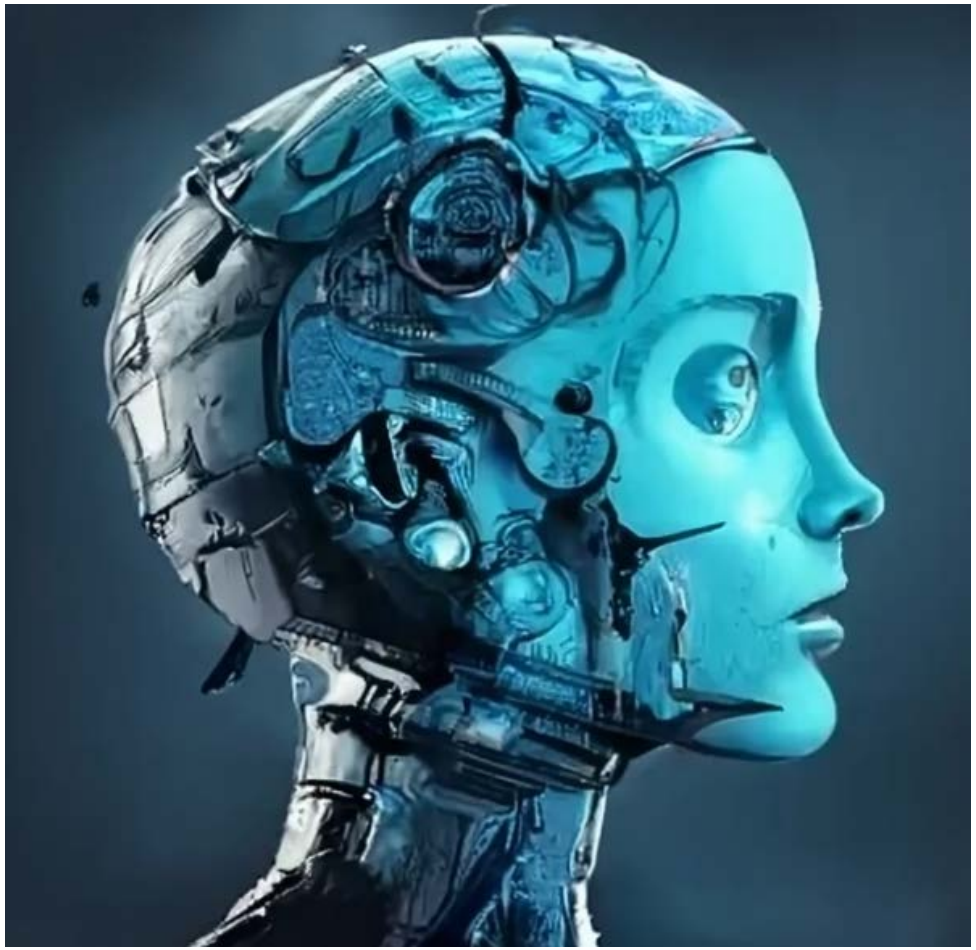
Machine Intelligence

Introduction to Artificial Neural Networks

Dr. Cory Merkel



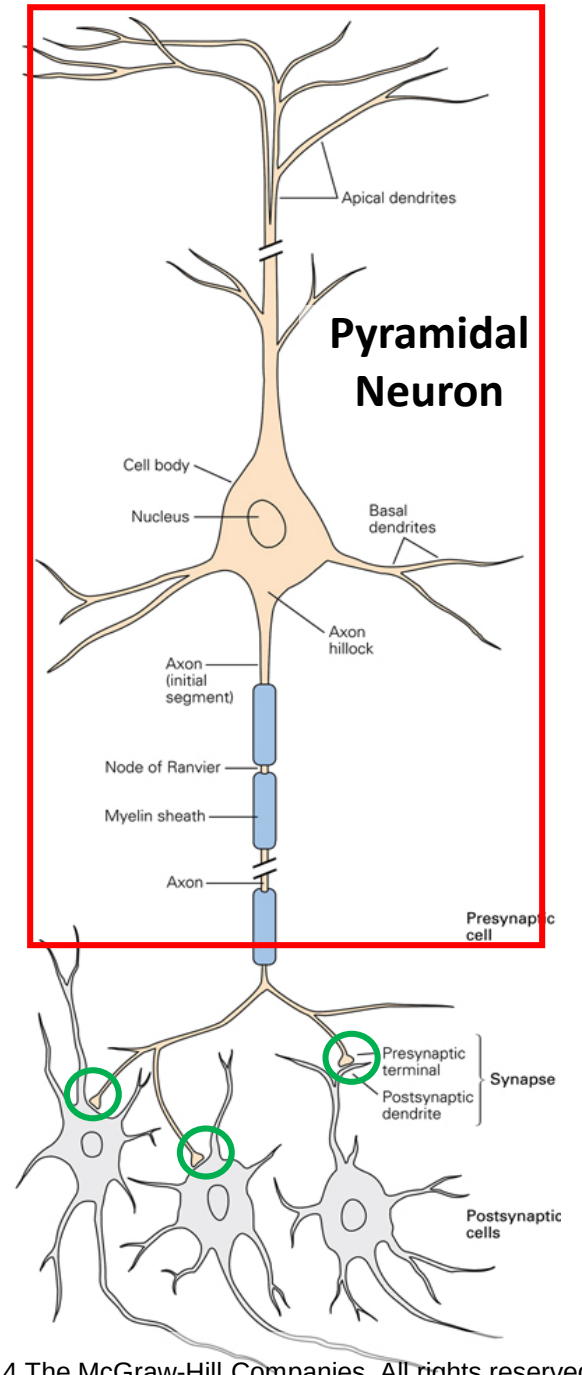
From Brains to Artificial Neural Networks

- Abstract models of biological *neuronal networks* (networks of real neurons in the brain)

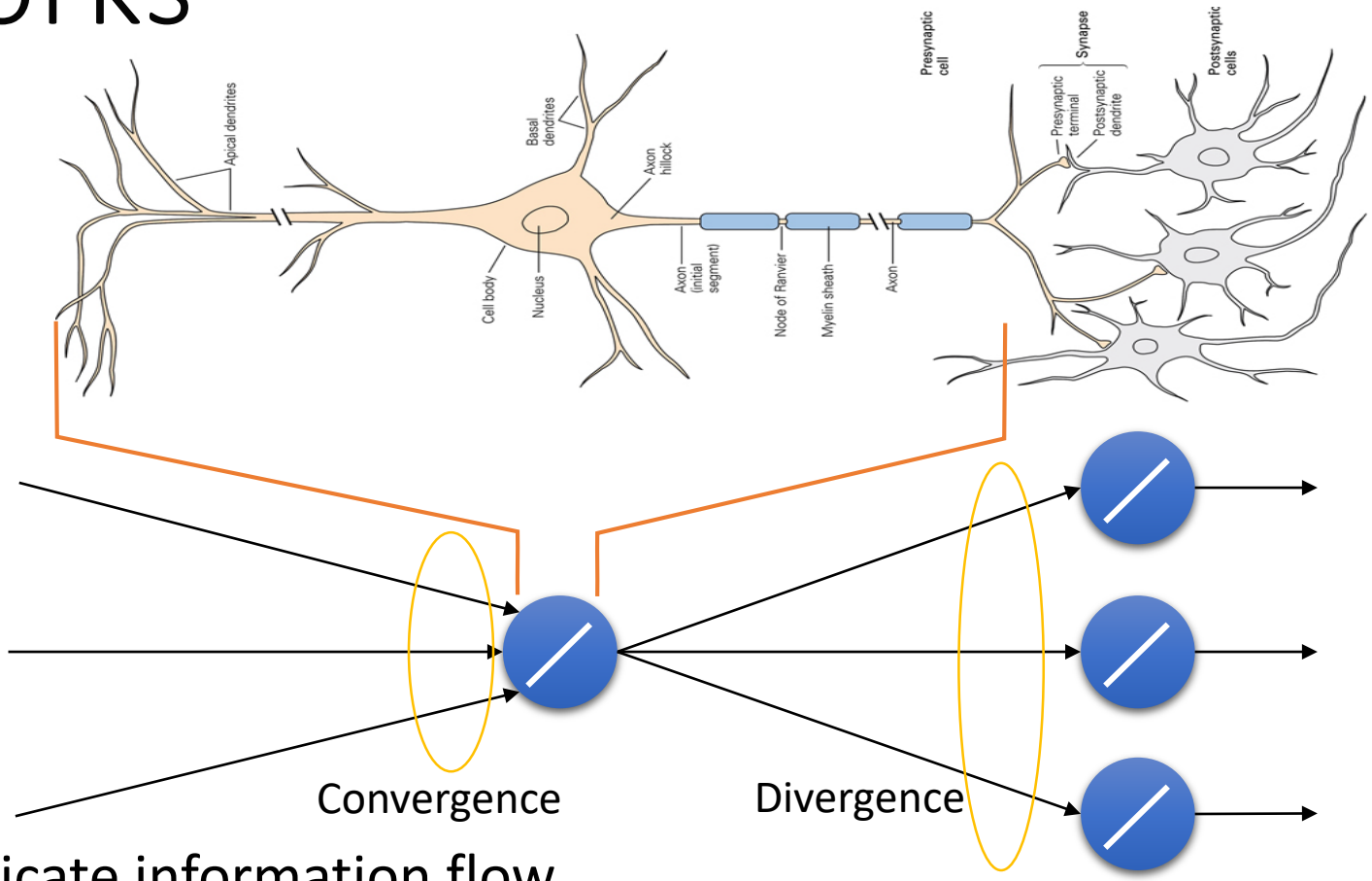
- 3 primary parts of an ANN:
  - Activation units (neurons)
  - Weights (synapses)
  - Training/learning algorithms

Symbols:

N/A



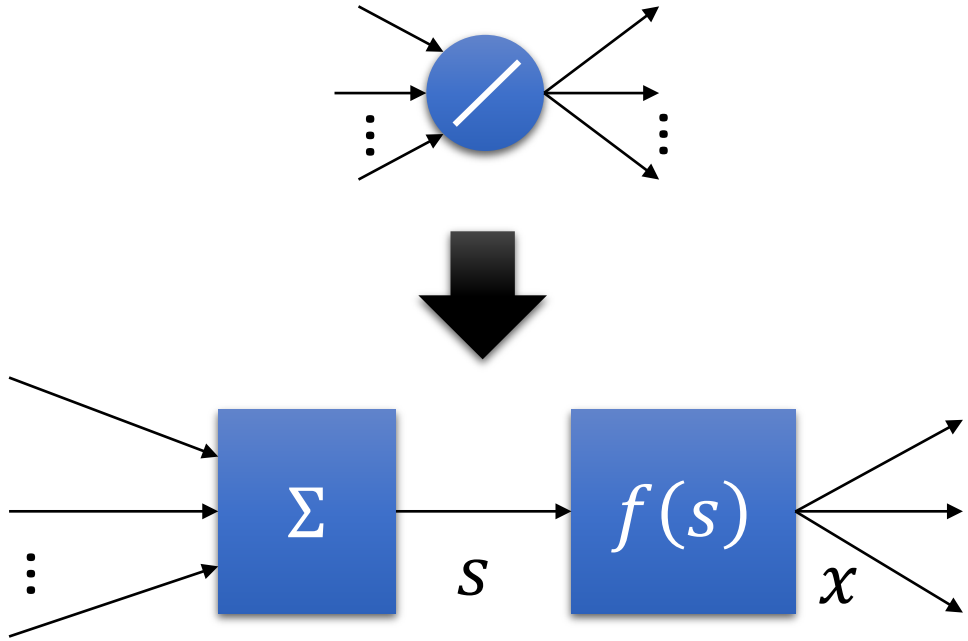
From Brains to Artificial Neural Networks



- Arrows indicate information flow
  - Feedforward networks have information flow in 1 direction

Artificial Neurons

- Facilitate integration of information and linear or non-linear computation with the ANN



- Neuron sums inputs and applies *activation function* $f(s)$

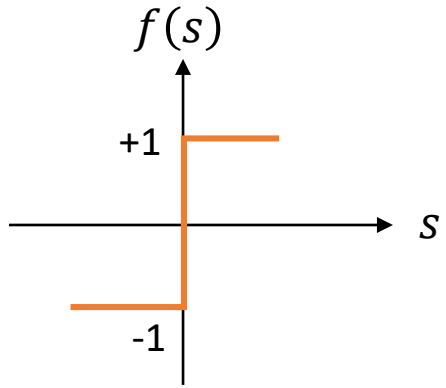
Artificial Neurons

- Activation functions can be discrete (digital), continuous (analog), or spiking

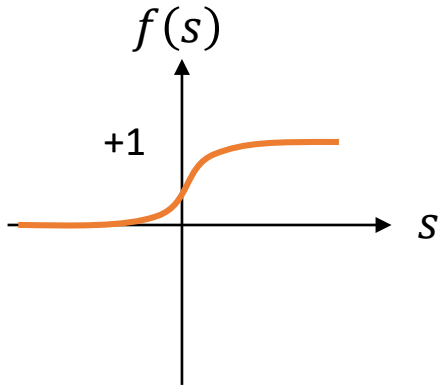
Threshold | Identity | Saturating Linear | Sigmoid | Spiking

- Can be unipolar or bipolar (not the same as biological neuron polarity)

Bipolar Threshold:

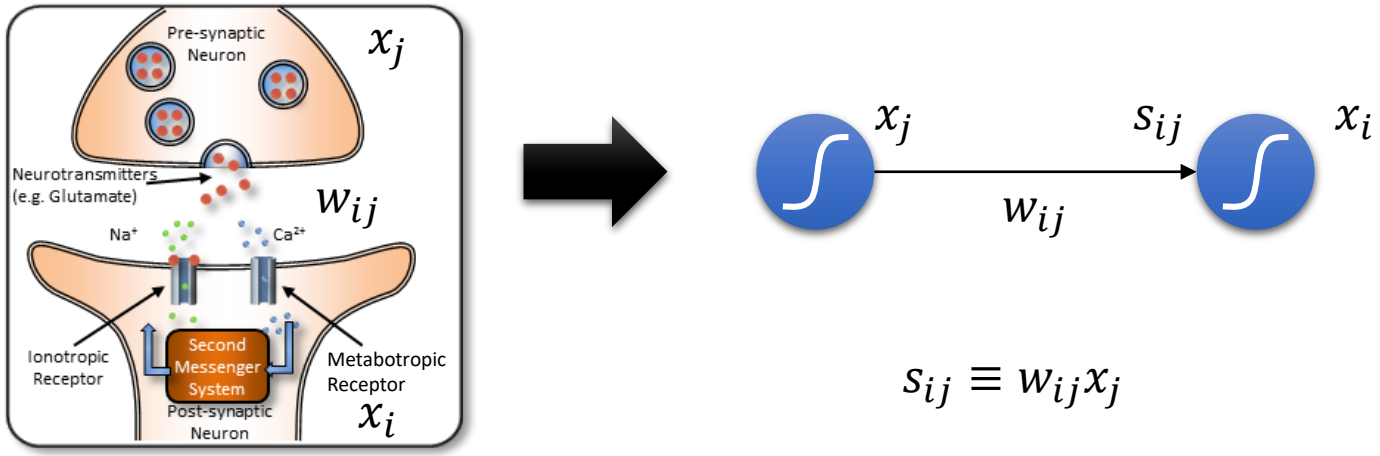


Unipolar Sigmoid:



Artificial Synapses

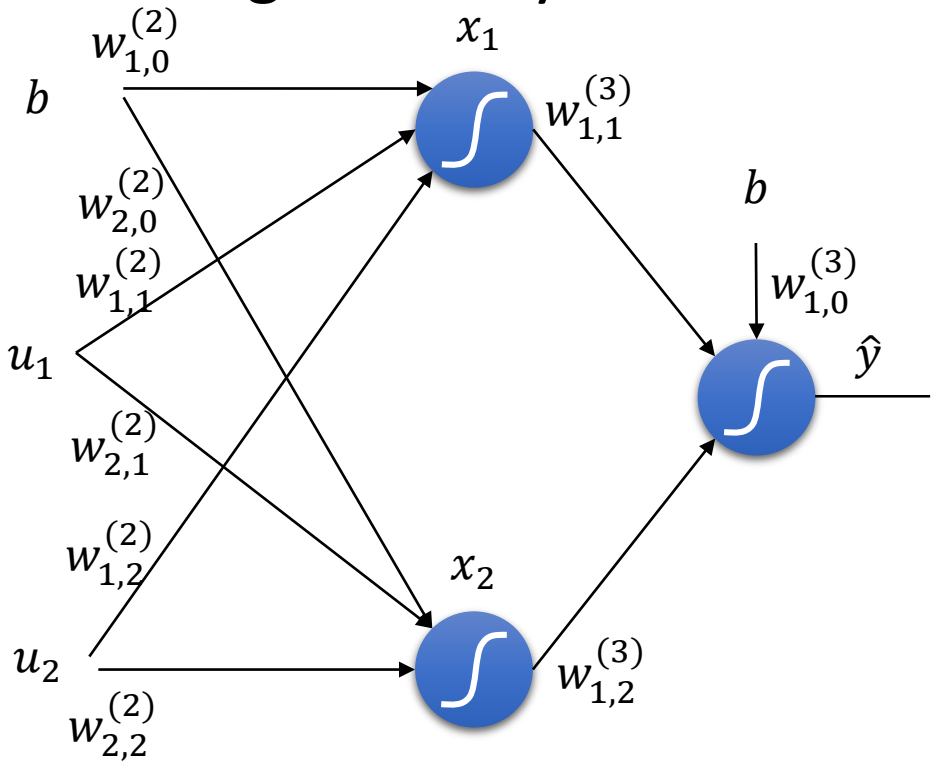
- Typically unidirectional (indicated by arrow)
- The complex behavior of the biological synapse is reduced to a *weight* value w that corresponds to
  - The strength or efficacy of the synaptic connection
  - The type of connection (excitatory or inhibitory)

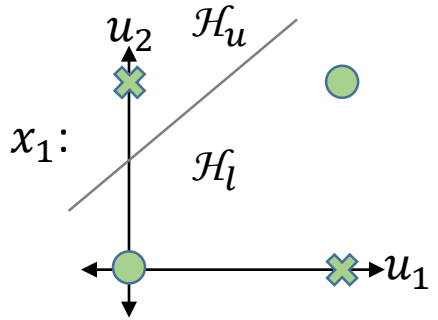


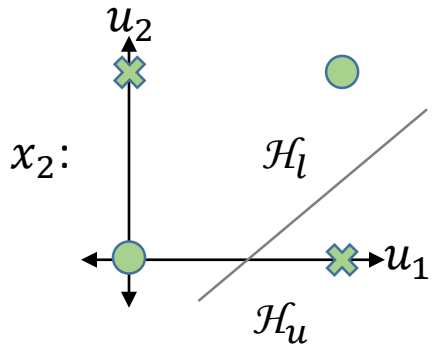
- Weights can be strictly positive (excitatory), strictly negative (inhibitory), or bipolar

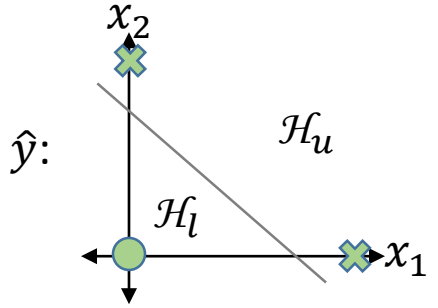
How Do We Solve Non-Linearly Separable Problems?

- Adding more layers:







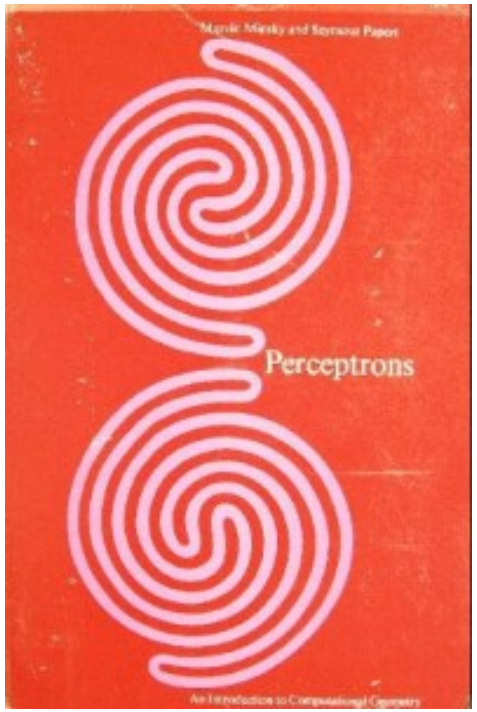


- The 2nd , or *hidden* layer (1st layer is the input layer) transforms the problem into a linearly separable one

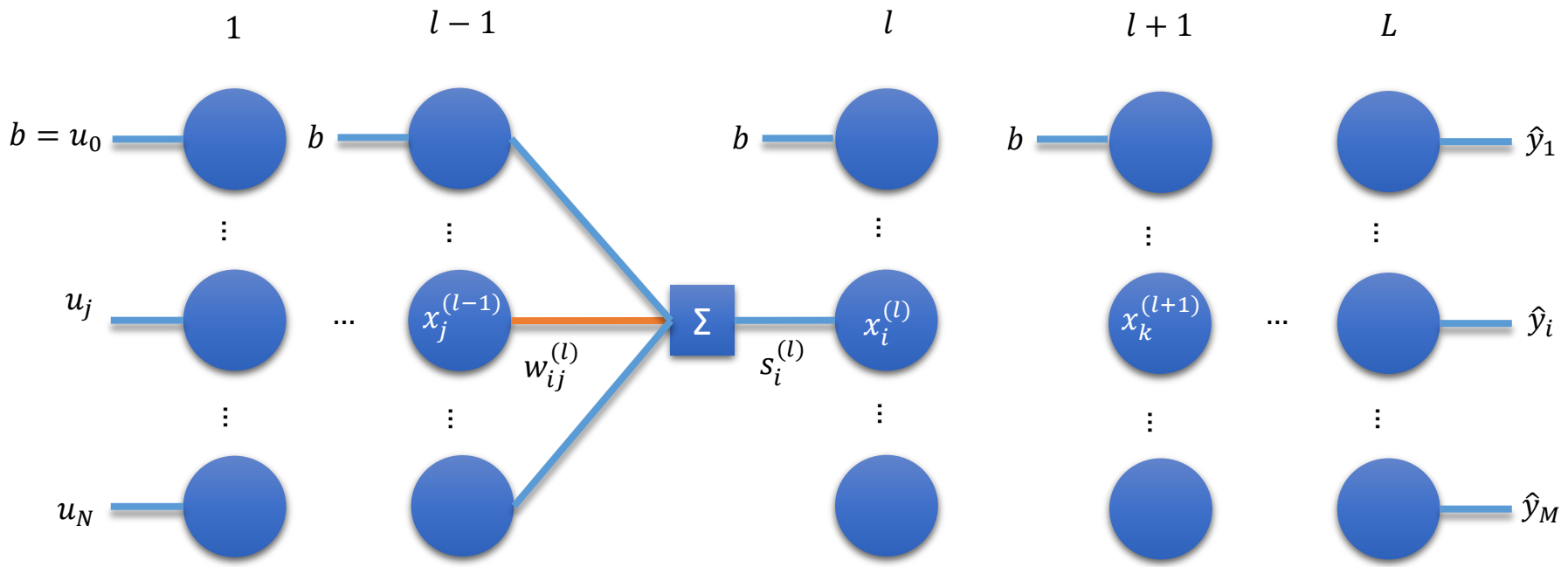
Perceptron Limitations

- Marvin Minsky (1969): *Perceptrons: An Introduction to Computational Geometry*

- Can only solve linearly separable problems



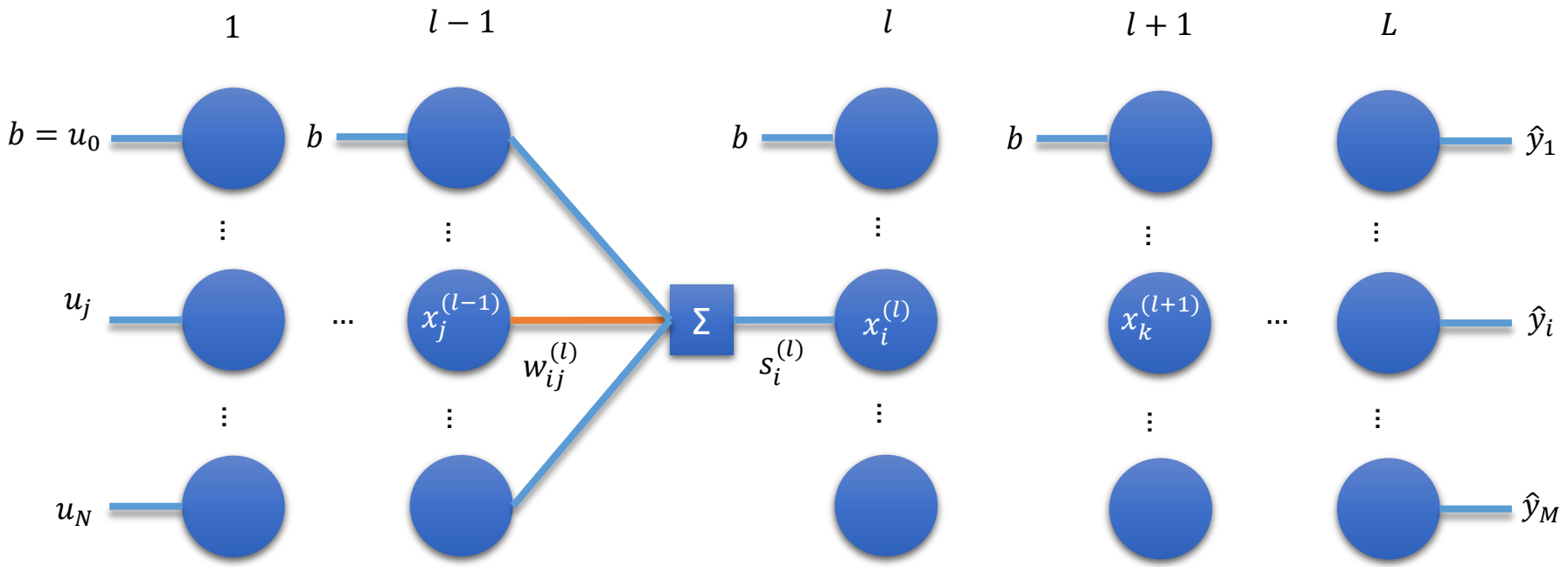
Multilayer Perceptron (MLP)



- Neuron outputs are calculated layer by layer, from 1 to L→*Forward propagation*

$$\mathbf{x}^{(l)} = \mathbf{W}^{(l)}\mathbf{x}^{(l-1)}$$

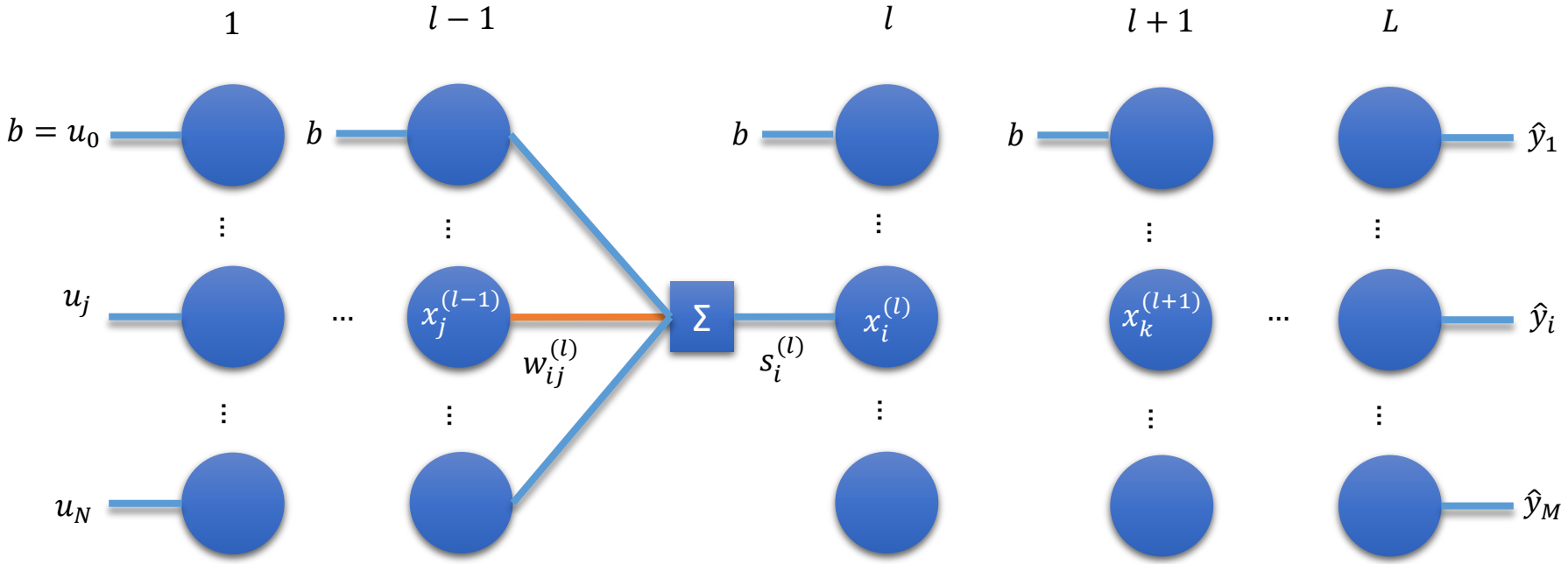
Multilayer Perceptron (MLP)



- Train the same as perceptron, linear regression, and logistic regression:

$$\Delta w_{ij}^{(l)} = -\alpha \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}}$$

Multilayer Perceptron (MLP)



- Cost function:

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{m}\sum_{p=1}^{m}\sum_{i=1}^{M} y_i^{(p)} \ln\left(h_{\mathbf{W}_i}\left(\mathbf{u}^{(p)}\right)\right) + \left(1 - y_i^{(p)}\right) \ln\left(1 - h_{\mathbf{W}_i}\left(\mathbf{u}^{(p)}\right)\right)$$

Backpropagation Algorithm

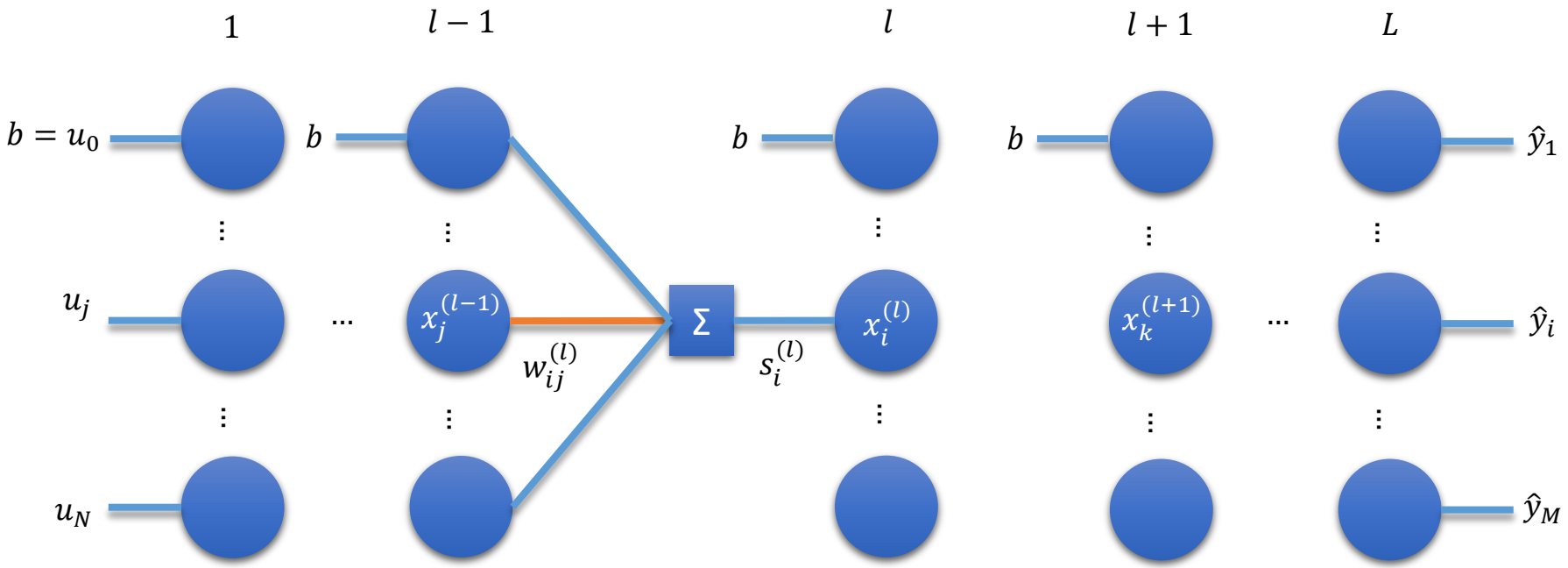
- Invented independently by Bryson and Ho (1969), Werbos (1974), Parker (1985), and Rumelhart (1986)

- One of the most widely-used NN training algorithms

- Simple application of the chain rule:

$$\frac{\partial f\big(g(x)\big)}{\partial x} = \frac{\partial f\big(g(x)\big)}{\partial g(x)} \frac{\partial g(x)}{\partial x}$$

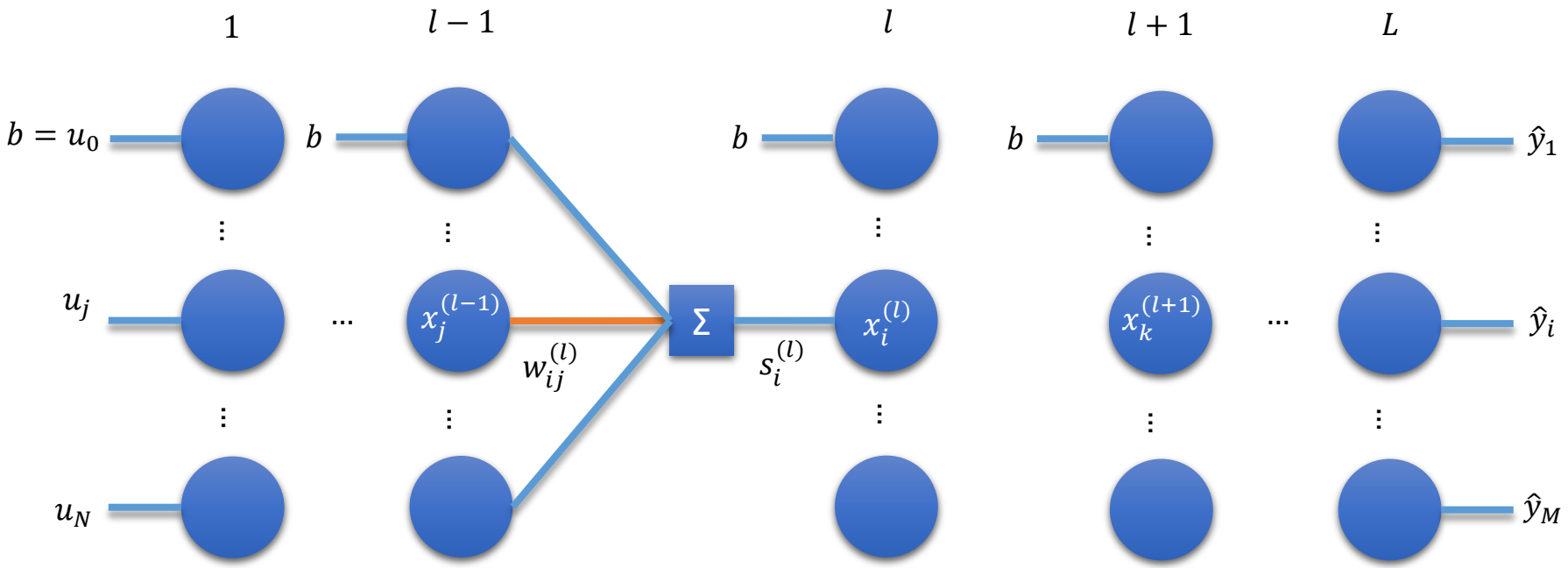
Backpropagation Algorithm Derivation



- Let's look at how a small change in the weight $w_{ij}^{(l)}$ affects the cost **for a single training pattern p:**

$$\frac{\partial \mathcal{L}^{(p)}}{\partial w_{ij}^{(l)}} = ???$$

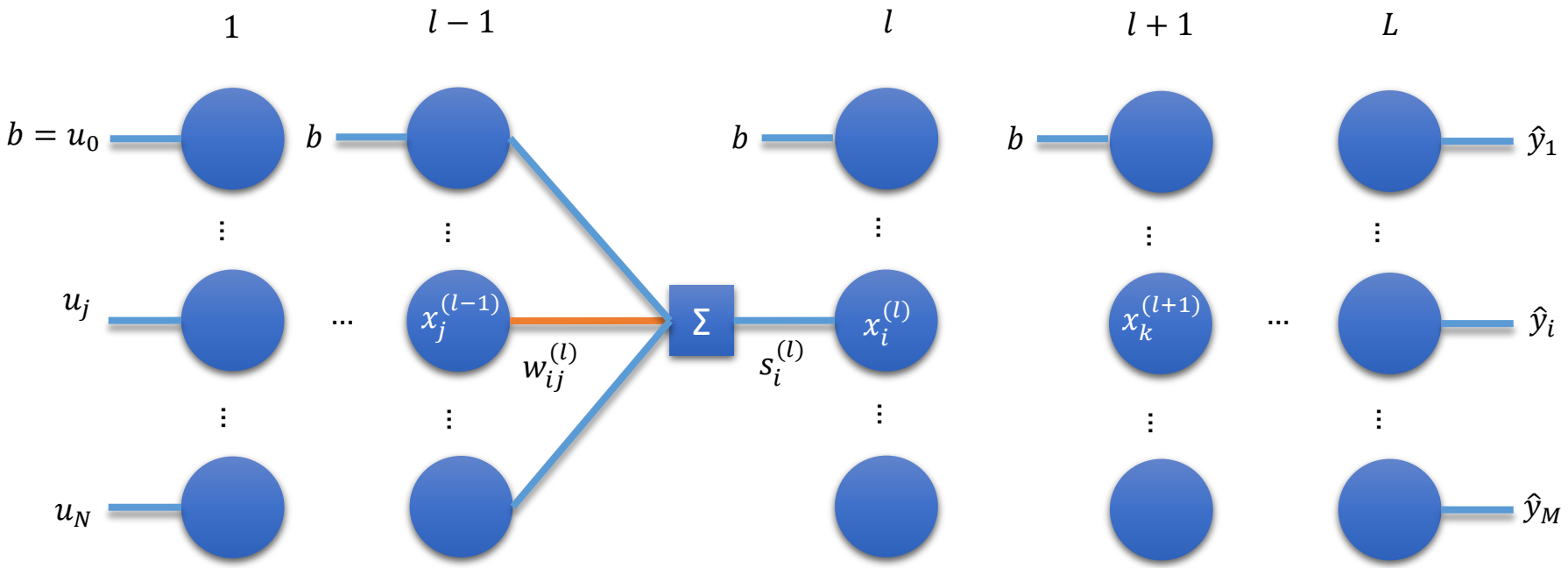
Backpropagation Algorithm Derivation



- Use the chain rule:

$$\frac{\partial \mathcal{L}^{(p)}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}^{(p)}}{\partial x_i^{(l,p)}} \frac{\partial x_i^{(l,p)}}{\partial s_i^{(l,p)}} \frac{\partial s_i^{(l,p)}}{\partial w_{ij}^{(l)}}$$

Backpropagation Algorithm Derivation



- Use the chain rule:

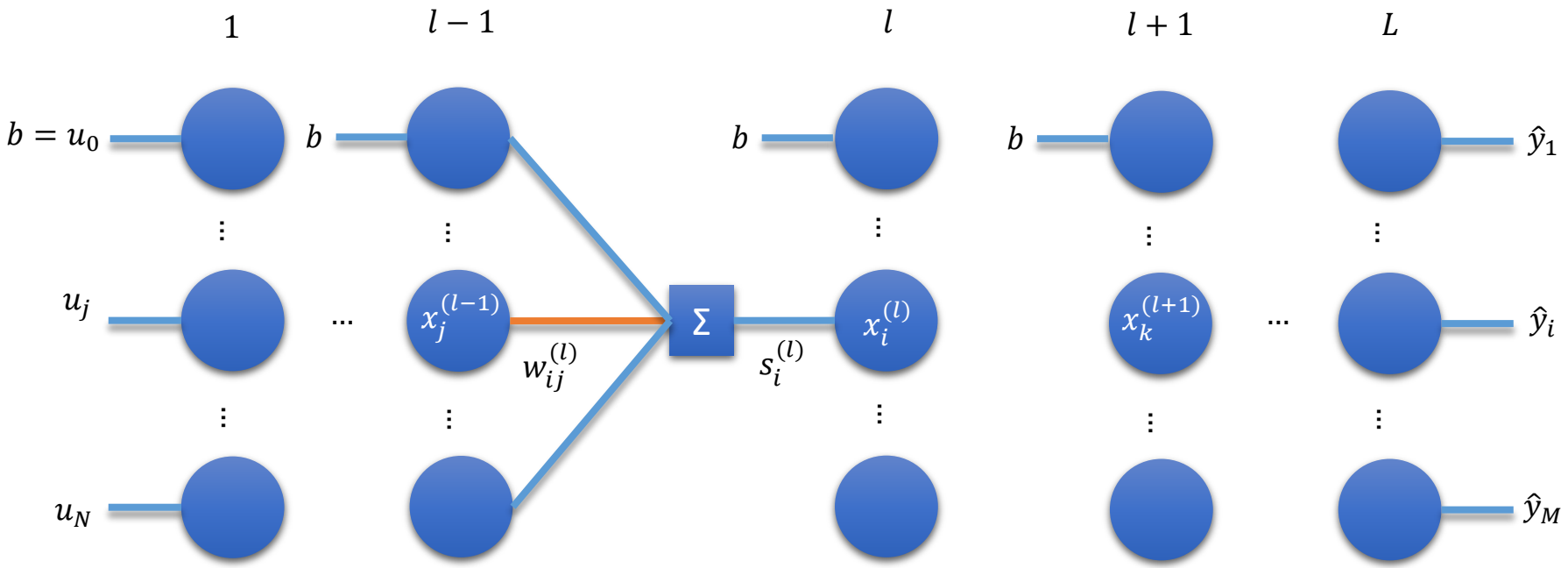
$$\frac{\partial \mathcal{L}^{(p)}}{\partial w_{ij}^{(l)}} = \left(\frac{\partial \mathcal{L}^{(p)}}{\partial x_i^{(l,p)}} \frac{\partial x_i^{(l,p)}}{\partial s_i^{(l,p)}} \right) \frac{\partial s_i^{(l,p)}}{\partial w_{ij}^{(l,p)}}$$

$$\left(s_i^{(l,p)} = \sum x_j^{(l-1,p)} w_{ij}^{(l)} \right)$$

$\delta_i^{(l,p)}$

$x_j^{(l-1,p)}$

Backpropagation Algorithm Derivation

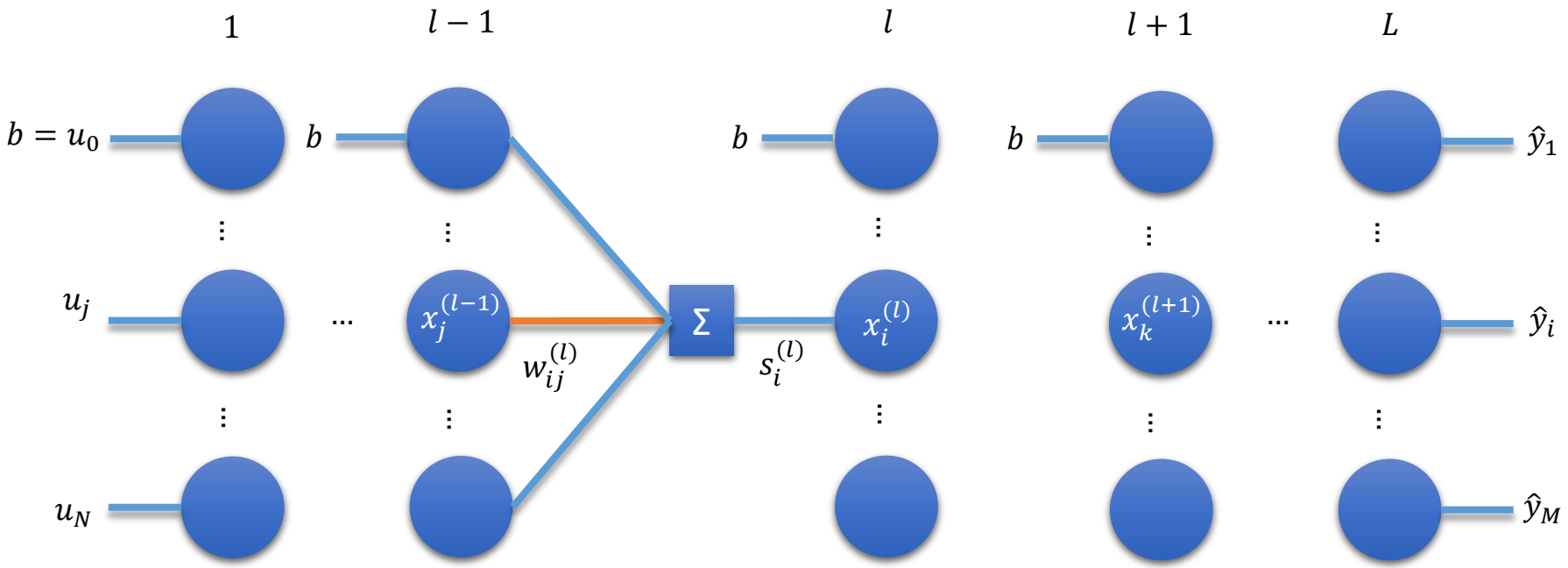


- Now:

$$\frac{\partial \mathcal{L}^{(p)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l,p)} x_j^{(l-1,p)}$$

- Just need to calculate $\delta_i^{(l,p)}$

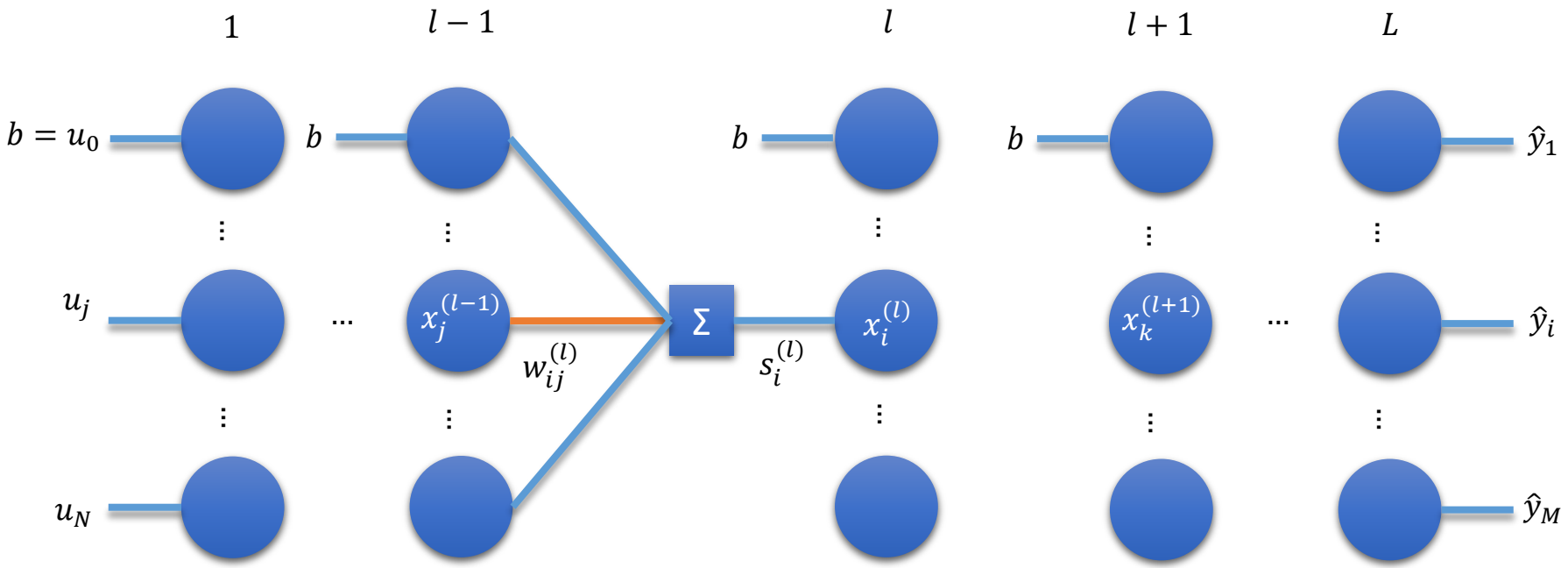
Backpropagation Algorithm Derivation



$$\delta_i^{(l,p)} \equiv \frac{\partial \mathcal{L}^{(p)}}{\partial x_i^{(l,p)}} \frac{\partial x_i^{(l,p)}}{\partial s_i^{(l,p)}}$$

$$f'\left(s_i^{(l,p)}\right)$$

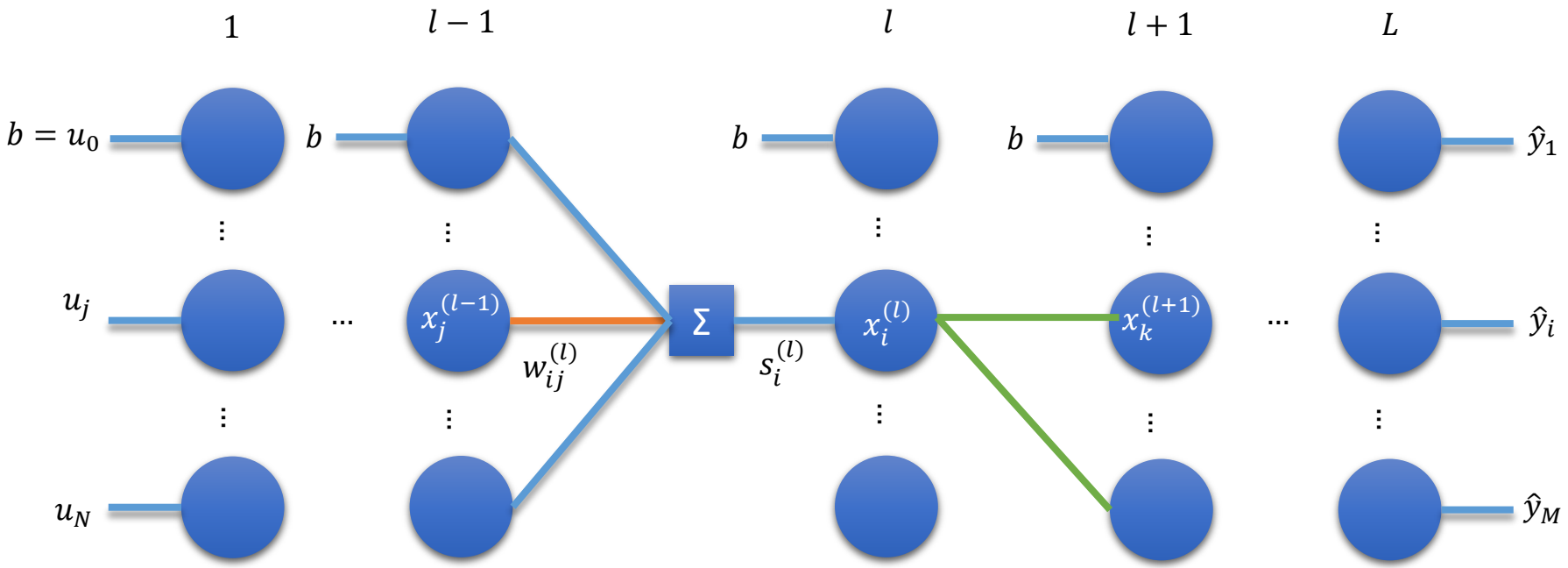
Backpropagation Algorithm Derivation



$$\delta_i^{(l,p)} = \frac{\partial \mathcal{L}^{(p)}}{\partial x_i^{(l,p)}} f'\left(s_i^{(l,p)}\right)$$

How does the output of a neuron affect the error?

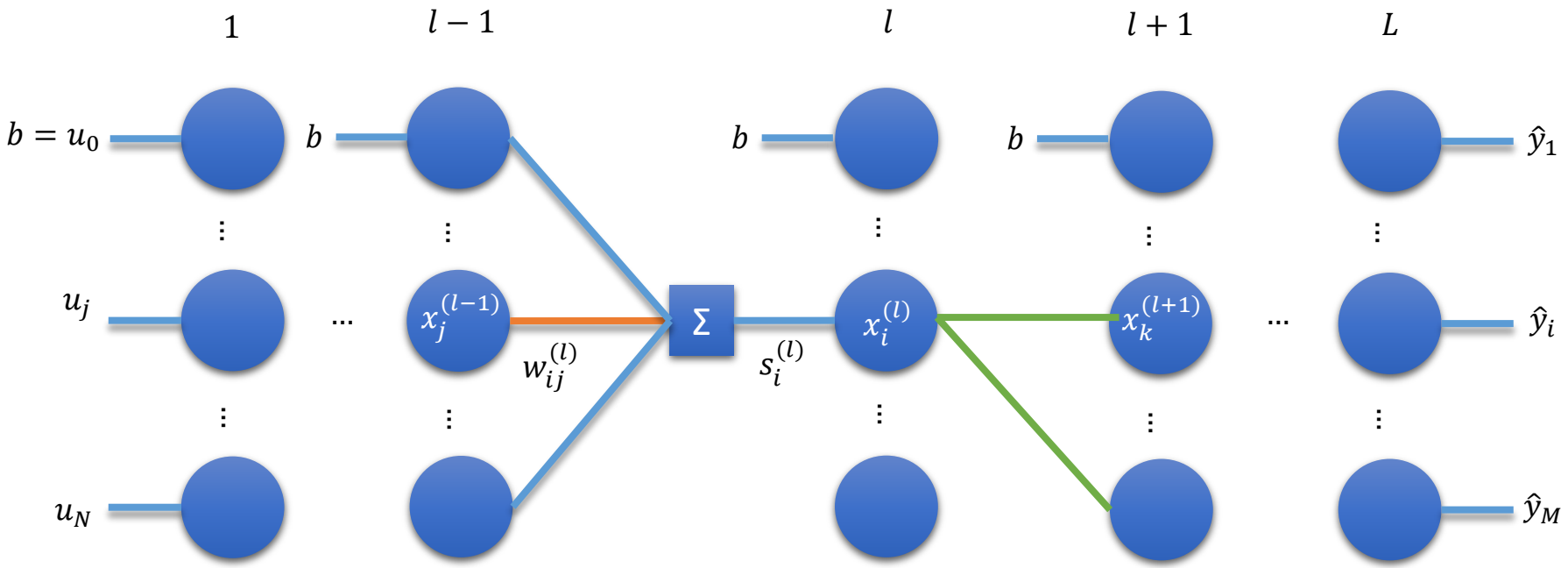
Backpropagation Algorithm Derivation



- Use the chain rule again:

$$\frac{\partial \mathcal{L}^{(p)}}{\partial x_i^{(l,p)}} = \sum_{k=1}^{N_{l+1}} \underbrace{\frac{\partial \mathcal{L}^{(p)}}{\partial x_k^{(l+1,p)}} \frac{\partial x_k^{(l+1,p)}}{\partial s_k^{(l+1,p)}}}_{\delta_k^{(l+1,p)}} \underbrace{\frac{\partial s_k^{(l+1,p)}}{\partial x_i^{(l,p)}}}_{w_{ki}^{(l+1)}}$$

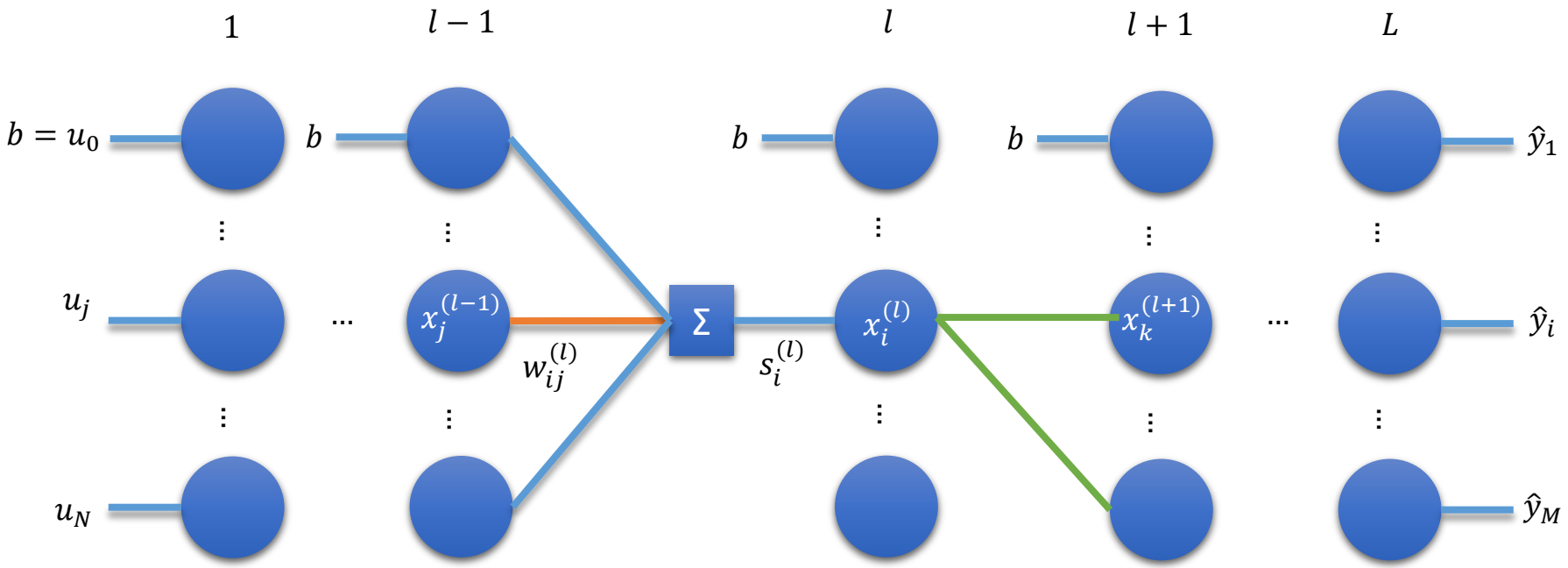
Backpropagation Algorithm Derivation



$$\delta_i^{(l,p)} = \left[\sum_{k=1}^{N_{l+1}} \delta_k^{(l+1,p)} w_{ki}^{(l+1)}\right] f'\left(s_i^{(l,p)}\right)$$

$$= \left(\boldsymbol{\delta}^{(l+1,p)} \cdot \mathbf{w}_i^{(l+1)}\right) f'\left(s_i^{(l,p)}\right)$$

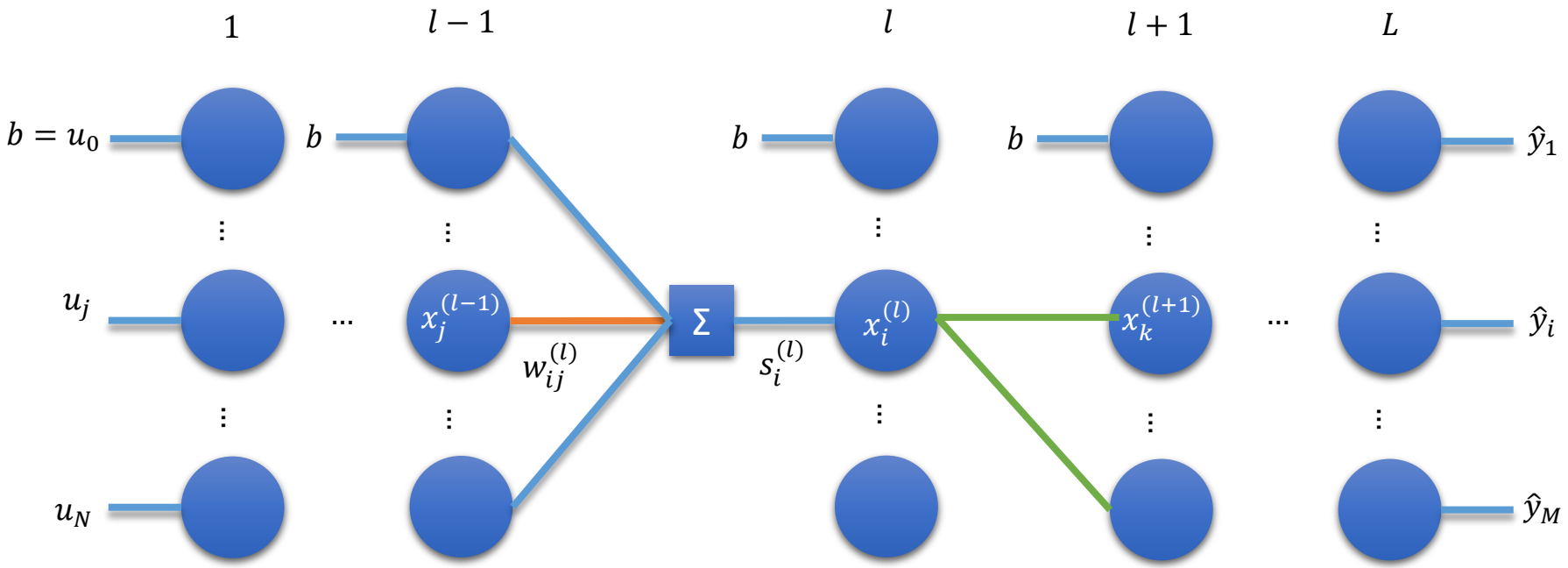
Backpropagation Algorithm Derivation



- For the output layer:

$$\delta_i^{(L,p)} = \frac{\partial \mathcal{L}^{(p)}}{\partial x_i^{(L,p)}} f'\left(s_i^{(L,p)}\right) = \hat{y}_i^{(p)} - y_i^{(p)}$$

Backpropagation Algorithm Derivation



- Finally, consider all of the patterns:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \sum_{p=1}^{m} \delta_i^{(l)} x_j^{(l-1)}$$

Putting it All Together

initialize weights to random values
for epoch=1:epochs
        for p=1:m

% Calculate the outputs using forward propagation
$\hat{y}_i^{(p)} = \cdots \forall i = 1 \text{ to } M$

% Calculate the output delta values:
$\delta_i^{(L,p)} = \hat{y}_i^{(p)} - y_i^{(p)} \; \forall i = 1 \text{ to } M$

% Calculate the other delta values
$\delta_i^{(l,p)} = \left(\boldsymbol{\delta}^{(l+1,p)} \cdot \mathbf{w}_i^{(l+1)}\right) f'\left(s_i^{(l,p)}\right) \forall i = 0 \text{ to } N_l \; \forall l = L-1 \text{ to } 2$

% Update the cost derivative
$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} \mathrel{+}= \delta_i^{(l,p)} x_j^{(l-1,p)} \forall i, j, l$

% Update all of the weight values
$w_{ij}^{(l)} \coloneqq w_{ij}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} \forall i, j, l$