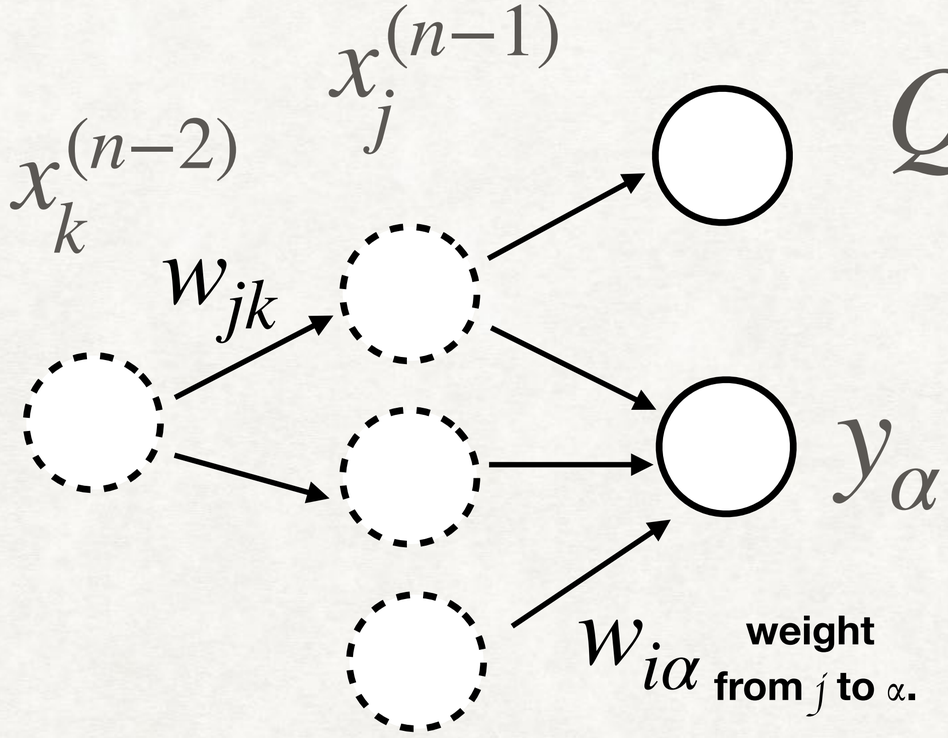
Online setting

Input=State *s*

$$y_\alpha^*(\mathbf{s}) = r + \gamma \max_{a'} Q(\mathbf{s}', a')$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

REPLACING THE Q-VALUE TABLE



$$Q(s, \alpha)$$

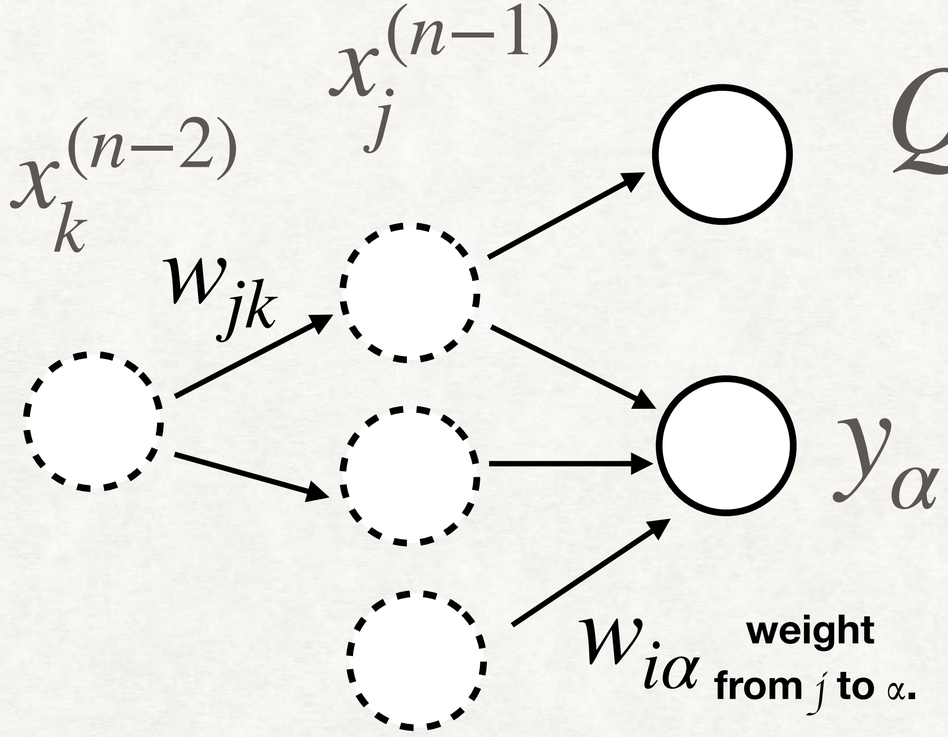
The reward collected is only relevant to the action taken!

Input=State *s*

$$y_\alpha^*(\mathbf{s}) = r + \gamma \max_{a'} Q(\mathbf{s}', a')$$

ARTIFICIAL NEURAL NETWORKS AS FUNCTION APPROXIMATIONS

REPLACING THE Q-VALUE TABLE



$Q(s, \alpha)$

Trick: I construct my target into following way.

For the actions not taken the corresponding output neuron has target its true output. No changes!

Input=State *s*

For the action taken the target is set as:

$$y_\alpha^*(\mathbf{s}) = r + \gamma \max_{a'} Q(\mathbf{s}', a')$$


DALL·E

THANK YOU!