Multi-Layer Perceptrons

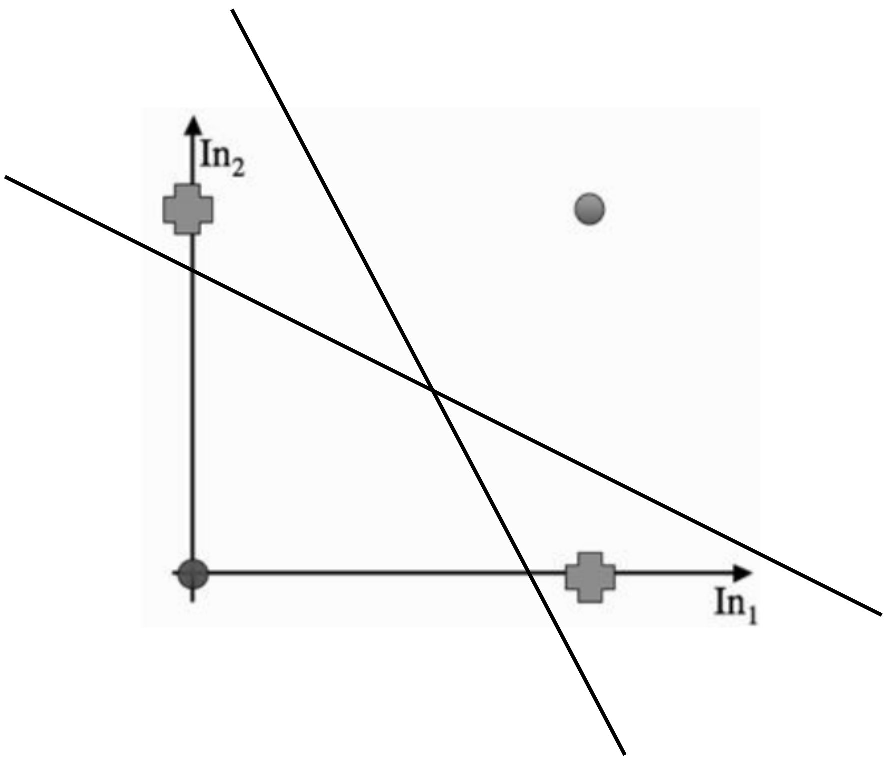
Learning outcomes

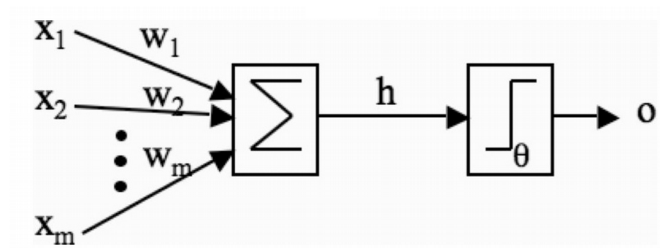
UNIVERSITY OF LEEDS

- Construct a multi-layer neural network that classifies a given dataset in 2D, overcoming the limitation on learning separability.
- Define an appropriate error to minimise for Feed-forward neural networks.
- Derive the update rule of the weights of the NN, through backpropagation.
- Apply NNs to real-world data sets

Perceptron limitations

XOR



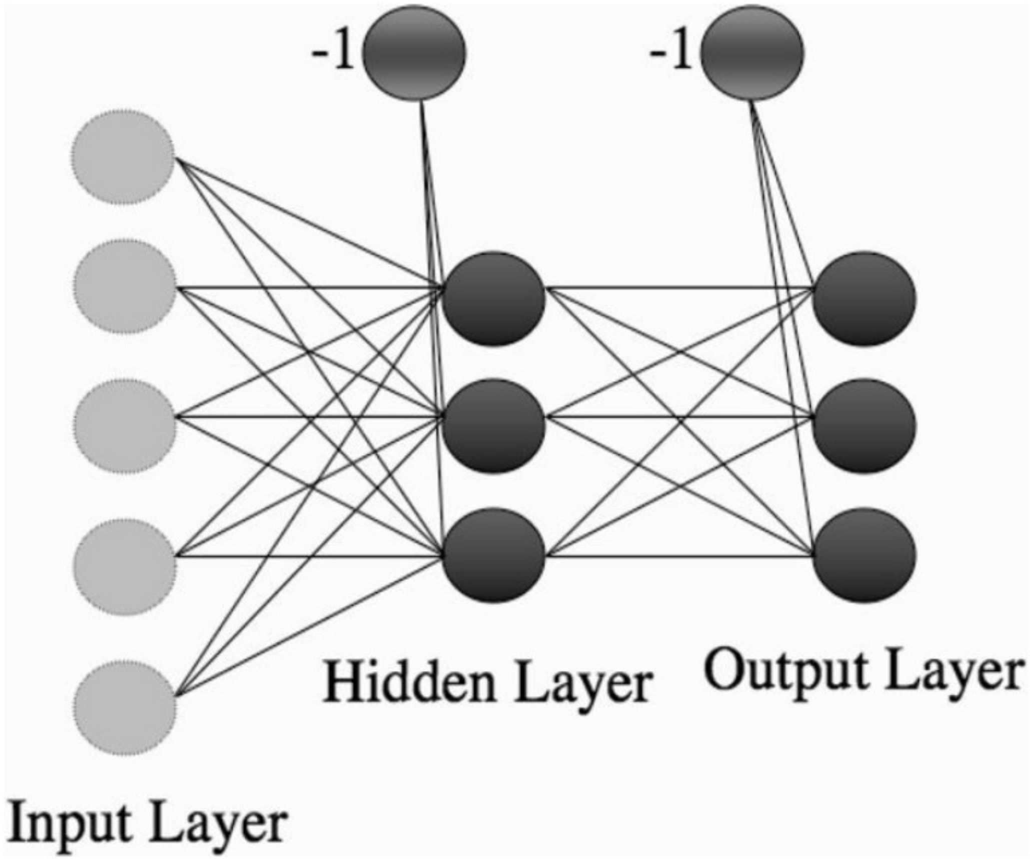


$$h_w(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 > 0$$

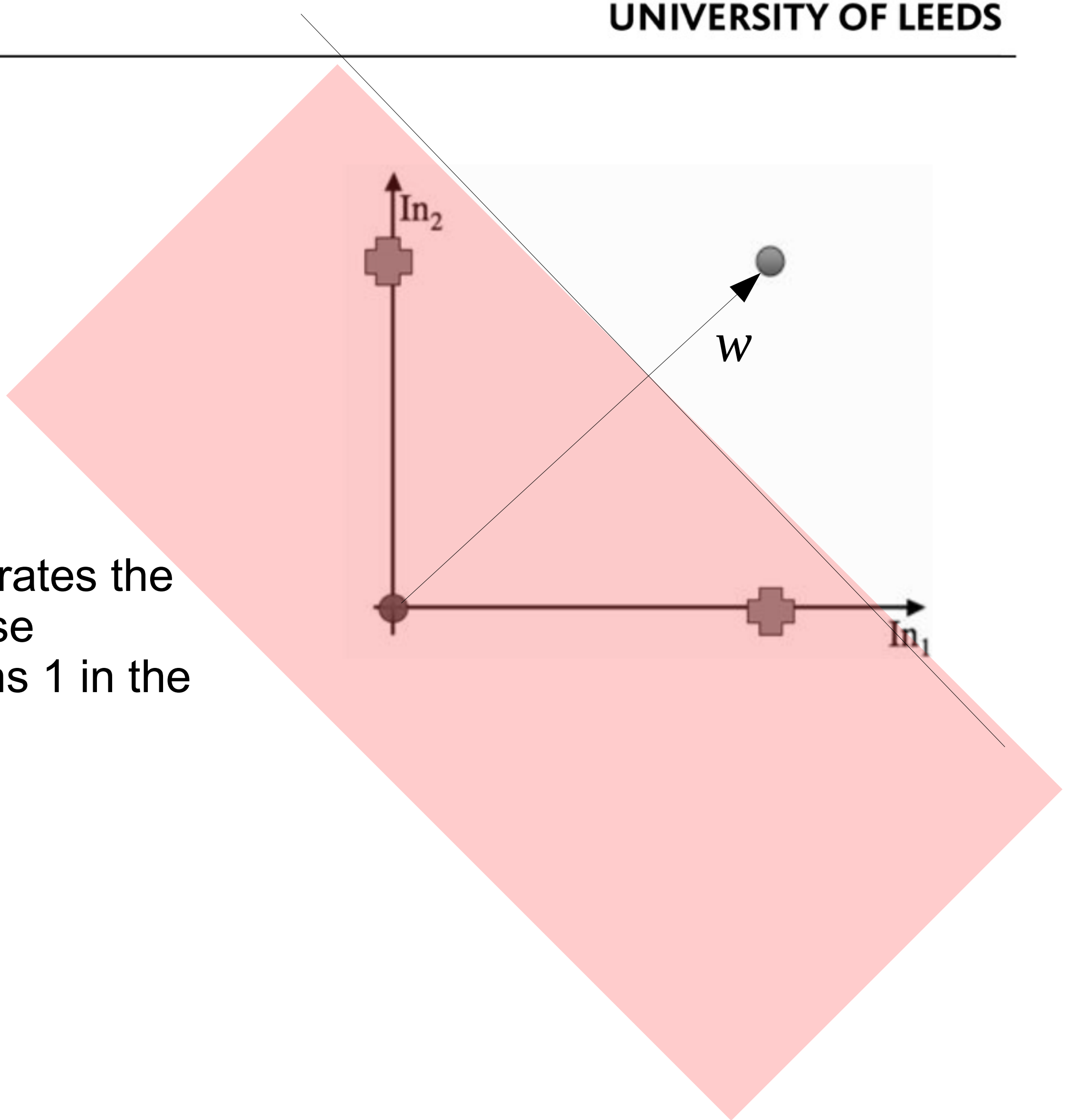
Multi-layer Perceptron

UNIVERSITY OF LEEDS

Multi-layer perceptrons are used to obtain non-linear classifers.



MLP and XOR



Choose a straight line that separates the points as in the figure, and whose corresponding perceptron returns 1 in the highlighted area

Possible solution:

$$-2x_1 - x_2 + 2.5 = 0$$

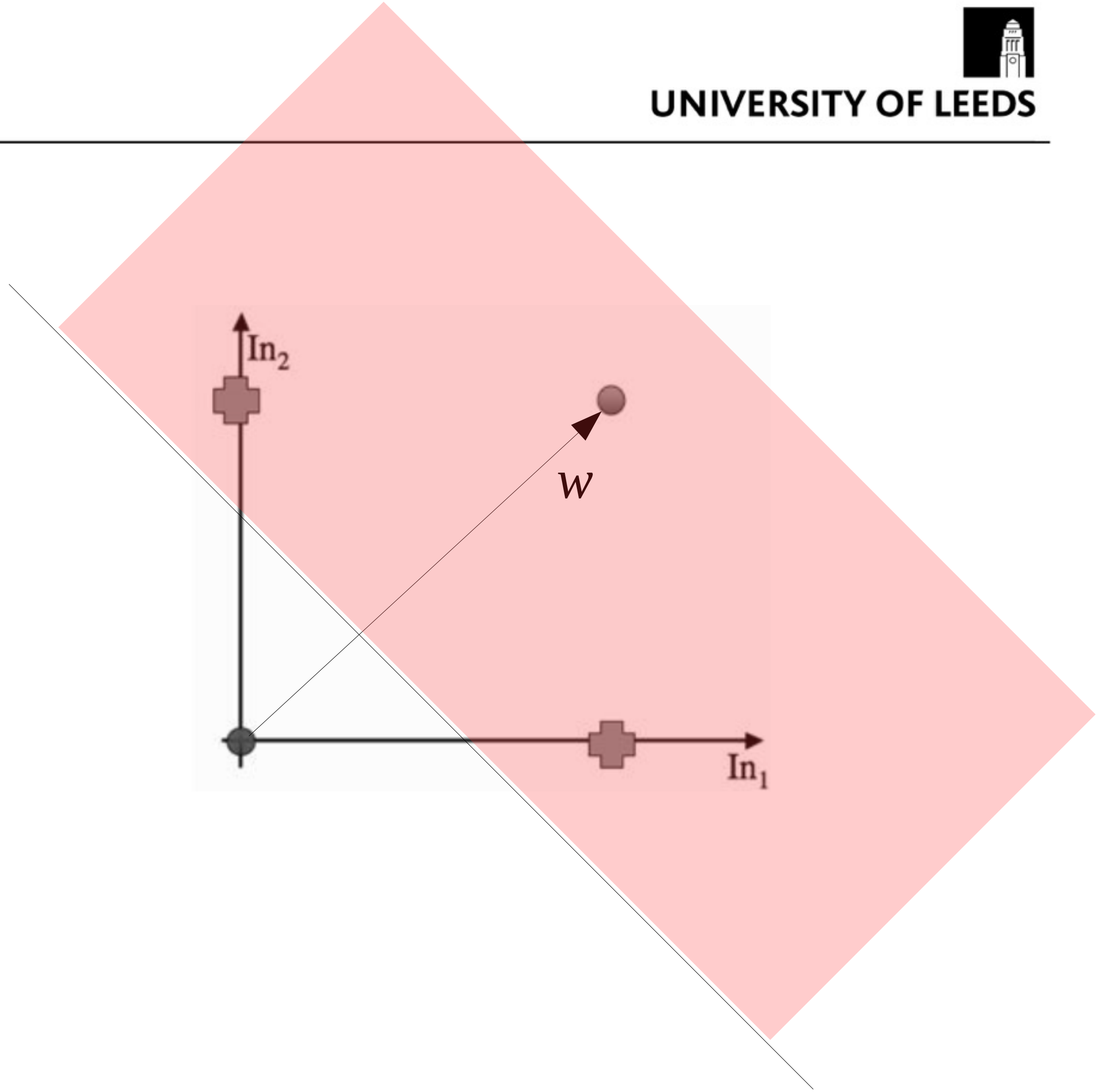
$$w = \langle -2, -1 \rangle$$

MLP and XOR

Now for the other points:

$$x_1 + x_2 - 0.5 = 0$$

$$w = \langle 1,1 \rangle$$



MLP and XOR

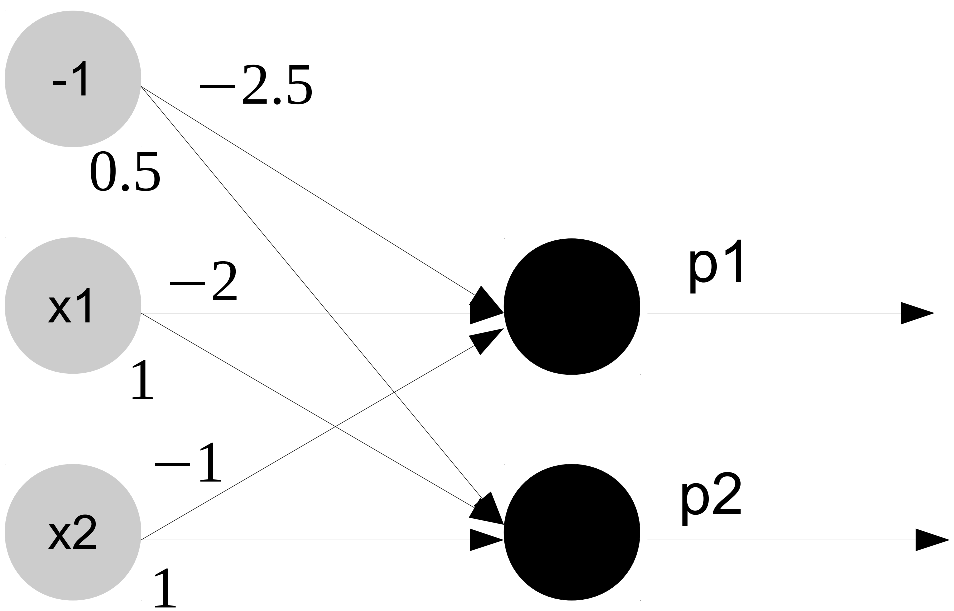
$$-2x_1 - x_2 + 2.5 \geq 0$$

$$x_1 + x_2 - 0.5 \geq 0$$

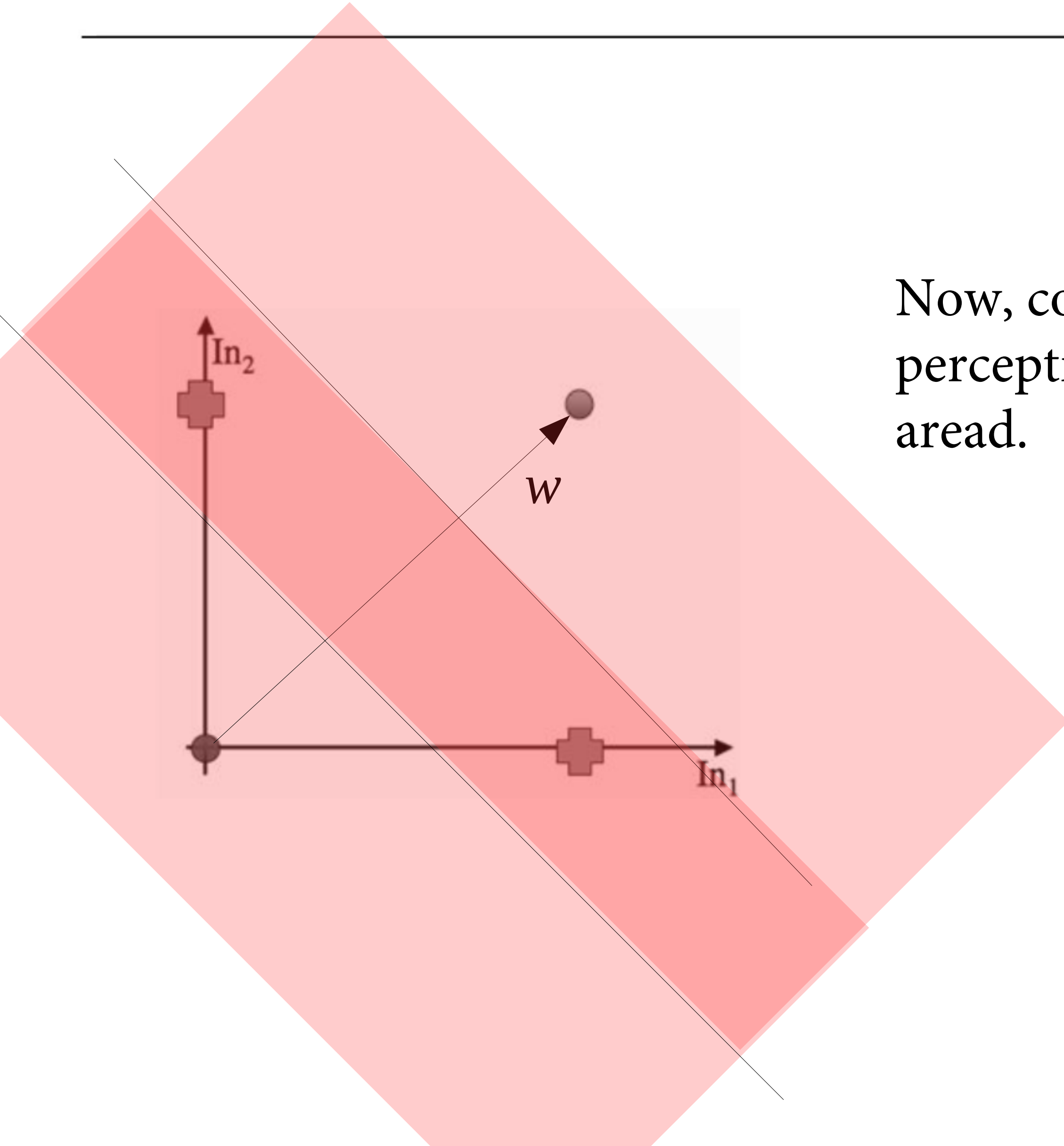
These are 2 perceptrons with weights:

$$\langle -2, -1, -2.5 \rangle$$

$$\langle 1, 1, 0.5 \rangle$$



MLP and XOR



Now, combine the outputs of those perceptrons by intersecting the shaded aread.

MLP and XOR

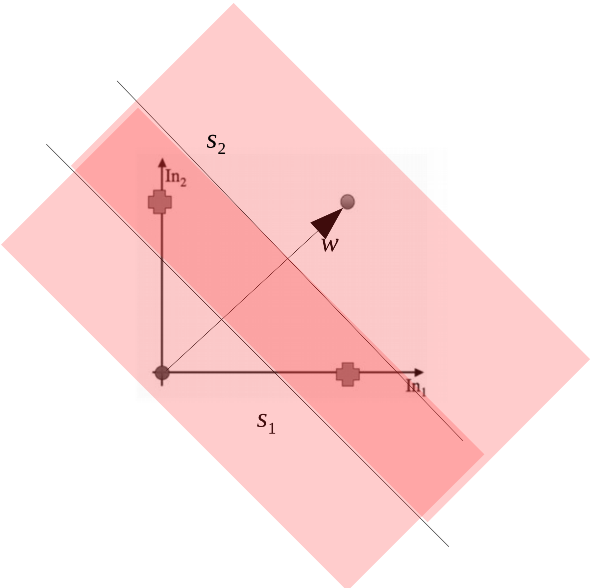
Their outputs are:

What we want

x1	x2	p1	p2	o
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

$$p_1 = -2x_1 - x_2 + 2.5 \geq 0$$

$$p_2 = x_1 + x_2 - 0.5 \geq 0$$

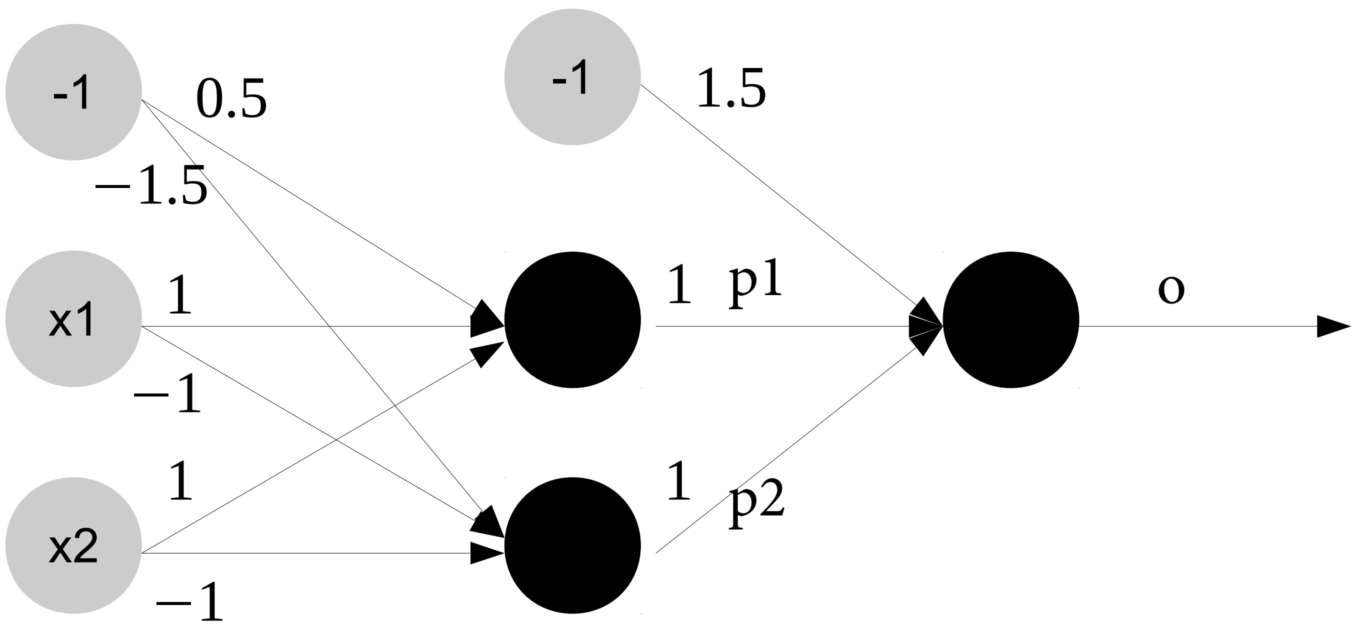


MLP and XOR

$$p_1 \equiv x_1 + x_2 - 0.5 \geq 0$$

$$p_2 \equiv -x_1 - x_2 + 1.5 \geq 0$$

$$o \equiv p_1 + p_2 - 1.5 \geq 0$$



MLP: A Universal Approximator

UNIVERSITY OF LEEDS

$$g(x)=\sum_{j}^{N} w_j \sigma(y_j^T x+\theta_j) \qquad \text{given} \qquad f(x) \qquad \epsilon>0$$

$$|g(x)-f(x)|<\epsilon$$

MLP is a universal function approximator, that is, it can represent any function. From a theoretical point of view, this can be done with a single hidden layer.
In practice, that hidden layer would have to be incredibly large.
Recent development in neural networks (often referred to as “deep” learning), favour “deep” structures, where many layers can be used to create an hierarchical classification.

Error definition

$$E(\boldsymbol{X}) = \sum_{\boldsymbol{x}_n \in \boldsymbol{X}} |y_n - t_n|$$

Number of errors on the training set

$$E_p(\boldsymbol{X}) = \sum_{\boldsymbol{x}_n \in \boldsymbol{X}} \boldsymbol{w}^T \boldsymbol{x}_n (y_n - t_n)$$

The Perceptron error

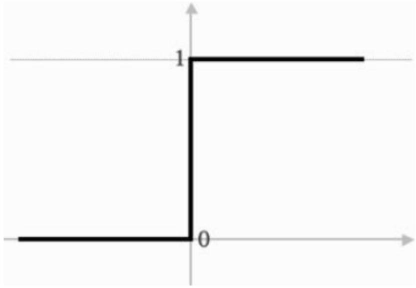
$$E_m(\boldsymbol{X}) = \frac{1}{2} \sum_{\boldsymbol{x}_n \in \boldsymbol{X}} (y_n - t_n)^2$$

Squared error function (differentiable!)
Usually known as the Mean Squared Error (MSE)

$$y = h\left(\sum_{i=1}^{M} w_i x_i\right)$$

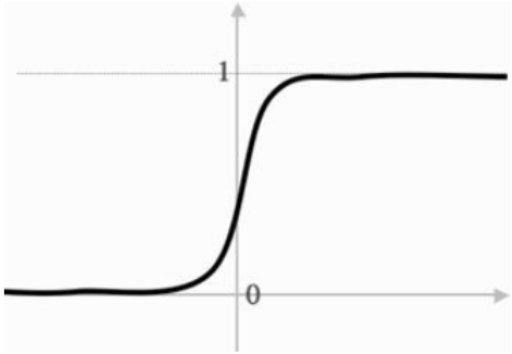
Output is differentiable if h is

$h =$



Not good

A different activation function



The sigmoid function: $$h(x)=\frac{1}{1+e^{-\beta x}}\equiv\sigma_\beta$$

$$\sigma_\beta'(x)=?$$

The derivative of the sigmoid

The sigmoid function: $$h(x) = \frac{1}{1+e^{-\beta x}} \equiv \sigma_\beta$$

$$\sigma_\beta'(x) = ?$$

Two useful properties of derivatives:

$$f(x) = e^x \qquad f'(x) = e^x$$

Example: $$\left(e^{x^2}\right)' = e^{x^2} \cdot 2x$$

Chain rule: $$(f \circ g)'(x) = f'(g(x)) \cdot g'(x)$$

Hint: $$\frac{1}{1+e^{-\beta x}} = \left(1+e^{-\beta x}\right)^{-1}$$

The derivative of the sigmoid

UNIVERSITY OF LEEDS

The sigmoid function: $$h(x)=\frac{1}{1+e^{-\beta x}}\equiv\sigma_\beta$$

We derive the most external function first

Then this

$$\sigma_\beta{}'(x)=\left(\left(1+e^{-\beta x}\right)^{-1}\right)'=-1\left(1+e^{-\beta x}\right)^{-2}\cdot\left(1+e^{-\beta x}\right)'=$$

$$=-1\left(1+e^{-\beta x}\right)^{-2}\cdot e^{-\beta x}\cdot\left(-\beta x\right)'=-1\left(1+e^{-\beta x}\right)^{-2}\cdot e^{-\beta x}\cdot\left(-\beta\right)$$

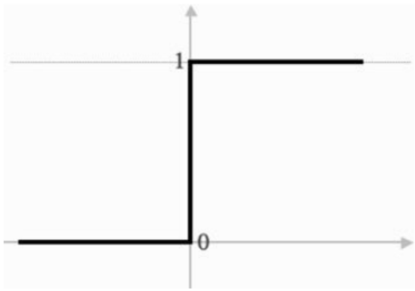
and finally this one

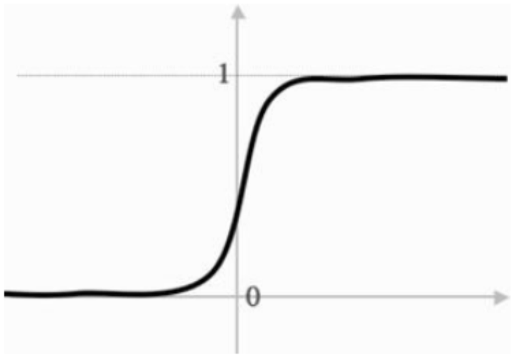
$$\sigma_\beta{}'(x)=-1\left(1+e^{-\beta x}\right)^{-2}\cdot e^{-\beta x}\cdot(-\beta)=\frac{\beta e^{-\beta x}}{\left(1+e^{-\beta x}\right)^2}$$

Let's note that:

$$1-\sigma_\beta=1-\frac{1}{1+e^{-\beta x}}=\frac{1+e^{-\beta x}-1}{1+e^{-\beta x}}=\frac{e^{-\beta x}}{1+e^{-\beta x}} \qquad \Rightarrow\sigma_\beta{}'=\beta\sigma_\beta\left(1-\sigma_\beta\right)$$

A different activation function





before

$$h(\boldsymbol{w}^T\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad \boldsymbol{w}^T\boldsymbol{x} > 0 \\ 0 & \text{if} \quad \boldsymbol{w}^T\boldsymbol{x} \leq 0 \end{cases}$$

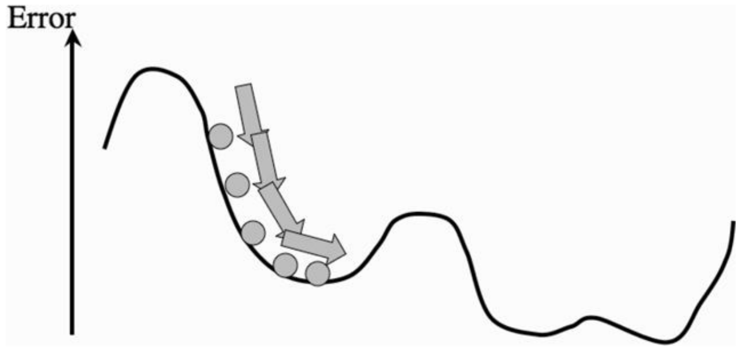
after

$$h(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+e^{-\beta \boldsymbol{w}^T\boldsymbol{x}}}$$

$$\sigma_\beta'(x) = \beta \frac{e^{-\beta x}}{(1+e^{-\beta x})^2} = \beta\,\sigma_\beta(x)(1-\sigma_\beta(x))$$

Gradient descent (again)

UNIVERSITY OF LEEDS



$$w_{t+1} = w_t - \eta \nabla E(x)$$

Perceptron

$$E_p(X) = \sum_{x_n \in X} w^t x_n (y_n - t_n)$$

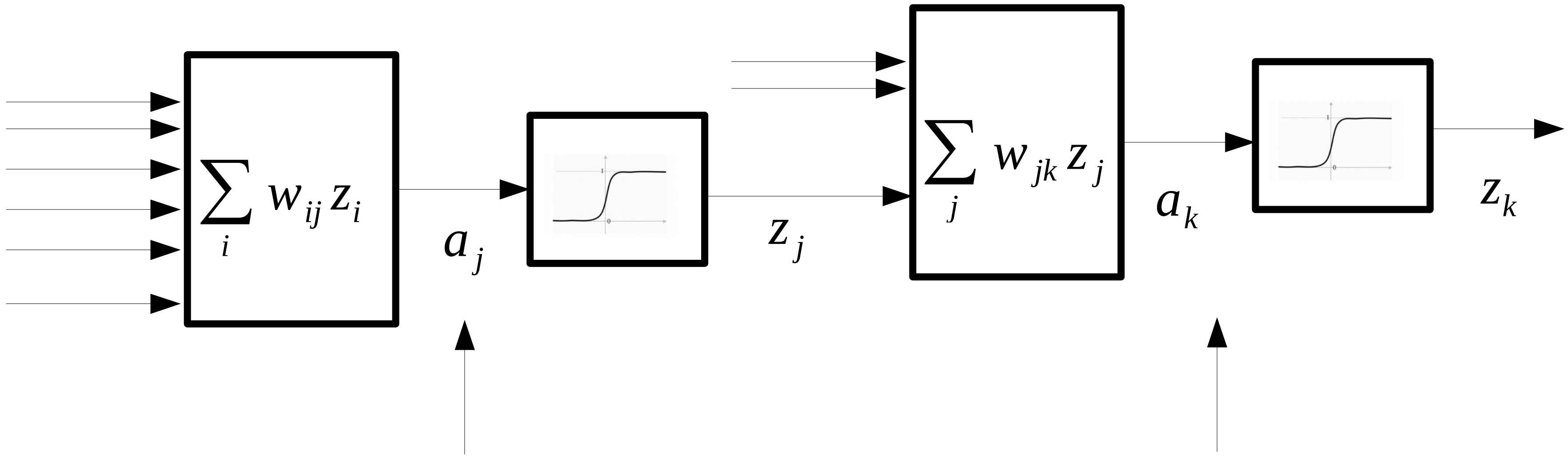
$$w_{t+1} = w_t - \eta (y - t) x$$

Multi-Layer P

$$E_m(X) = \frac{1}{2} \sum_{x_n \in X} (y_n - t_n)^2$$

?

Backpropagation of errors, notation



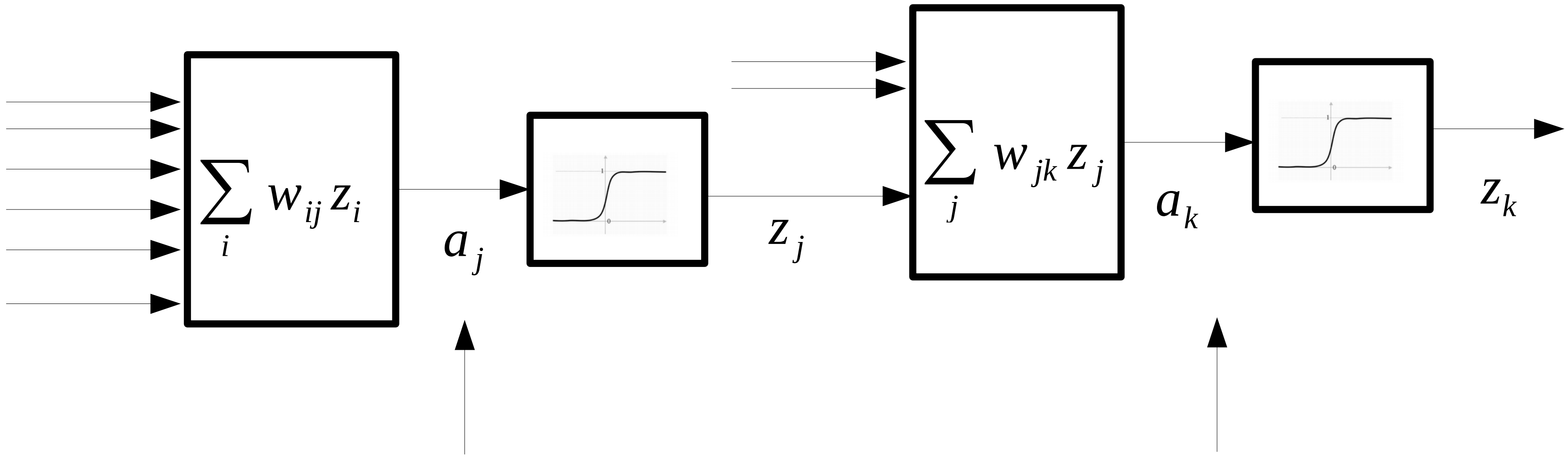
Neuron j is a **hidden** neuron

Neuron k is an **output** neuron

...

$$a_j = \sum_{i=1}^{N} w_{ij} z_i \qquad z_j = h(a_j) \qquad a_k = \sum_{j=1}^{M} w_{jk} z_j \qquad z_k = h(a_k)$$

Forward pass



Neuron j is a **hidden** neuron

Neuron k is an **output** neuron

$$\ldots \qquad a_j = \sum_{i=1}^{N} w_{ij} z_i \qquad z_j = h(a_j) \qquad a_k = \sum_{j=1}^{M} w_{jk} z_j \qquad z_k = h(a_k)$$

Forward pass: compute all the z

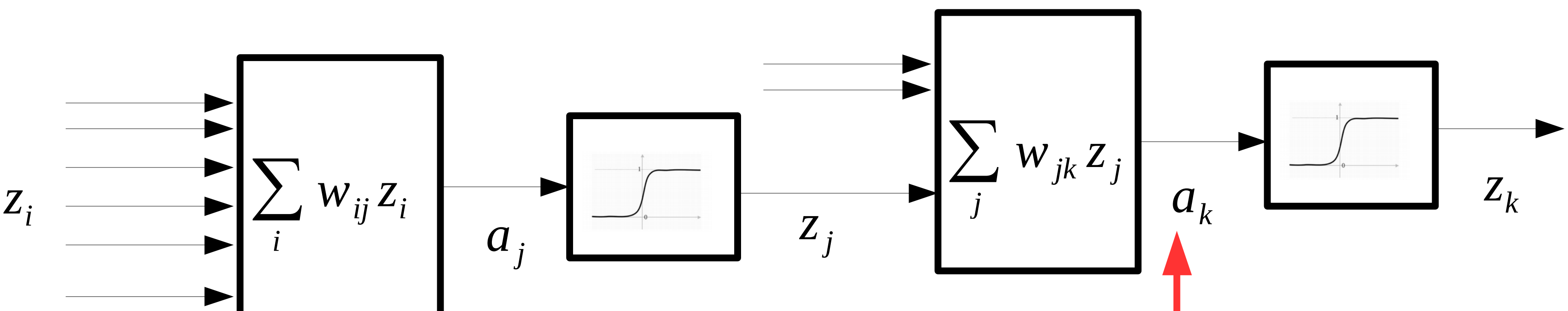
Backward pass, output neuron

How does a_k affect the error?

$$E(\mathbf{x}) = \frac{1}{2}(y-t)^2 = \frac{1}{2}\sum_k (z_k - t_k)^2$$

$$\frac{\partial E}{\partial a_k} = \frac{\partial}{\partial a_k}\frac{1}{2}(z_k - t_k)^2 = \frac{\partial}{\partial a_k}\frac{1}{2}(\sigma(a_k) - t_k)^2 = (\sigma(a_k) - t_k)\,\sigma(a_k)(1-\sigma(a_k))$$

Now this useful, because it cancels out the exponent in the derivation



We call this derivative δ: $\delta_k = (z_k - t_k)\, z_k (1 - z_k)$

Backward pass

Backward pass, output neuron

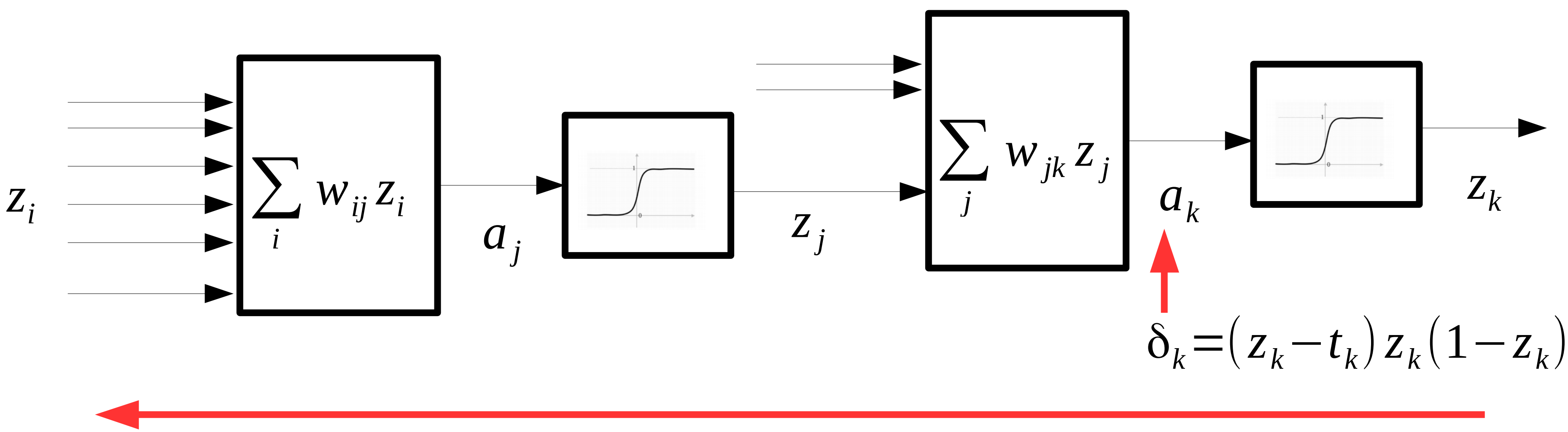
UNIVERSITY OF LEEDS

One step backward, inside the box: how does w_{jk} affect the error?

$$a_k = \sum_j w_{jk} z_j$$

We apply the chain rule again: $$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial w_{jk}} = \delta_k \cdot ?$$

$$\frac{\partial a_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} w_{0k} z_0 + w_{1k} z_1 + w_{2k} z_2 + \cdots + w_{jk} z_j = ?$$

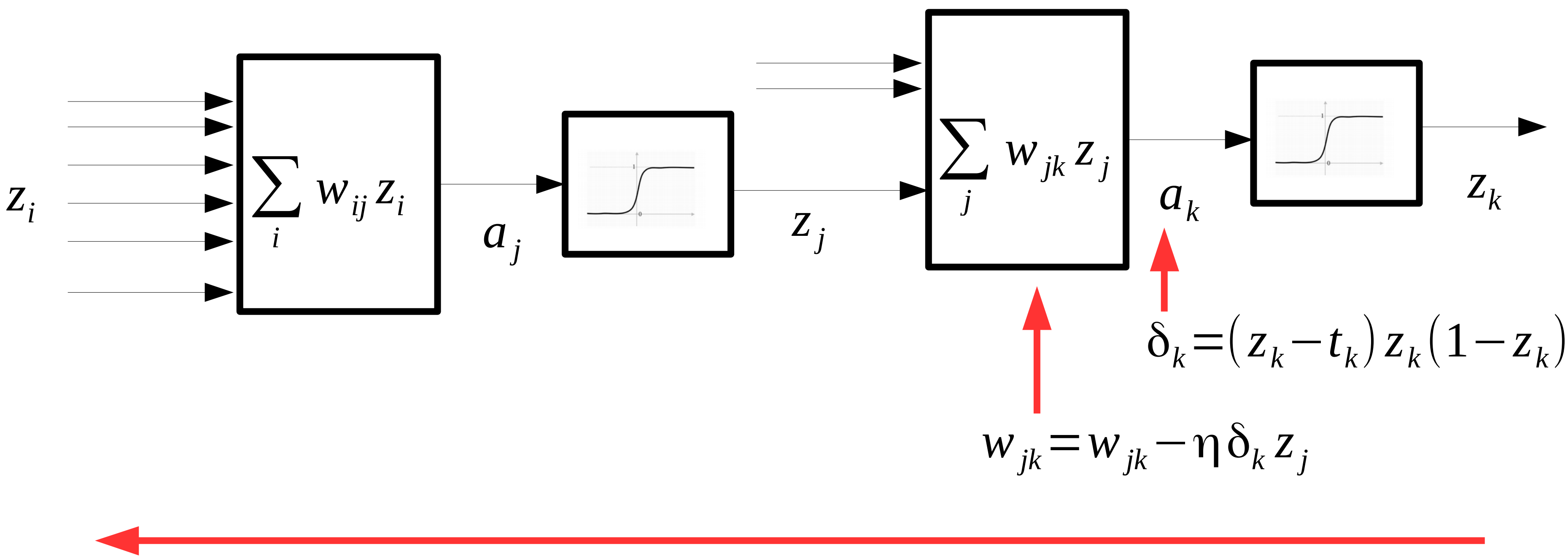


Backward pass, output neuron

One step backward, inside the box: how does w_{jk} affect the error?

We apply the chain rule again: $$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial w_{jk}} = \delta_k z_j$$

$$\frac{\partial a_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} w_{0k} z_0 + w_{1k} z_1 + w_{2k} z_2 + \cdots + w_{jk} z_j = z_j$$

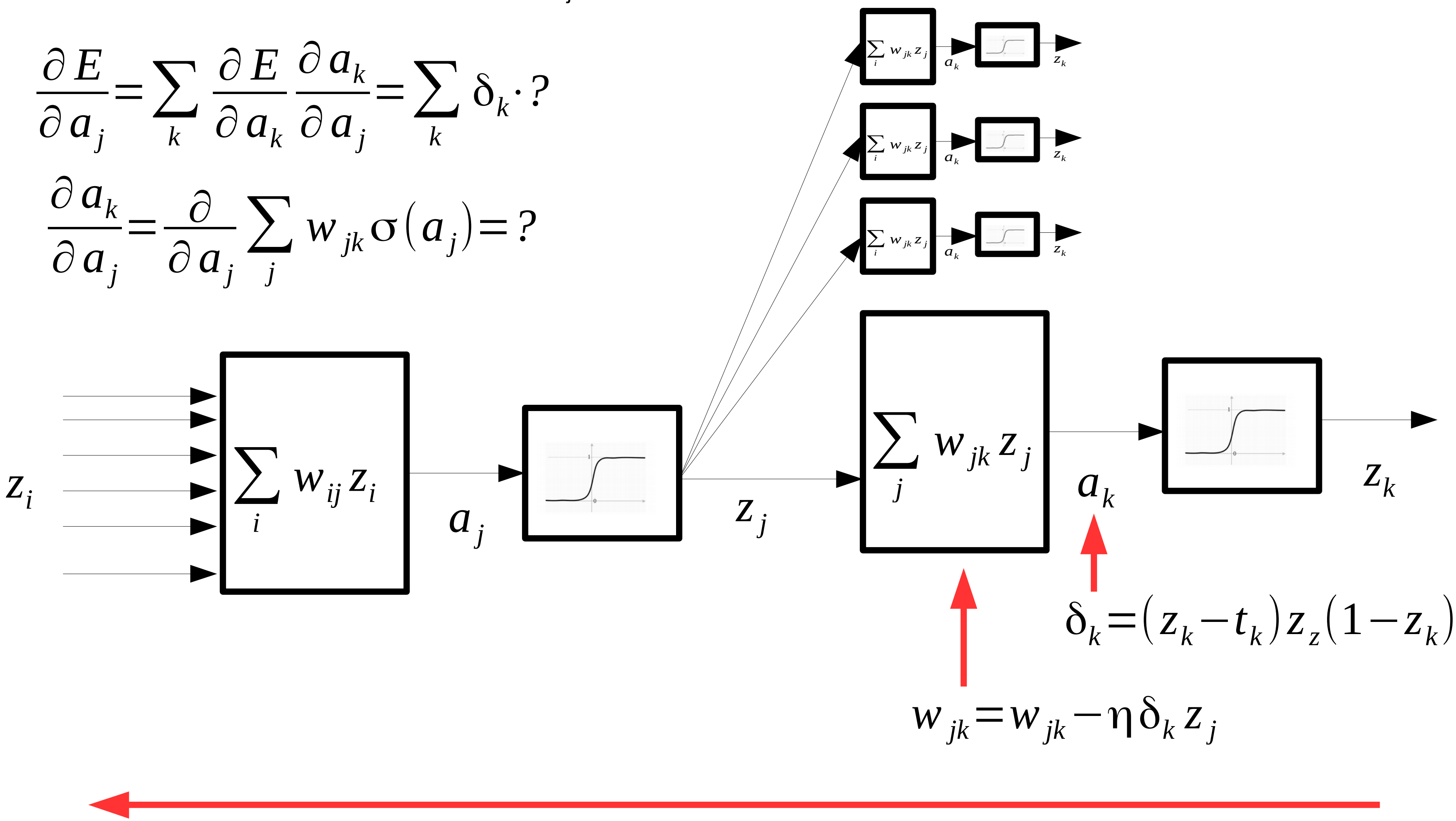


Backward pass, hidden neuron

One step backward: how does a_j affect the error?

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \cdot ?$$

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial}{\partial a_j} \sum_j w_{jk} \sigma(a_j) = ?$$



$$\delta_k = (z_k - t_k) z_z (1 - z_k)$$

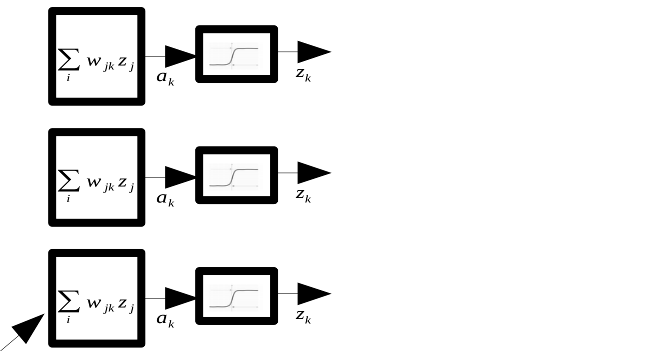
$$w_{jk} = w_{jk} - \eta \delta_k z_j$$

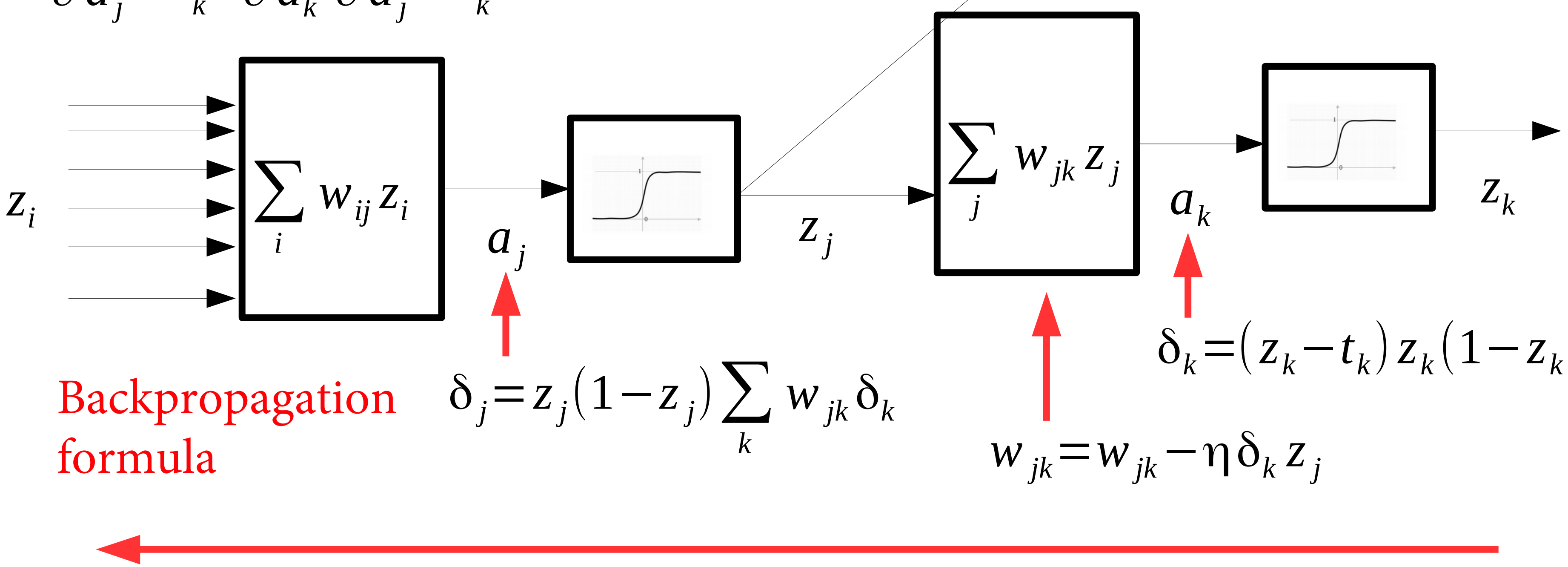
Backward pass, hidden neuron

One step backward: how does a_j affect the error?

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial}{\partial a_j} \sum_l w_{lk} \sigma(a_l) = w_{jk} \sigma(a_j)(1-\sigma(a_j))$$

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k w_{jk} z_j (1-z_j)$$





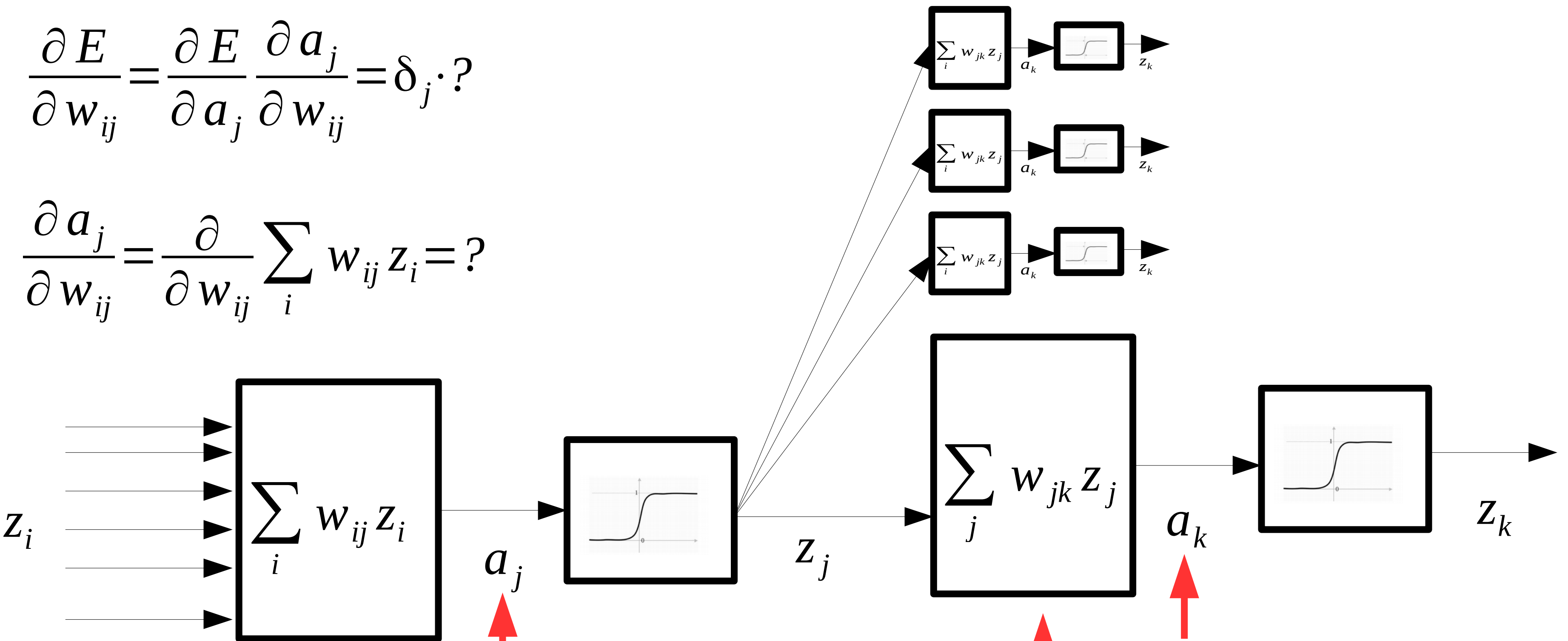
Computing delta

UNIVERSITY OF LEEDS

One step backward, inside the box: how does w_{ij} affect the error?

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \delta_j \cdot ?$$

$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_i w_{ij} z_i = ?$$



$$\delta_j = z_j (1 - z_j) \sum_k w_{jk} \delta_k$$

$$w_{jk} = w_{jk} - \eta \delta_k z_j$$

$$\delta_k = (z_k - t_k) z_k (1 - z_k)$$

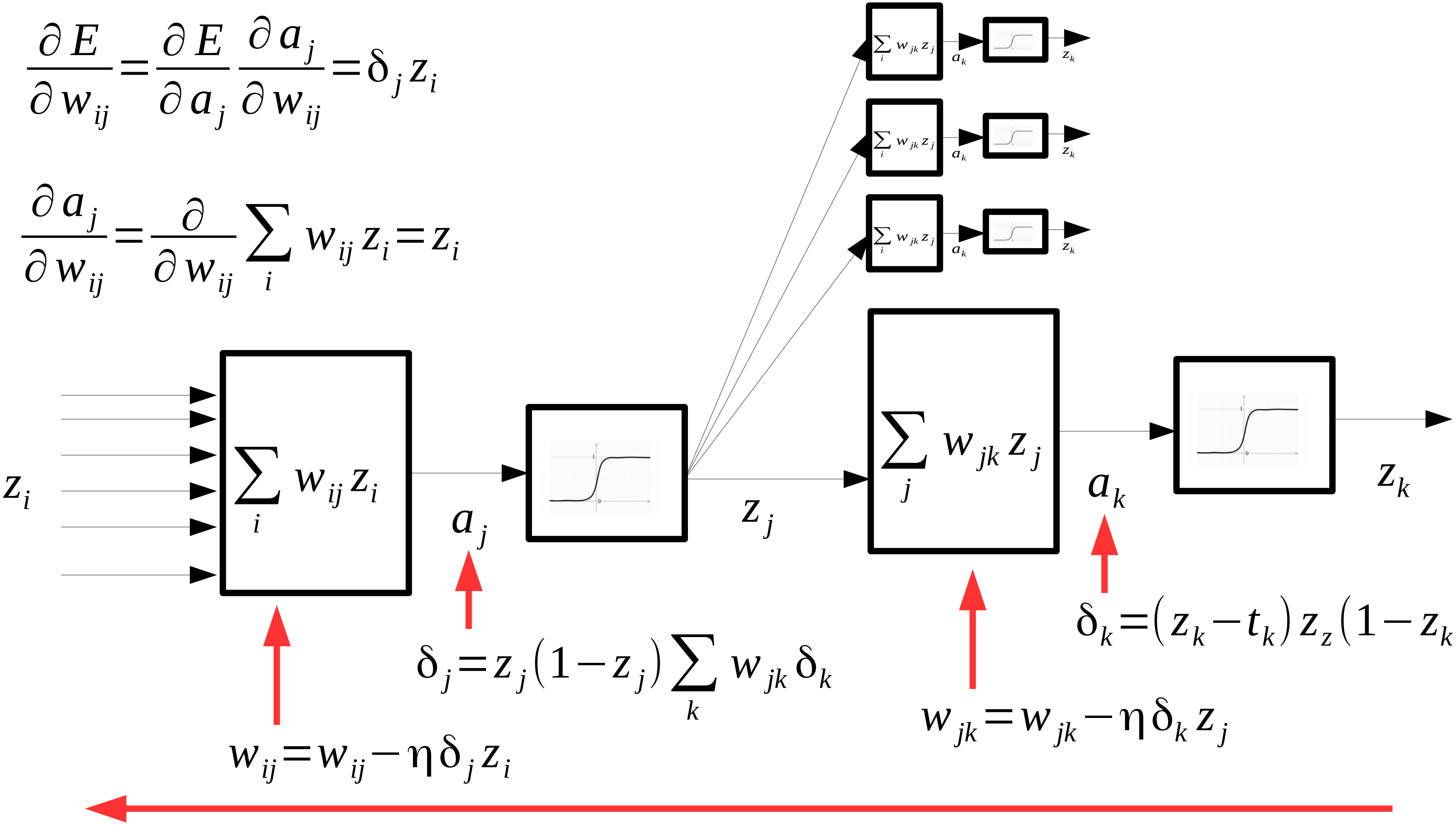
Computing delta

UNIVERSITY OF LEEDS

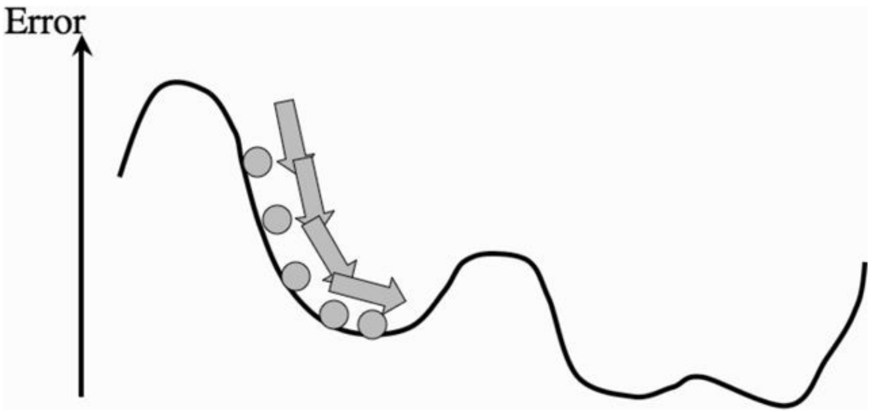
One step backward, inside the box: how does w_{ij} affect the error?

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \delta_j z_i$$

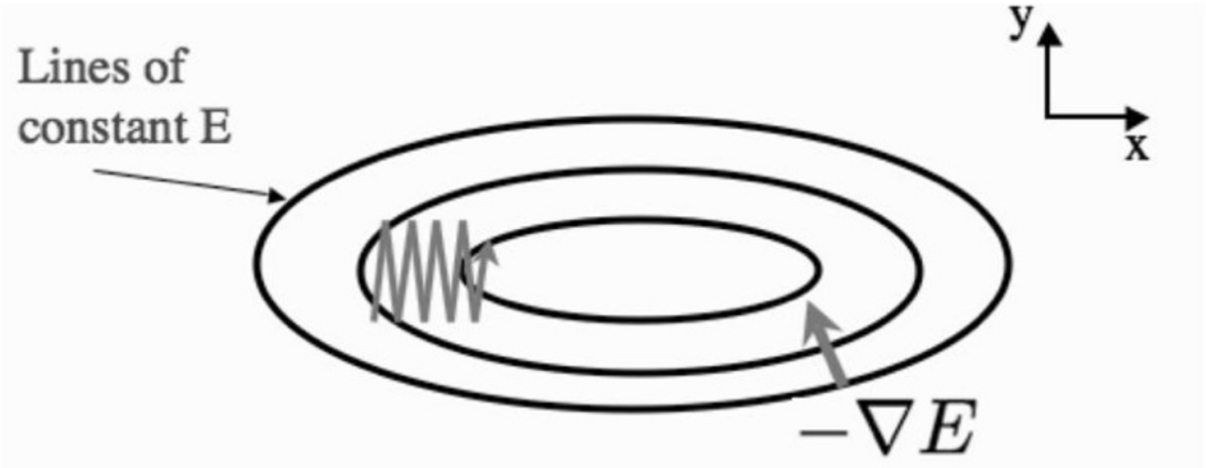
$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\sum_i w_{ij} z_i = z_i$$



Local Minima



Start with weights close to 0: where the decision is actually made

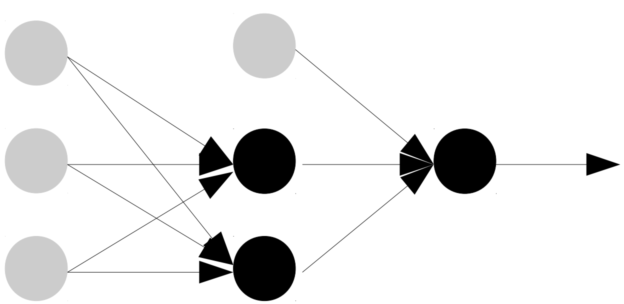


Multiple random restarts

Using MLPs

UNIVERSITY OF LEEDS

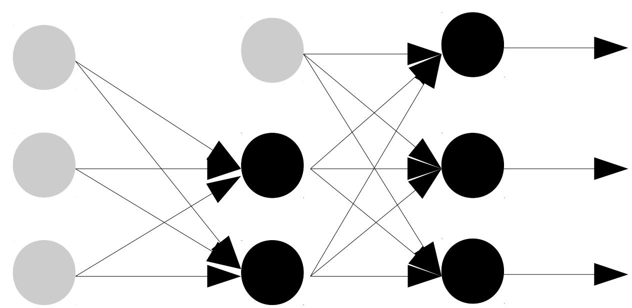
Regression



$$h(a)=a$$

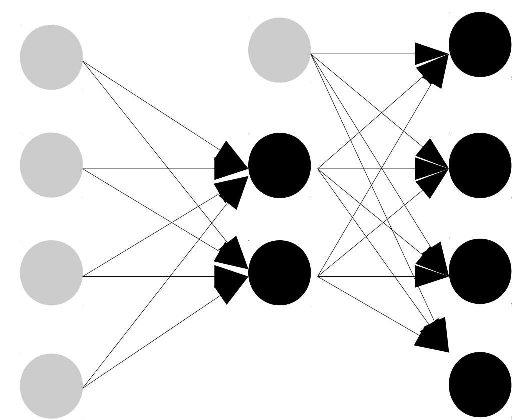
Last neuron linear

Classification



One output per class, pick highest

Compression



Middle "bottleneck" layer

Training "recipe"

UNIVERSITY OF LEEDS

Choose features

Normalize (rescale) data: $x' = \frac{x - \bar{x}}{\sigma}$ or $x' = \frac{x - min(x)}{max(x) - min(x)}$

Zero mean, unit variance

in [0,1]

Create training, validation, and test sets

Decide whether you need hidden layers and how big.
Try several ones.

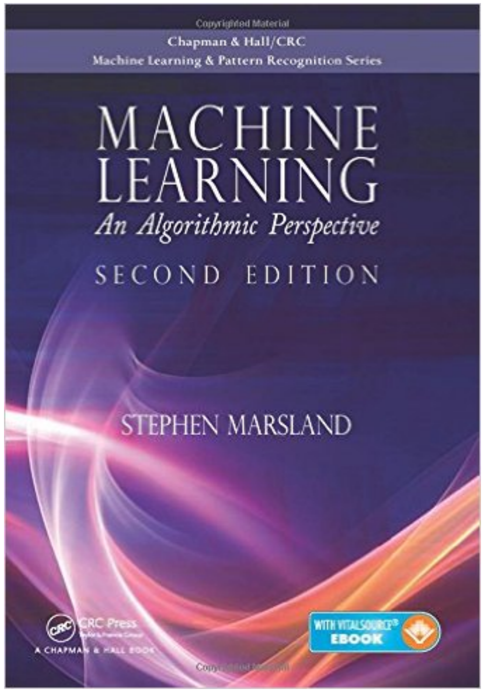
Train

Test

Conclusion

Learning outcomes

- Construct a multi-layer neural network that classifies a given dataset in 2D, overcoming the limitation on learning separability.
- Define an appropriate error to minimise for Feed-forward neural networks.
- Derive the update rule of the weights of the NN, through backpropagation.
- Apply NNs to real-world data sets


Copyrighted Material
Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series
MACHINE LEARNING
An Algorithmic Perspective
SECOND EDITION
STEPHEN MARSLAND
CRC Press
A CHAPMAN & HALL BOOK
WITH VITALSOURCE® EBOOK
Copyrighted Material

Chapter 4