Foundations of Machine Learning

Module 6: Neural Network
Part B: Multi-layer Neural Network

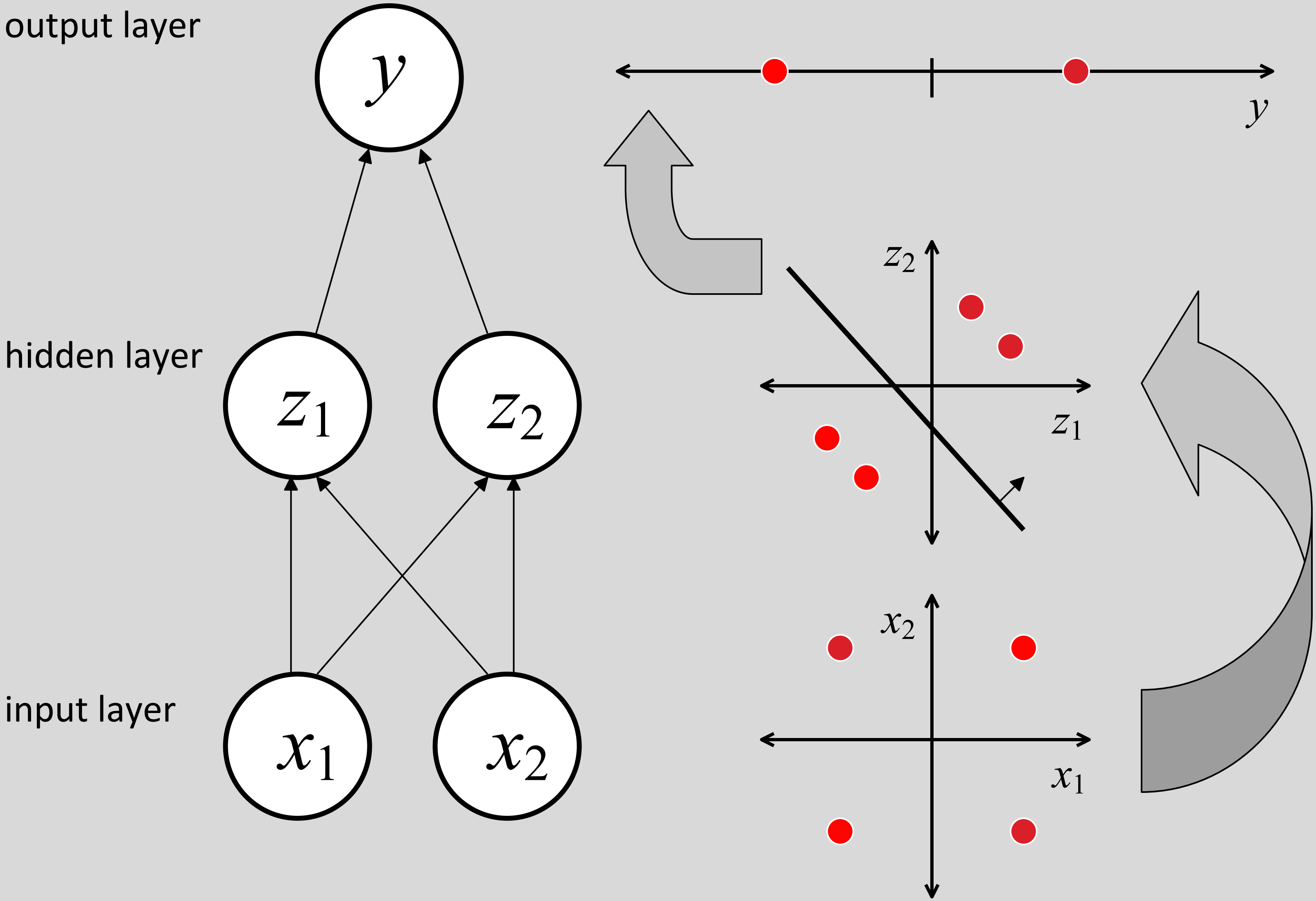
Sudeshna Sarkar
IIT Kharagpur

Limitations of Perceptrons

- Perceptrons have a *monotinicity* property:

  If a link has positive weight, activation can only increase as the corresponding input value increases (*irrespective* of other input values)
- Can't represent functions where input *interactions* can cancel one another's effect (e.g. XOR)
- Can represent only linearly separable functions

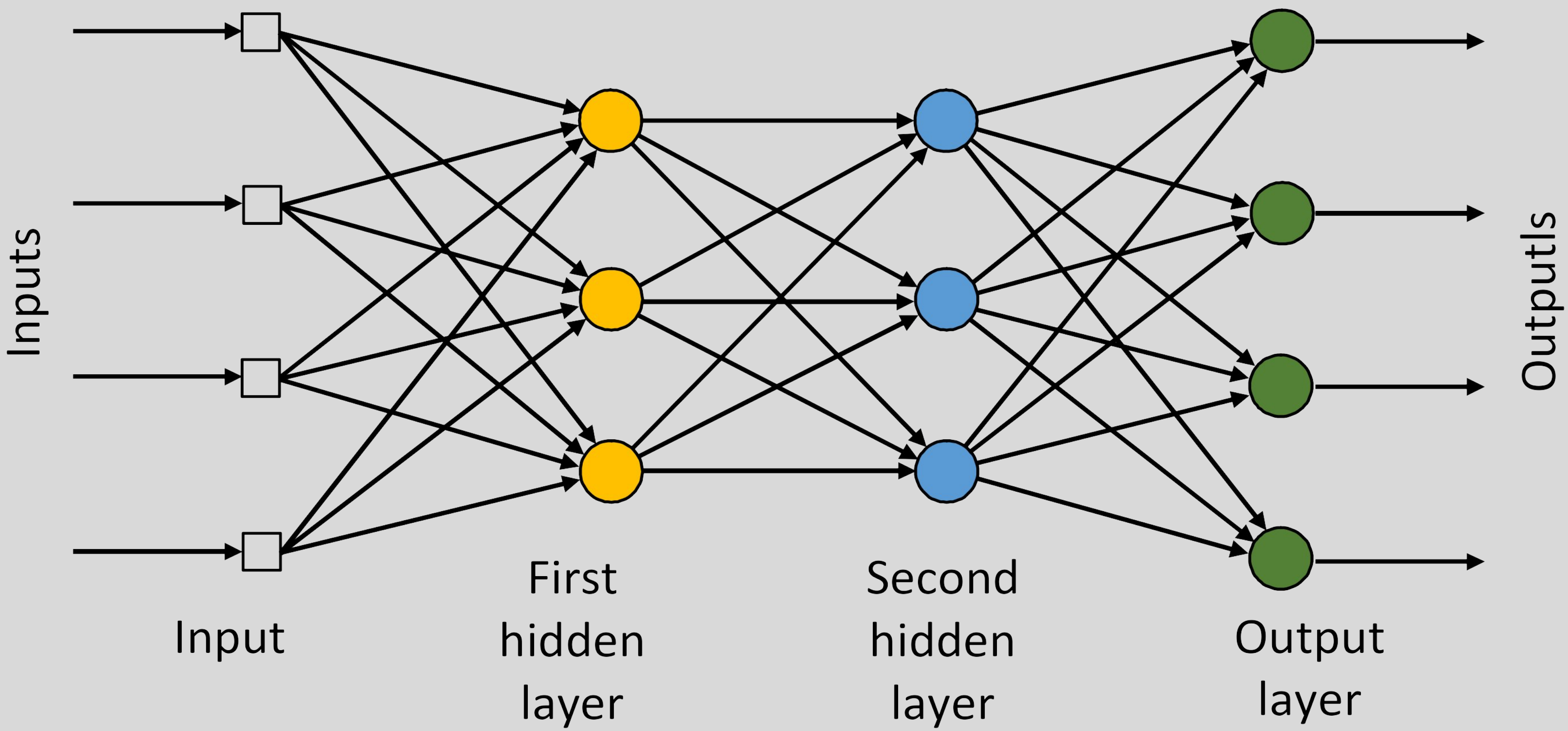
A solution: multiple layers



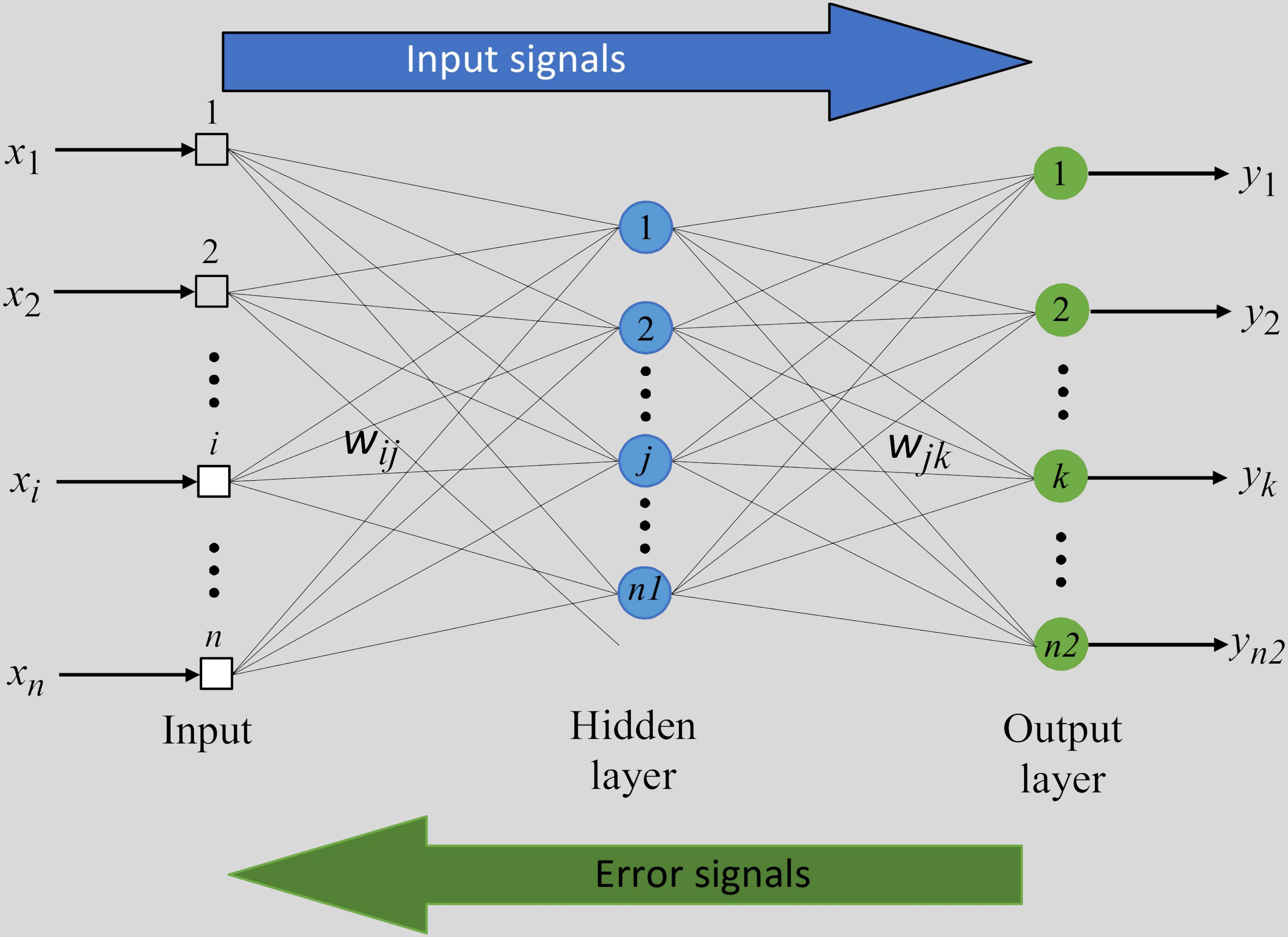
Power/Expressiveness of Multilayer Networks

- Can represent interactions among inputs
- Two layer networks can represent any Boolean function, and continuous functions (within a tolerance) as long as the number of hidden units is sufficient and appropriate activation functions used
- Learning algorithms exist, but weaker guarantees than perceptron learning algorithms

Multilayer Network

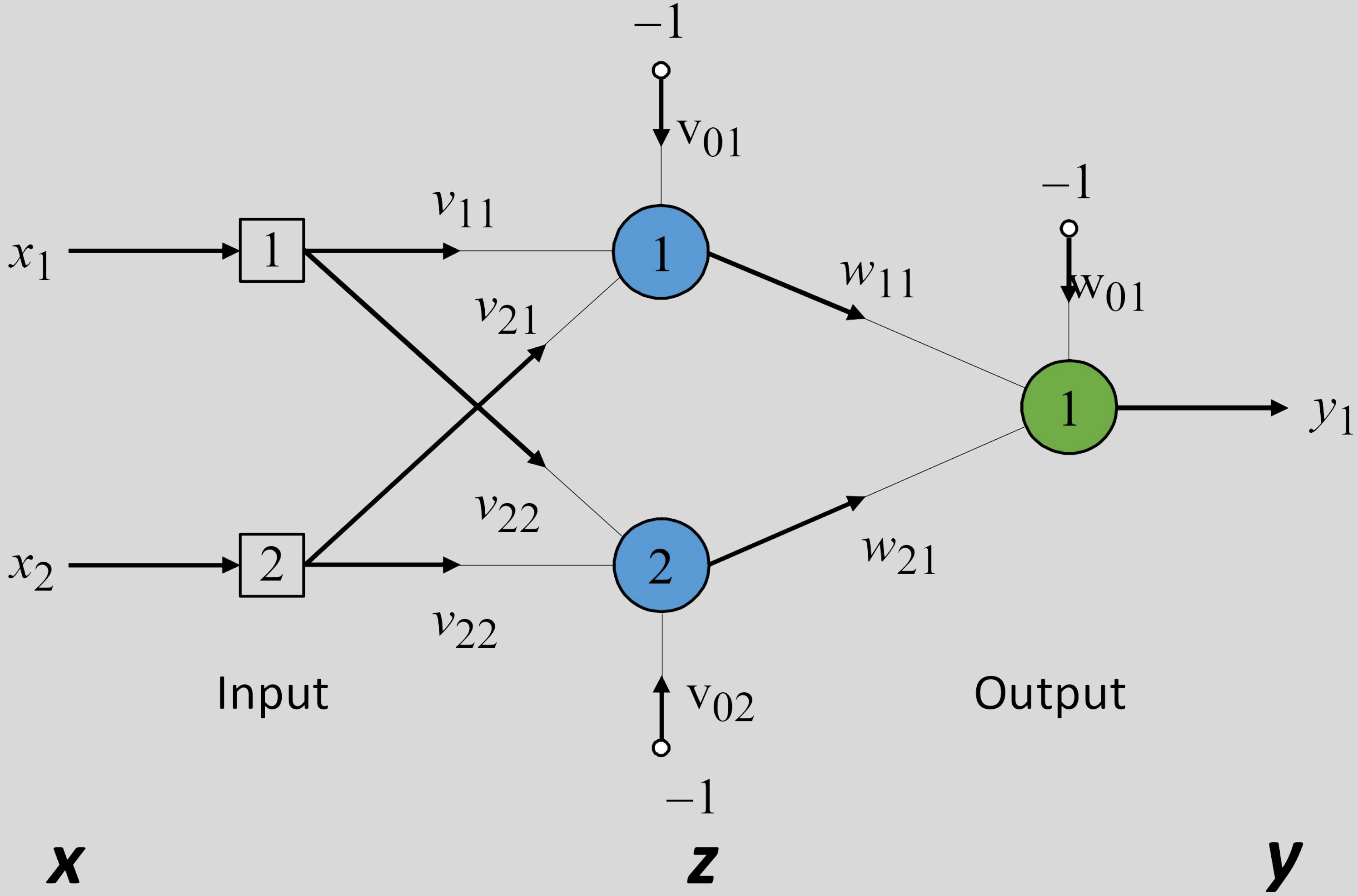


Two-layer back-propagation neural network



The back-propagation training algorithm

- Step 1: Initialisation
  Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range



Backprop

- Initialization
  - Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range
- Forward computing:
  - Apply an input vector ***x*** to input units
  - Compute activation/output vector ***z*** on hidden layer

  $z_j = \varphi(\sum_i v_{ij} x_i)$
  - Compute the output vector ***y*** on output layer

  $y_k = \varphi(\sum_j w_{jk} z_j)$

  y is the result of the computation.

Learning for BP Nets

- Update of weights in W (between output and hidden layers):
  - delta rule
- Not applicable to updating V (between input and hidden)
  - don't know the target values for hidden units z1, Z2, … ,ZP
- Solution: Propagate errors at output units to hidden units to drive the update of weights in V (again by delta rule) (error BACKPROPAGATION learning)
- Error backpropagation can be continued downward if the net has more than one hidden layer.
- How to compute errors on hidden units?

Derivation

- For one output neuron, the error function is
$E = \frac{1}{2}(y - \hat{y})^2$
- For each unit j, the output o_j is defined as
$o_j = \varphi\left(net_j\right) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$
The input net_j to a neuron is the weighted sum of outputs o_k of previous n neurons.
- Finding the derivative of the error:
$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$

Derivation

- Finding the derivative of the error:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = o_i$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\varphi\left(net_j\right) = \varphi\left(net_j\right)\left(1-\varphi\left(net_j\right)\right)$$

Consider E as as a function of the inputs of all neurons $Z = \{z_1, z_2, \ldots\}$ receiving input from neuron j,

$$\frac{\partial E\left(o_j\right)}{\partial o_j} = \frac{\partial E\left(net_{z_1}, net_{z_2}, \ldots\right)}{\partial o_j}$$

taking the total derivative with respect to o_j, a recursive expression for the derivative is obtained:

$$\frac{\partial E}{\partial o_j} = \sum_l \left(\frac{\partial E}{\partial net_{z_l}}\frac{\partial net_{z_l}}{\partial o_j}\right) = \sum_l \left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_{z_l}} w_{jz_l}\right)$$

$$\frac{\partial E}{\partial o_j} = \sum_l \left(\frac{\partial E}{\partial net_{z_l}} \frac{\partial net_{z_l}}{\partial o_j} \right) = \sum_l \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial net_{z_l}} w_{jz_l} \right)$$

- Therefore, the derivative with respect to o_j can be calculated if all the derivatives with respect to the outputs o_{z_l} of the next layer – the one closer to the output neuron – are known.
- Putting it all together:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

With

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = \begin{Bmatrix} (o_j - t_j)\, o_j (1 - o_j) \text{ if } j \text{ is an output neuron} \\ \left(\sum_Z \delta_{z_l} w_{jl}\right) o_j (1 - o_j) \text{ if } j \text{ is an inner neuron} \end{Bmatrix}$$

To update the weight w_{ij} using gradient descent, one must choose a learning r

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

Backpropagation Algorithm

Thank You