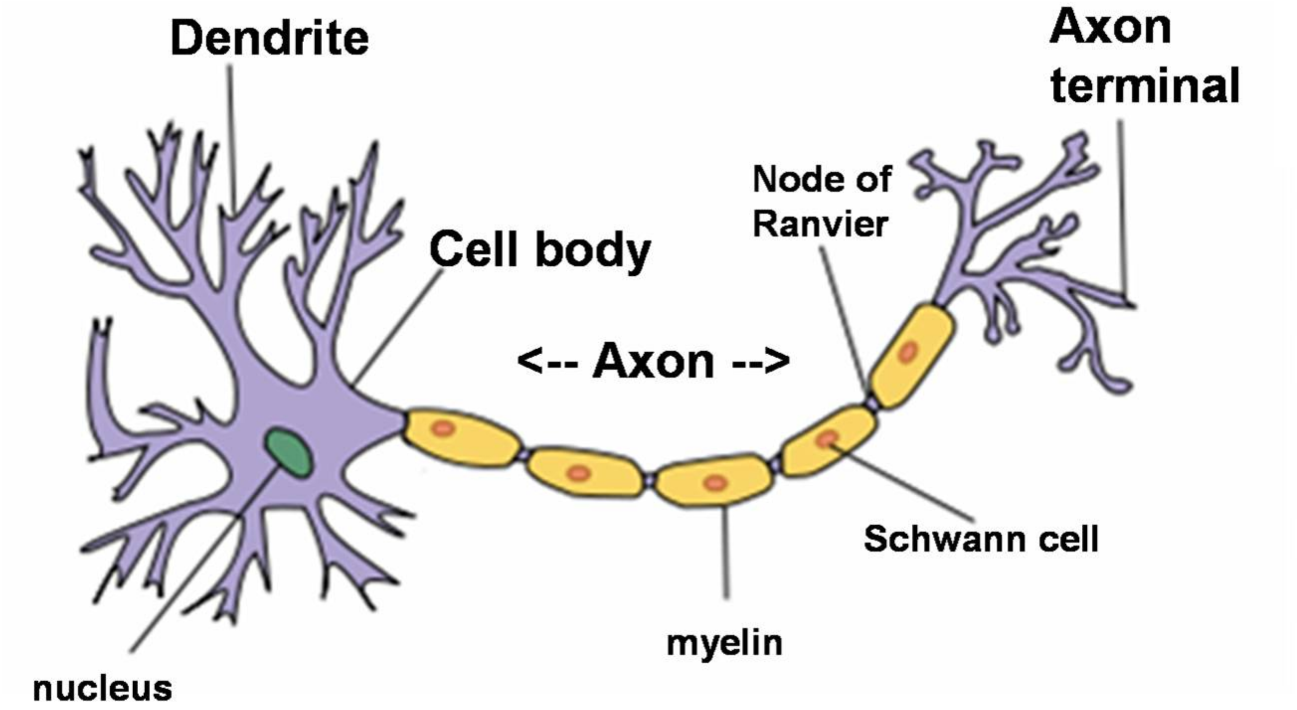
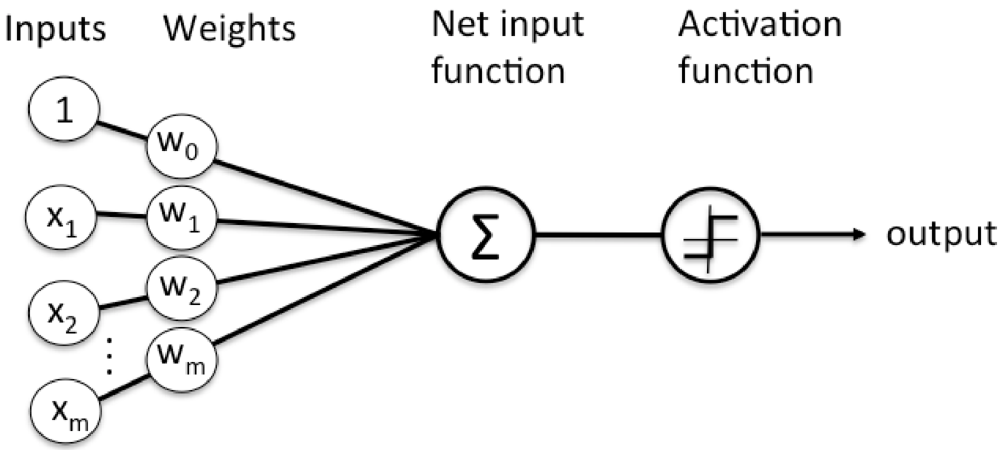
Perceptron & Multi-layer Neural Network

CS385 Machine Learning – Artificial Neural Network

Neuron - perceptron





$$In = W_0 + W_1 x_1 + W_2 x_2 + \dots + W_m x_m$$

$$out = g(In)$$

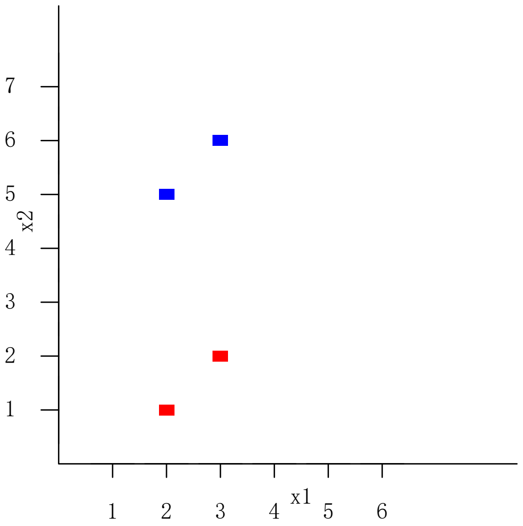
→ { Hard Threshold
  sigmoid

Perceptrons

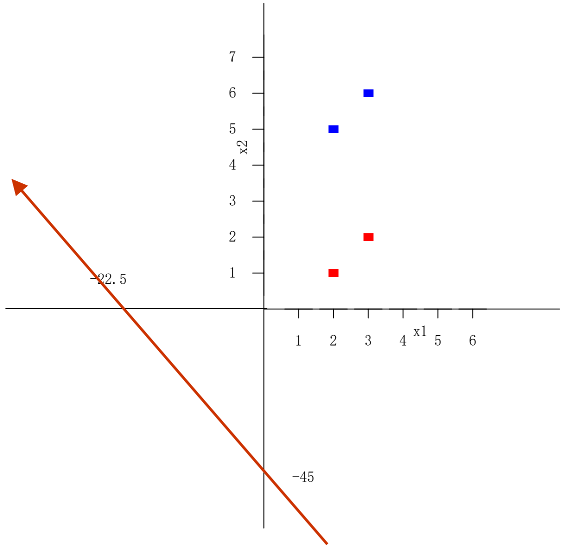
- The first generation of neural networks
- They were popularised by Frank Rosenblatt in the early 1960's.
  - They appeared to have a very powerful learning algorithm.
  - Lots of grand claims were made for what they could learn to do.
- In 1969, Minsky and Papert published a book called "Perceptrons" that analysed what they could do and showed their limitations.
  - Many people thought these limitations applied to all neural network models.
- The perceptron learning procedure is still widely used today for tasks with enormous feature vectors that contain many millions of features.

Perceptrons: training

- Same as the learning method of logistic regression (with hard threshold or sigmoid function)
- Pick training cases using any policy that ensures that every training case will keep getting picked.
  - If the output unit is correct, leave its weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector.
- This is guaranteed to find a set of weights that gets the right answer for all the training cases if any such set exists.

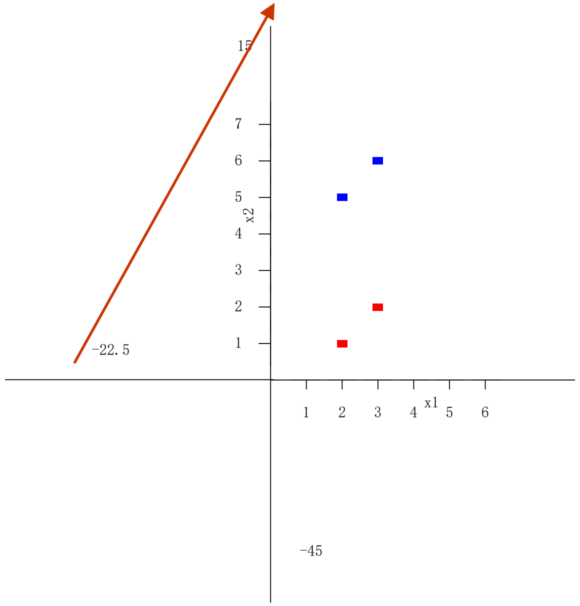


W= (0,0,0)



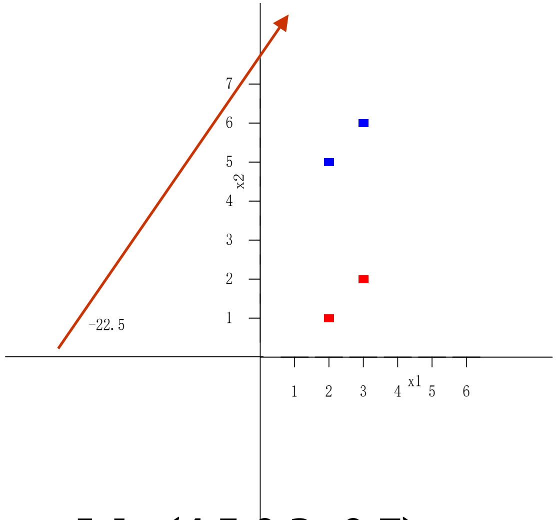
W= (4.5,0.2,0.1)

$0.2 \times x_1 + 0.1 \times x_2 + 4.5 = 0$



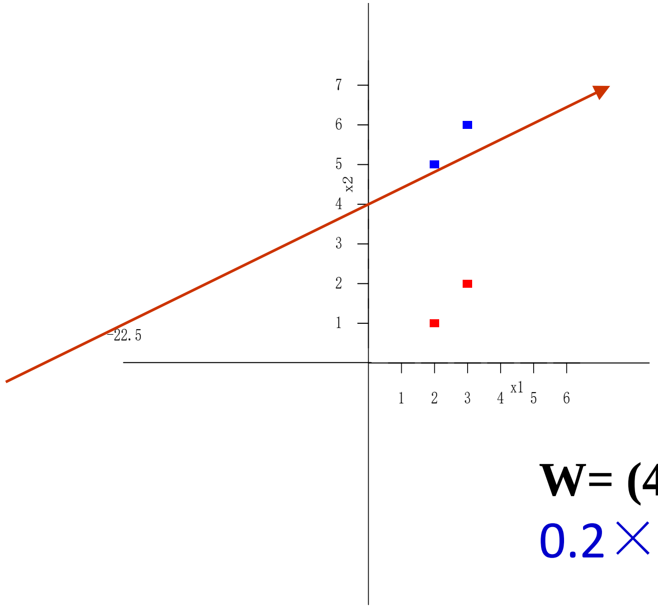
W= (4.5,0.2,-0.3)

$0.2 \times x_1 - 0.3 \times x_2 + 4.5 = 0$



W= (4.5,0.2,-0.7)

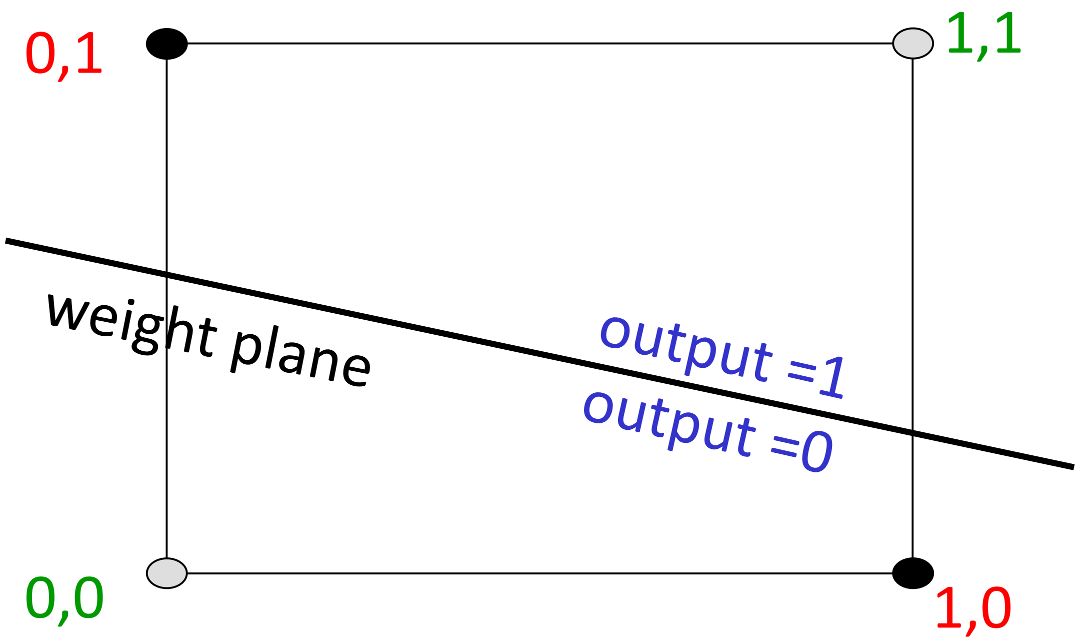
$0.2 \times x_1 - 0.7 \times x_2 + 4.5 = 0$



W= (4.5,0.2,-1.1)

$0.2 \times x_1 - 1.1 \times x_2 + 4.5 = 0$

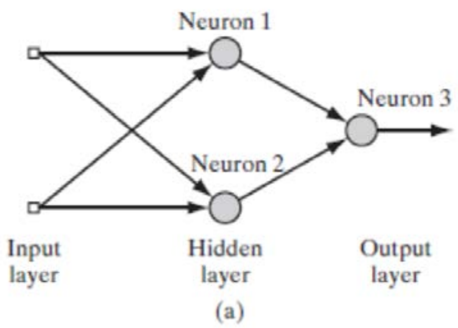
What perceptrons can't do



The positive and negative cases cannot be separated by a plane

Hidden units

- Without hidden units are very limited
  - More layers of linear units do not help. Its still linear.
- We need multiple layers of adaptive, non-linear hidden units.

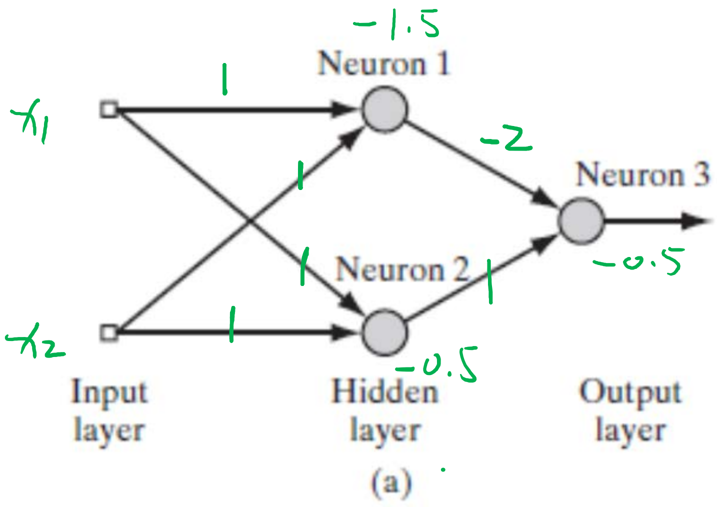


Neuron1 $w_{11} = w_{12} = +1 \quad b_1 = -\frac{3}{2}$

Neuron2 $w_{21} = w_{22} = +1 \quad b_2 = -\frac{1}{2}$

Neuron3 $w_{31} = -2 \quad w_{32} = +1 \quad b_3 = -\frac{1}{2}$

N.N. Inference



$x_1 = 0 \quad x_2 = 0$:

Neuron1: $0 \times 1 + 0 \times 1 + (-1.5) < 0 \rightarrow 0$

Neuron2: $0 \times 1 + 0 \times 1 + (-0.5) < 0 \rightarrow 0$

Neuron3: $0 \times -2 + 0 \times 1 + (-0.5) < 0 \rightarrow 0$

$x_1 = 1, \quad x_2 = 1$:

Neuron1: $1 \times 1 + 1 \times 1 + (-1.5) > 0 \rightarrow 1$

Neuron2: $1 \times 1 + 1 \times 1 + (-0.5) > 0 \rightarrow 1$

Neuron3: $1 \times -2 + 1 \times 1 + (-0.5) < 0 \rightarrow 0$

$x_1 = 0, \quad x_2 = 1$:

Neuron1: $0 \times 1 + 1 \times 1 + -1.5 < 0 \rightarrow 0$

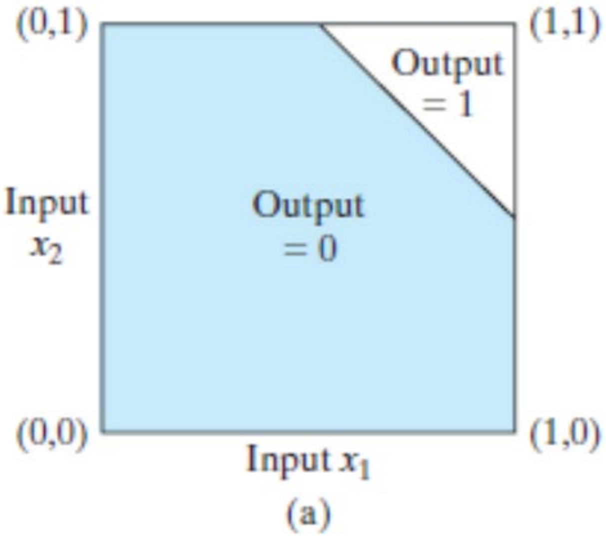
Neuron2: $0 \times 1 + 1 \times 1 - 0.5 > 0 \rightarrow 1$

Neuron3: $0 \times -2 + 1 \times 1 - 0.5 > 0 \rightarrow 1$

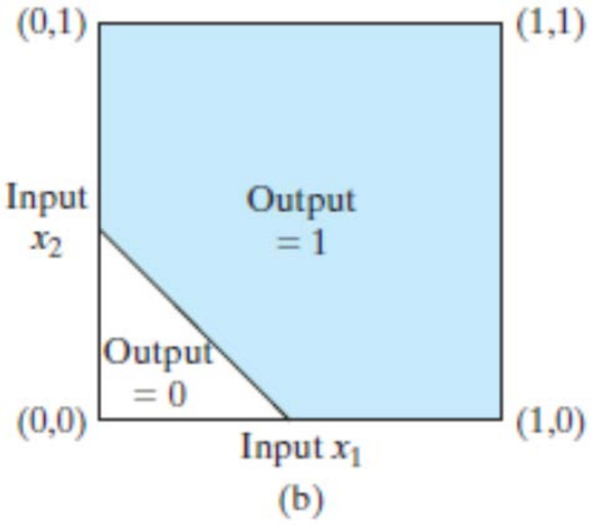
$x_1 = 1, \quad x_2 = 0$:

...

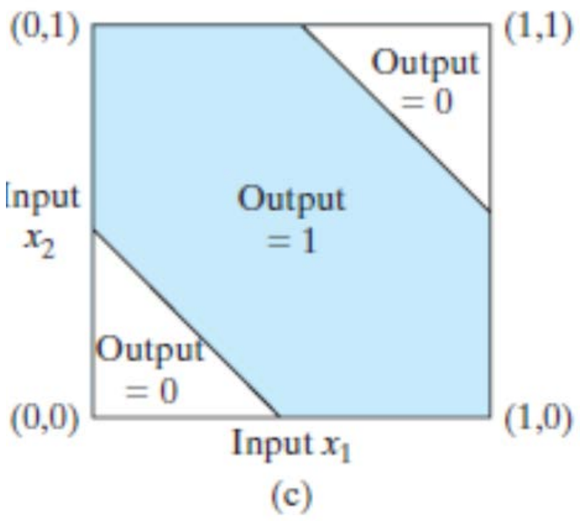
XOR Problem



Neuron1



Neuron2

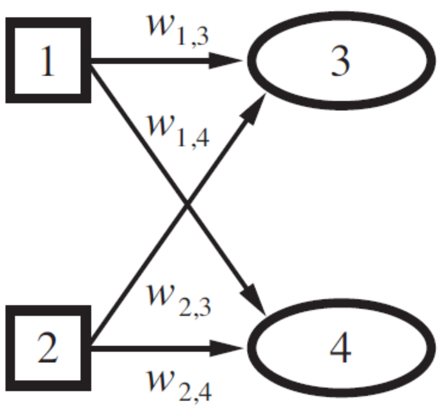


Neuron3

Loss function

- how can we train such nets?
  - We need an efficient way of adapting all the weights, not just the last layer. This is hard.
  - Learning the weights going into hidden units is equivalent to learning features.
  - This is difficult because nobody is telling us directly what the hidden units should do.

Learning - single layered



x_1 x_2	y_1 y_2

output can be a vector

For a single sample (x1,x2)

output Nodes

$$a1 = g\left(w_{1,3} * x_1 + w_{2,3} * x_2 + w_{0,3}\right)$$
$$a2 = g\left(w_{1,4} * x_1 + w_{2,4} * x_2 + w_{0,4}\right)$$

$$E(\mathbf{w}) = \frac{1}{2}[(y_1 - a_1)^2 + (y_2 - a_2)^2]$$

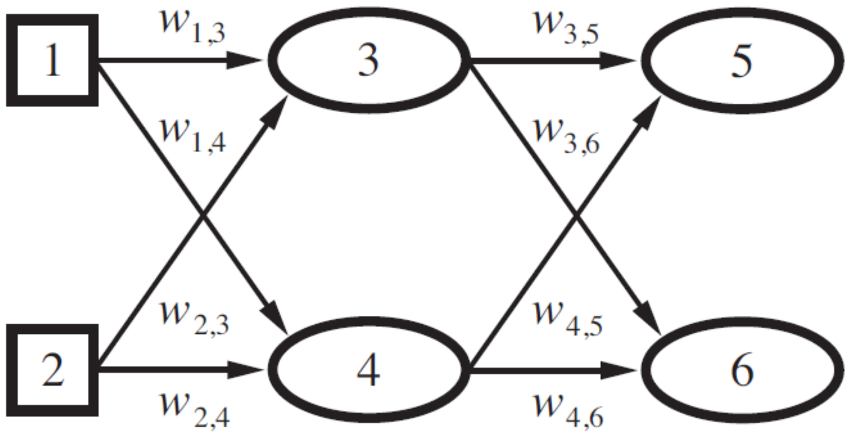
$$\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w1}} \frac{1}{2}(y_1 - a_1)^2 + \frac{\partial}{\partial \mathbf{w2}} \frac{1}{2}(y_2 - a_2)^2$$

$(W_{13}, W_{23}, W_{03}, W_{14}, W_{24}, W_{04})$

(W_{13}, W_{23}, W_{03})

$(W_{14}\ W_{24}\ W_{04})$

Difficulty in learning with multilayer



Error at the output layer is clear,

Error at the hidden layers seems mysterious

- the training data do not say what value the hidden nodes should have.

Error Back-propagate

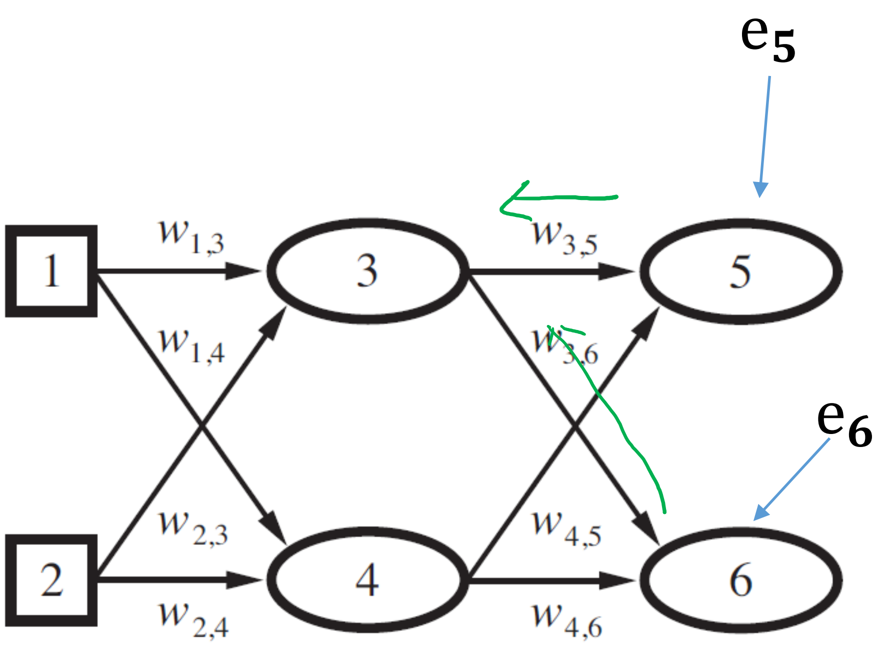
Error of Node3

$$e_3 = \frac{w_{3,5}}{w_{3,5} + w_{4,5}} e_5 + \frac{w_{3,6}}{w_{3,6} + w_{4,6}} e_6$$

Now Error at the hidden layers can be estimated

Error of e_5 from Node3 and Node4 via $w_{3,5}$, $w_{4,5}$

$$e_3 = w_{3,5} e_5 + w_{3,6} e_6$$



Part of e_5

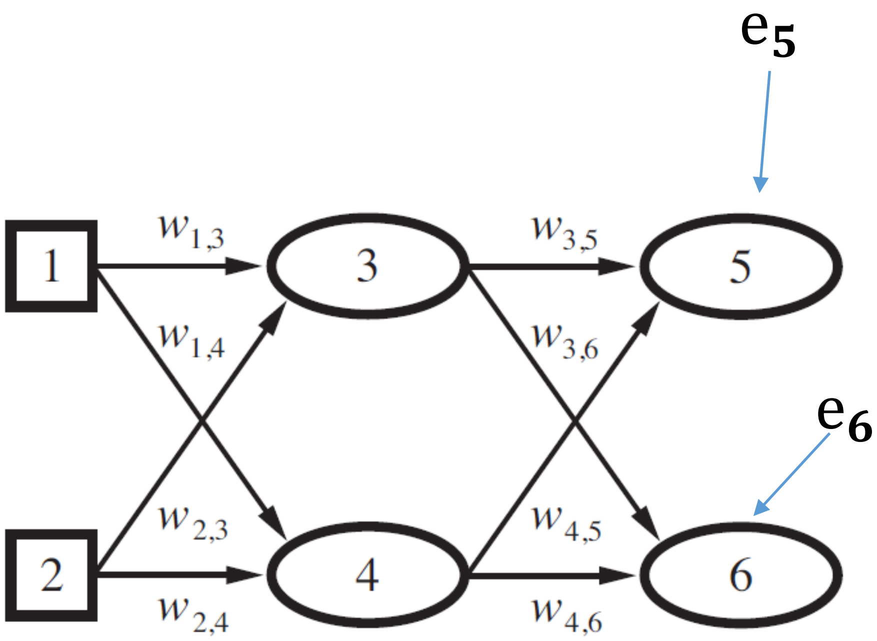
Output layer weight learning – chain rule

$$e_5 = \frac{1}{2}(y_1 - a_1)^{\,2}$$

$$a_1 = sigmoid(in_5)$$

Now Error at the hidden layers can be estimated

$$in_5 = w_{3,5} * o_3 + w_{4,5} * o_4 + w_{0,5}$$

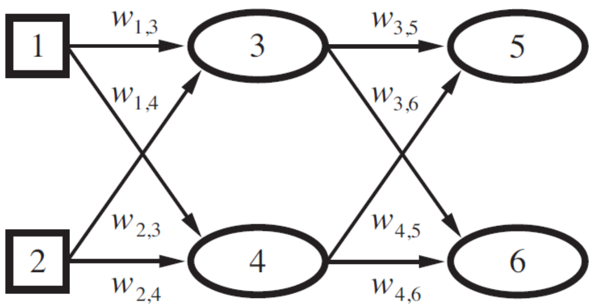


$$\frac{\partial e_5}{\partial w_{3,5}} = \frac{\partial e_5}{\partial a_1} * \frac{\partial a_1}{\partial in_5} * \frac{\partial in_5}{\partial w_{3,5}}$$

$$= -(y_1 - a_1) * \underbrace{sigmoid(in_5)}_{o_5} * (1 - \underbrace{sigmoid(in_5)}_{o_5}) * o_3$$

$$sigmoid(x)' = sigmoid(x) * (1 - sigmoid(x))$$

Output layer weight learning – delta rule



e5

$$\frac{\partial e_5}{\partial w_{3,5}} = -(y_1 - a_1) * sigmoid(in_5) * (1 - sigmoid(in_5)) * o_3$$

$$= \underbrace{-(y_1 - o_5) * o_5 * (1 - o_5)}_{\delta_5} * o_3$$

$o_5 = a_1$

$o_6 = a_2$

$$\frac{\partial e_5}{\partial w_{4,5}} = \delta_5 * o_4$$

$$\frac{\partial e_5}{\partial w_{0,5}} = \delta_5$$

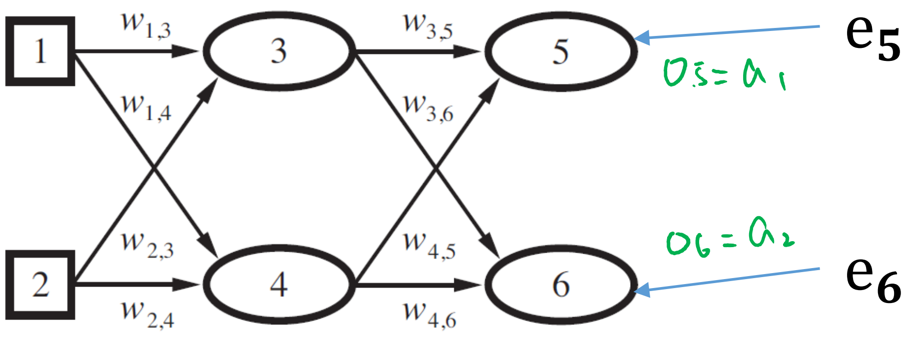
Gradient Descent

$$w_{3,5} = w_{3,5} - \alpha * \delta_5 * o_3$$

$$w_{4,5} = w_{4,5} - \alpha * \delta_5 * o_4$$

$$w_{0,5} = w_{0,5} - \alpha * \delta_5$$

Hidden layer weight learning



$$e_3 = e_5 + e_6$$

$$\frac{\partial e_3}{\partial w_{1,3}} = \boxed{\frac{\partial e_5}{\partial w_{1,3}}} + \frac{\partial e_6}{\partial w_{1,3}}$$

$$e_5 = \frac{1}{2}(y_1 - a_1)^2$$

$$a_1 = \text{sigmoid}(in_5)$$

$$in_5 = W_{35} O_3 + W_{45} O_4 + W_{05}$$

$$O_3 = \text{sigmoid}(in_3)$$

$$in_3 = W_{13} X_1 + W_{23} X_2 + W_{03}$$

$$\frac{\partial e_5}{\partial w_{1,3}} = \delta_5 * w_{3,5} * o_3 * (1 - o_3) * x_1$$

$$\frac{\partial e_5}{\partial w_{2,3}} = \delta_5 * w_{3,5} * o_3 * (1 - o_3) * x_2$$

...

$$\frac{\partial e_5}{\partial W_{13}} = \frac{\partial e_5}{\partial a_1} \cdot \frac{\partial a_1}{\partial in_5} \cdot \frac{\partial in_5}{\partial O_3} \cdot \frac{\partial O_3}{\partial in_3} \cdot \frac{\partial in_3}{\partial W_{13}}$$

$$\delta_5 \quad = -(y_1 - a_1) \cdot O_5 \cdot (1 - O_5) \cdot W_{35} \cdot O_3 \cdot (1 - O_3) \cdot X_1$$