COMP 472: Artificial Intelligence Machine Learning Neural Networks

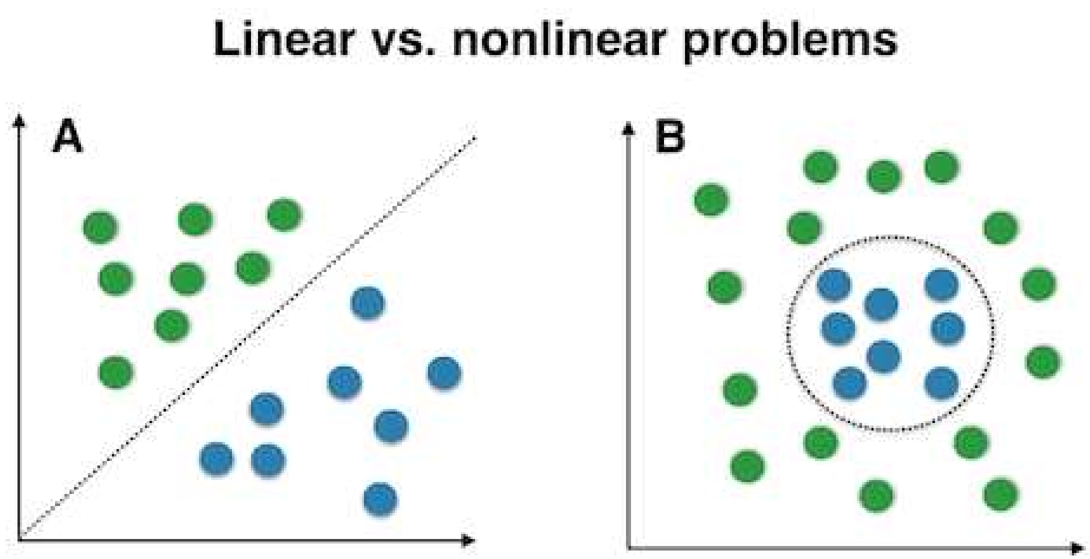
- Russell & Norvig: Sections 19.6, 21.1

Today

1. Introduction to ML
2. Naïve Bayes Classification
   a. Application to Spam Filtering
3. Decision Trees
4. (Evaluation
5. Unsupervised Learning)
6. Neural Networks
   a. Perceptrons
   b. Multi Layered Neural Networks

Limits of the Perceptron

- can only model linear decision boundaries
- but real-world problems cannot always be represented by linearly-separable functions…



http://sebastianraschka.com/Articles/2014_kernel_pca.html

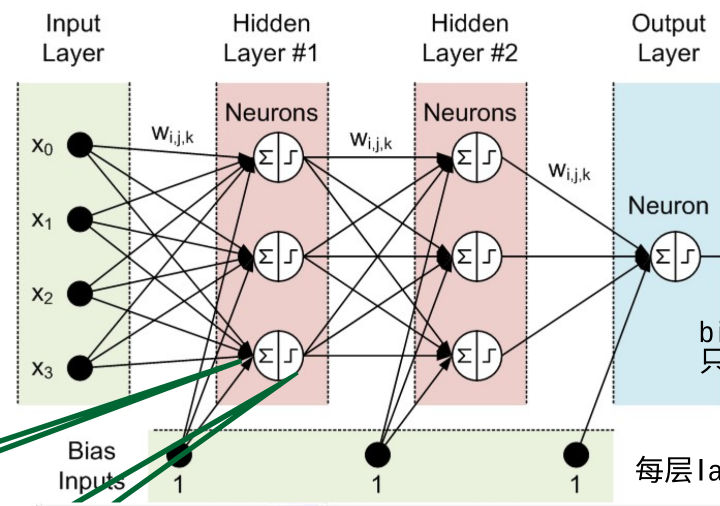
Multilayer Neural Networks

Perceptron(单个神经元)

- Solution:

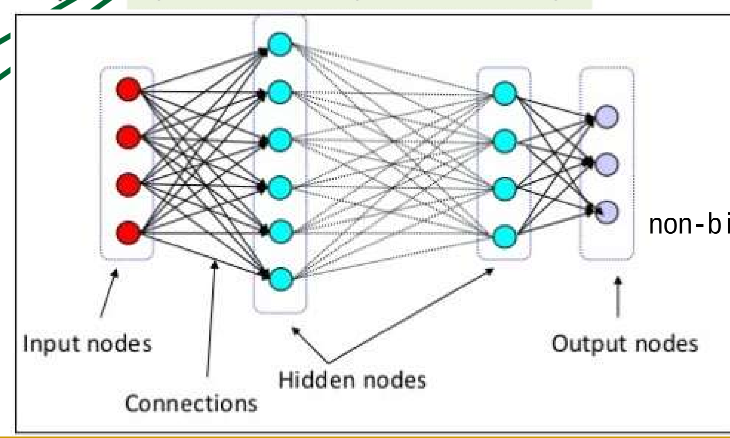
1. use a non-linear activation function
2. to learn more complex functions (more complex decision boundaries), have hidden nodes
3. and for non-binary decisions, have multiple output nodes

在perceptron的input layer与output layer间插入Hidden layers



binary decision 只有一个Output node

每层layer会给个bias



non-binary 3 classes

usual transfer function

Non-linear, differentiable activation function

http://www.kdnuggets.com/2016/10/deep-learning-key-terms-explained.html

Example

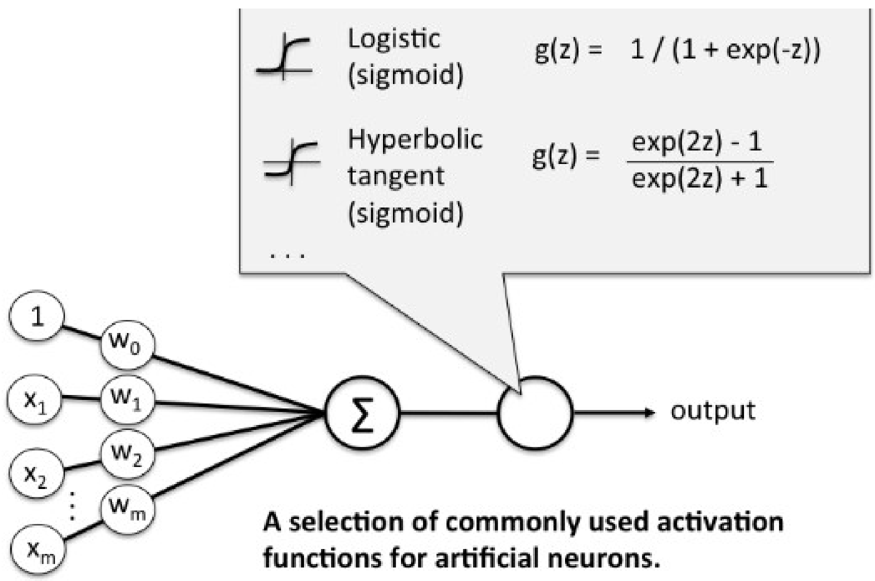
1

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	–1
2.5	2.1	1
8.0	7.7	–1
0.5	2.2	1
7.9	8.4	–1
7.0	7.0	–1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	–1

2

Activation Functions

- Backpropagation requires a differentiable activation function
- that returns continuous values within a range
  - eg. a value between 0 and 1 (instead of 0 or 1, like the perceptron)
- indicates how close/how far the output of the network is compared to the right answer

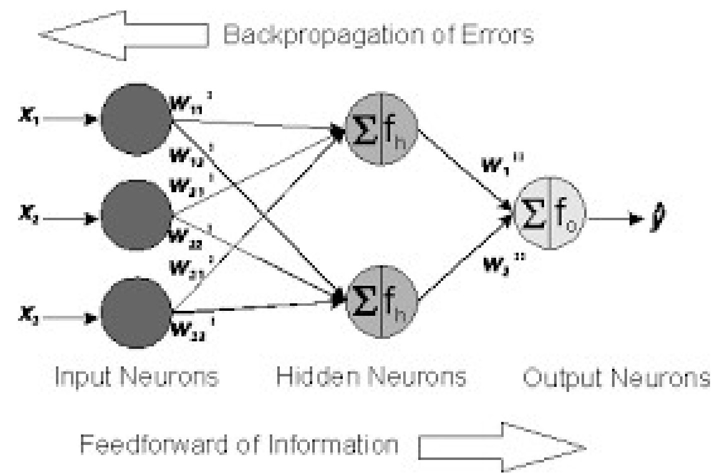


A selection of commonly used activation functions for artificial neurons.

https://medium.com/towards-data-science/activation-functions-and-its-types-which-is-better-a9a5310cc8f

Learning in a Neural Network

- Learning is the same as in a perceptron:



1. Feed-forward:
   - Input from the features is fed forward in the network from input layer towards the output layer
2. Backpropagation:
   - Error rate flows backwards from the output layer to the input layer (to adjust the weights in order to minimize the error)
3. Iterate until error rate is minimized
   - repeat the forward pass and back pass for the next data points until all data points are examined (1 epoch)
   - repeat this entire exercise (several epochs) until the overall error is minimised

Typical Cost Functions

- Error of the network is computed via a cost function

1. Quadratic Cost:
   - aka mean squared error (MSR)
   - minimize the difference between output values and target values

$$C = \frac{1}{n}\sum_{i=1}^{n}(T_i - O_i)^2$$

where n = nb of instances

   - used mostly for regression tasks

2. Cross-entropy:
   - used mostly for classification tasks
   - where we don't care about the exact value of the network output, we only care about the final class

$$C = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left(T_{ik}\,\log\left(O_{ik}\right)\right)$$

where:
n = nb of instances
K = nb of classes
log = base 2, base 10, ln, ….

Example of Cross Entropy

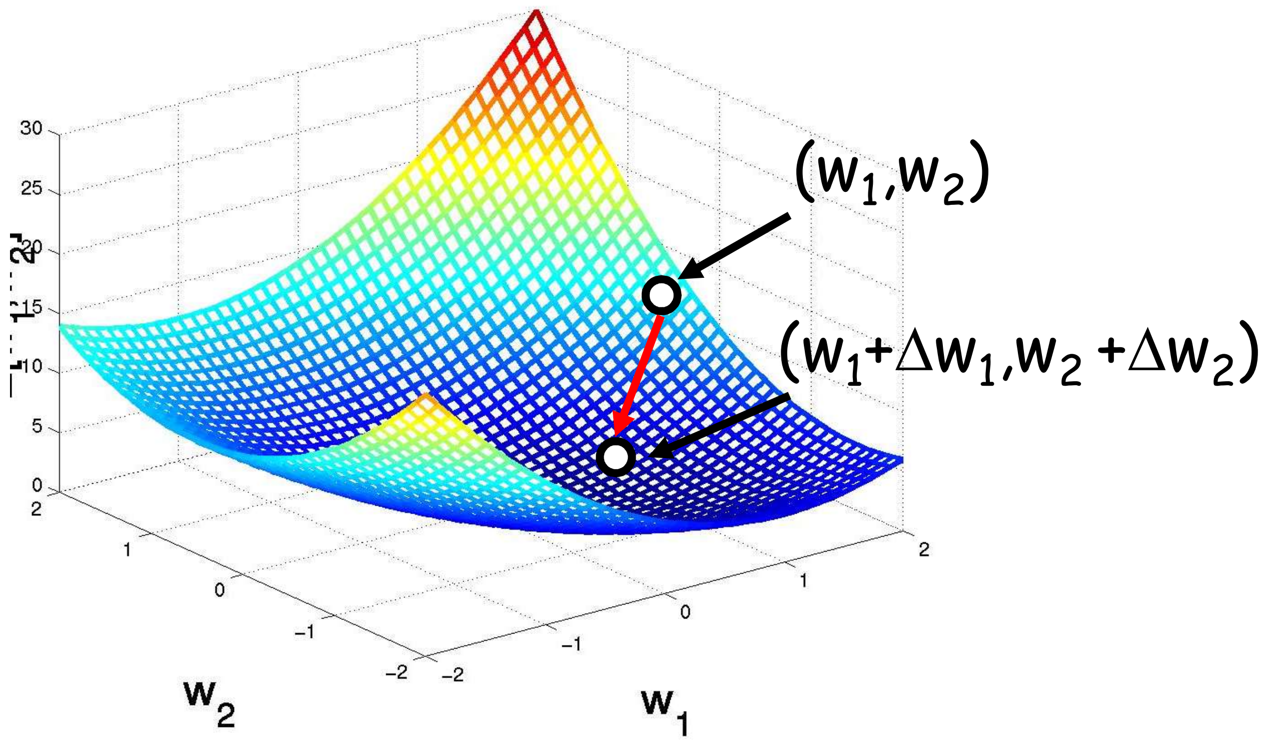
- Assume 3 classes: red, blue and green
  - i.e. $\mathrm{K} = 3$
- for a specific instance i, the target is green
  - ie. (red, blue, green) = (0, 0, 1) // real distribution
  - $T^1 = 0 \;\; T^2 = 0 \;\; T^3 = 1$
- but the model predicts the probabilities as:
  - (red, blue, green) = (0.1, 0.4, 0.5) // predicted distribution
  - $O^1 = 0.1 \;\; O^2 = 0.4 \;\; O^3 = 0.5$
- the cross entropy of two distributions (real and predicted)
  - $-\sum_{k=1}^{K} \left(T^k \ln\left(O^k\right)\right) = -((0)ln(0.1) + (0)ln(0.4) + (1)ln(0.5))$
- but we have several instances in the test set, so we will take the average of the cross-entropy across all instances i in the test set
  - $C = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}({T^k}_i \ln({O^k}_i))$
- see an example calculation computation in a few slides

Backpropagation

- In a multilayer network…
  - Computing the error in the output layer is clear.
  - Computing the error in the hidden layer is not clear, because we don't know what output it should be
- Intuitively:
  - A hidden node h is "responsible" for some fraction of the error in each of the output node to which it connects.
  - So the error values (δ):
    - are divided according to the weight of their connection between the hidden node and the output node
    - and are propagated back to provide the error values (δ) for the hidden layer.

Gradient Descent - Adjusting the Weights

$C(w_1, w_2)$

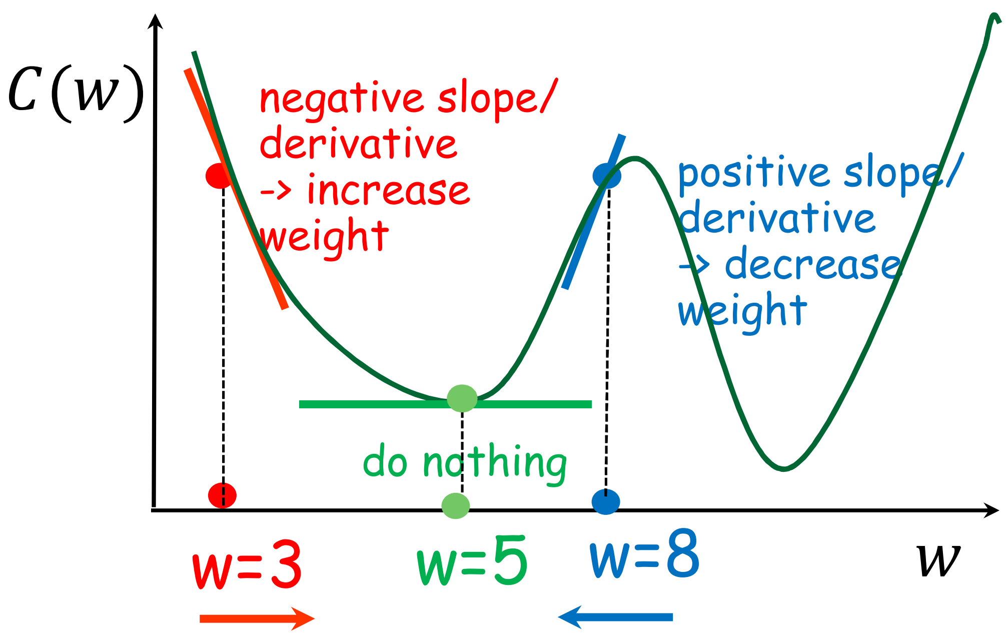


- Goal: minimize $C(w_1, w_2)$ by changing w_1 and w_2
- What is the best combination of change in w_1 and w_2 to minimize C faster?

Gradients

Gradient is just derivative in 1D

Eg: $C(w) = (w-5)^2$ derivative is: $\frac{\partial C}{\partial w} = 2(w-5)$



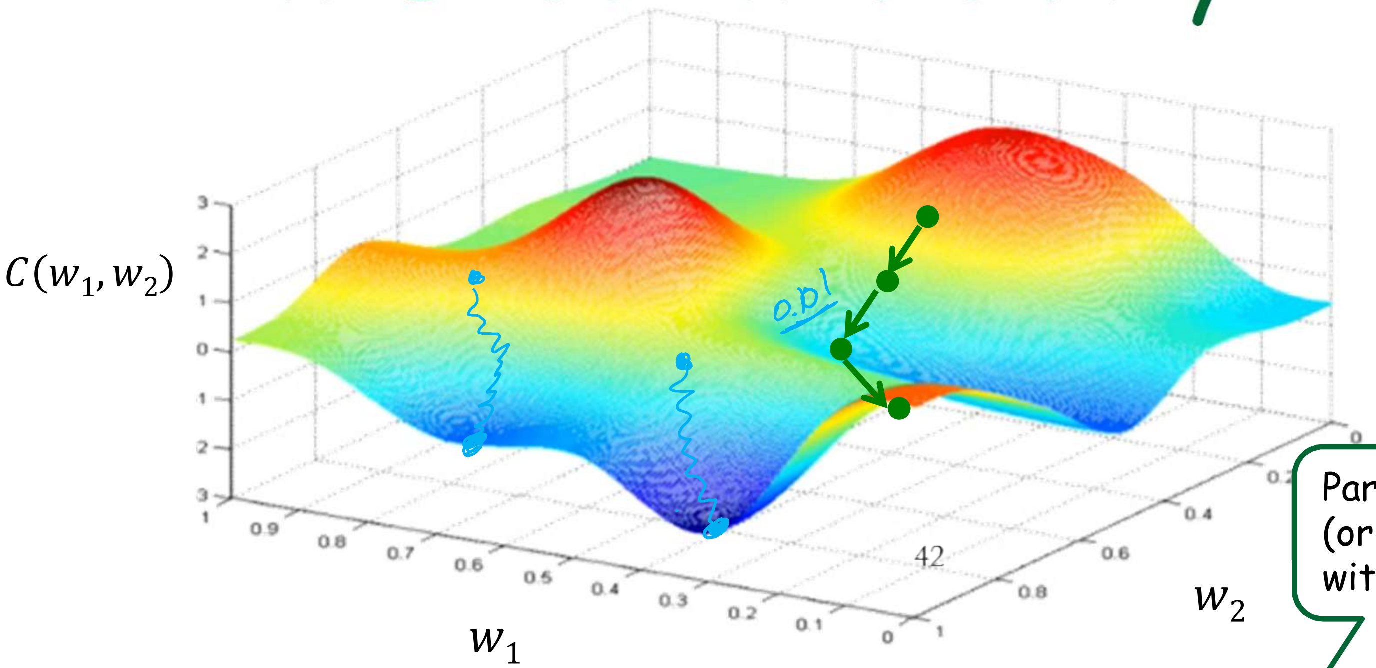
If w=3 $\frac{\partial C}{\partial w}(3) = 2(3-5) = -4$

derivative says increase w (go in opposite direction of derivative)

If w=8 $\frac{\partial C}{\partial w}(8) = 2(8-5) = 6$

derivative says decrease w (go in opposite direction of derivative)

Gradient Descent Visually



Partial derivative (or gradient) of C with respect to w1

- need to know how much a change in w1 will affect $C(w1,w2)$ i.e $\frac{\partial C}{\partial w_1}$
- need to know how much a change in w2 will affect $C(w1,w2)$ i.e $\frac{\partial C}{\partial w_2}$
- Gradient ∂C <u>points</u> in the opposite direction of steepest decrease of $C(w1,w2)$
- i.e. hill-climbing approach…

Source: Andrew Ng

Training the Network

After some calculus (see: https://en.wikipedia.org/wiki/Backpropagation we get…

- Step 0: Initialise the weights of the network randomly

// feedforward

- Step 1: Do a forward pass through the network

$$\boxed{O_i = g\left(\sum_j w_{ji} x_j\right) = \text{sigmoid}\left(\sum_j w_{ji} x_j\right) = \frac{1}{1+e^{-\left(\sum_j w_{ji} x_j\right)}}}$$

// propagate the errors backwards

- Step 2: For each output unit k, calculate its error term δ_k

$$\boxed{\delta_k \leftarrow g'(x_k) \times Err_k = O_k(1 - O_k) \times (O_k - T_k)}$$

- Step 3: For each hidden unit h, calculate its error term δ_h

$$\boxed{\delta_h \leftarrow g'(x_h) \times Err_h = O_h(1 - O_h) \times \sum_{k \in outputs} w_{hk} \delta_k}$$

- Step 4: Update each network weight w_{ij}:

$$\boxed{w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad \text{where } \Delta w_{ij} = -\eta\, \delta_j\, O_i}$$

- Repeat steps 1 to 4 until the cost (C) is minimised

Note: To be consistent with Wikipedia, we'll use O-T instead of T-O, but we will subtract the error in the weight update

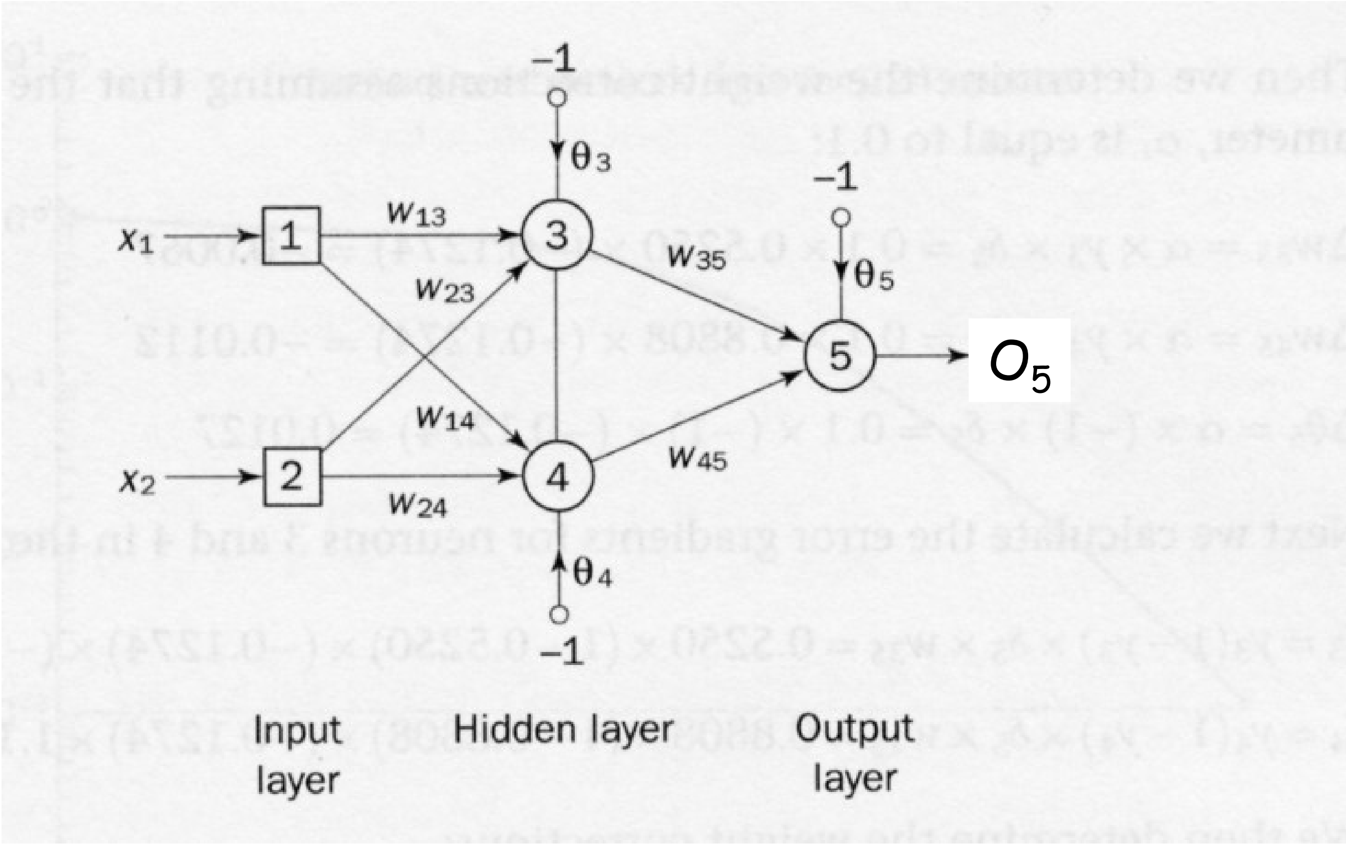
Derivative of sigmoid

note, if we use $g = \text{sigmoid}$:

$g'(x) = g(x)(1 - g(x))$

Sum of the weighted error term of the output nodes that h is connected to (ie. h contributed to the errors δ_k)

Example: XOR

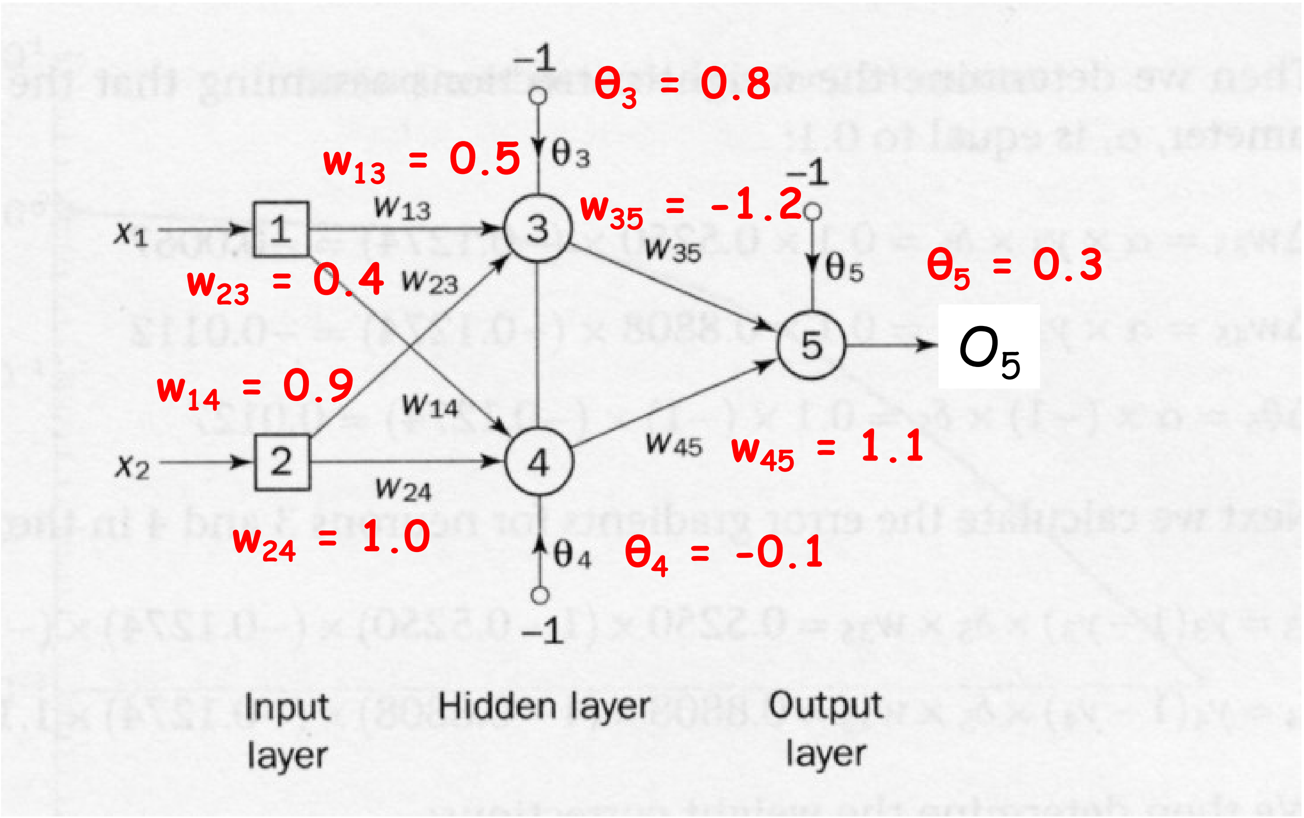


- 2 input nodes + 2 hidden nodes + 1 output node + 3 biases

source: Negnevitsky, Artificial Intelligence, p. 181

Example: Step 0 (initialization)

- Step 0: Initialize the network at random



Step 1: Feed Forward

- Step 1: Feed the inputs and calculate the output

$$O_i = \text{sigmoid}\left(\sum_j w_{ji} x_j\right) = \frac{1}{1+e^{-\left(\sum_j w_{ji} x_j\right)}}$$

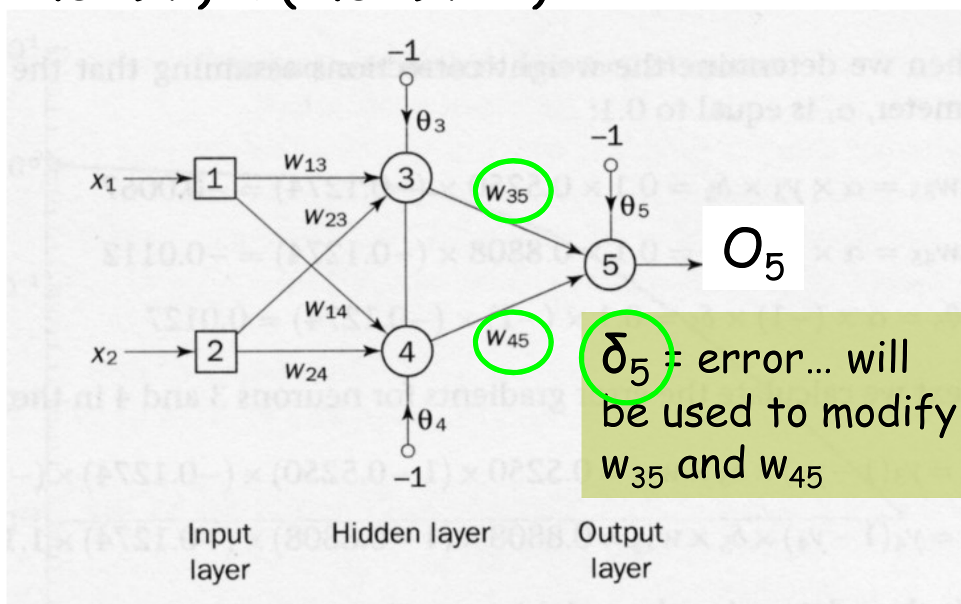
x_1	x_2	Target output T
1	1	0
0	0	0
1	0	1
0	1	1

- With (x_1=1, x_2=1) as input:
  - Output of the hidden node 3:
    - O_3= sigmoid($x_1 w_{13} + x_2 w_{23} - \theta_3$) = 1 / (1 + $e^{-(1x.5 + 1x.4 - 1x.8)}$)= 0.5250
  - Output of the hidden node 4:
    - O_4= sigmoid($x_1 w_{14} + x_2 w_{24} - \theta_4$) = 1 / (1 + $e^{-(1x.9 + 1x1.0 + 1x0.1)}$)= 0.8808
  - Output of neuron 5:
    - O_5 = sigmoid($O_3 w_{35} + O_4 w_{45} - \theta_5$) = 1 / (1 + $e^{-(0.5250x-1.2 + 0.8808x1.1 - 1x0.3)}$)= 0.5097

Step 2: Calculate error term of output layer

$$\boxed{\delta_k \leftarrow g'(x_k)\times Err_k = O_k(1-O_k)\times(O_k - T_k)}$$

- Error term of neuron 5 in the output layer:
  - $\delta_5 = O_5\,(1-O_5)\,(O_5 - T_5)$
    $= (0.5097) \times (1-0.5097) \times (0.5097-0)$
    $= 0.1274$

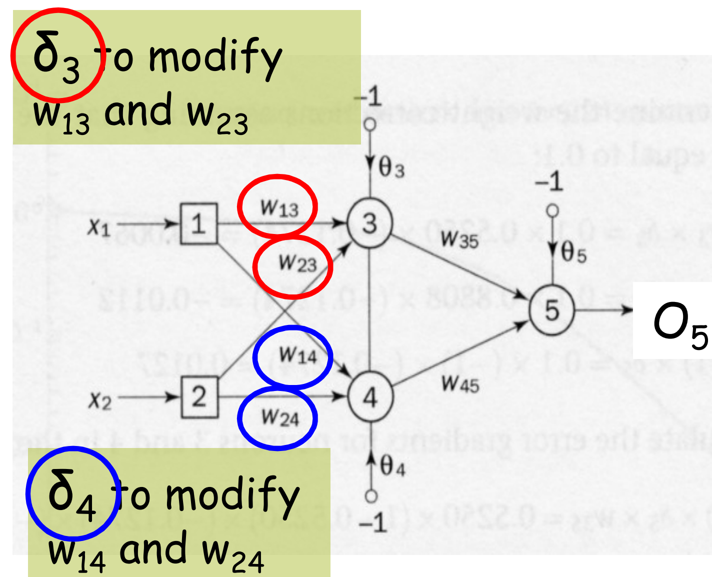


δ_5 = error… will be used to modify w_{35} and w_{45}

Step 3: Calculate error term of hidden layer

$$\boxed{\delta_h \leftarrow g'(x_h) \times Err_h = O_k(1 - O_k) \times \sum_{k \in outputs} w_{kh}\delta_k}$$

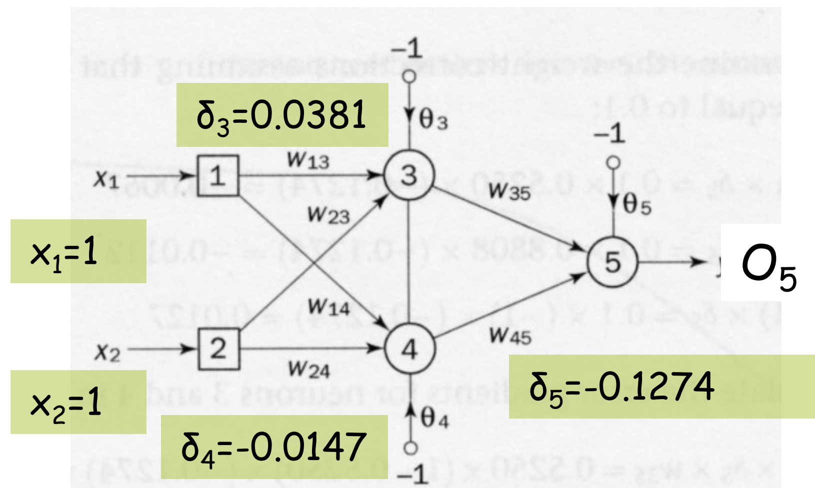
- Error term of neurons 3 & 4 in the hidden layer:
  - $\delta_3 = O_3\,(1-O_3)\;\; \delta_5\, w_{35}$
    $= (0.5250) \times (1-0.5250) \times (0.1274) \times (-1.2)$
    $= -0.0381$
  - $\delta_4 = O_4\,(1-O_4)\; \delta_5\, w_{45}$
    $= (0.8808) \times (1-0.8808) \times (0.1274) \times (1.1)$
    $= 0.0147$



Step 4: Update Weights

$$\boxed{w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad \text{where } \Delta w_{ij} = -\,\eta\, \delta_j\, x_i}$$

- Update all weights (assume a constant learning rate η = 0.1)
  - $\Delta w_{13} = -\alpha\ \delta_3\ x_1 = -0.1 \times -0.0381 \times 1 = 0.0038$
  - $\Delta w_{14} = -\alpha\ \delta_4\ x_1 = -0.1 \times 0.0147 \times 1 = -0.0015$
  - $\Delta w_{23} = -\alpha\ \delta_3\ x_2 = -0.1 \times -0.0381 \times 1 = 0.0038$
  - $\Delta w_{24} = -\alpha\ \delta_4\ x_2 = -0.1 \times 0.0147 \times 1 = -0.0015$
  - $\Delta w_{35} = -\alpha\ \delta_5\ O_3 = -0.1 \times 0.1274 \times 0.5250 = -0.00669$ // O_3 is seen as x_5 (output of 3 is input to 5)
  - $\Delta w_{45} = -\alpha\ \delta_5\ O_4 = -0.1 \times 0.1274 \times 0.8808 = -0.01122$ // O_4 is seen as x_5 (output of 4 is input to 5)
  - $\Delta\theta_3 = -\alpha\ \delta_3\ (-1) = -0.1 \times -0.0381 \times -1 = -0.0038$
  - $\Delta\theta_4 = -\alpha\ \delta_4\ (-1) = -0.1 \times 0.0147 \times -1 = -0.0015$
  - $\Delta\theta_5 = -\alpha\ \delta_5\ (-1) = -0.1 \times 0.1274 \times -1 = -0.0127$

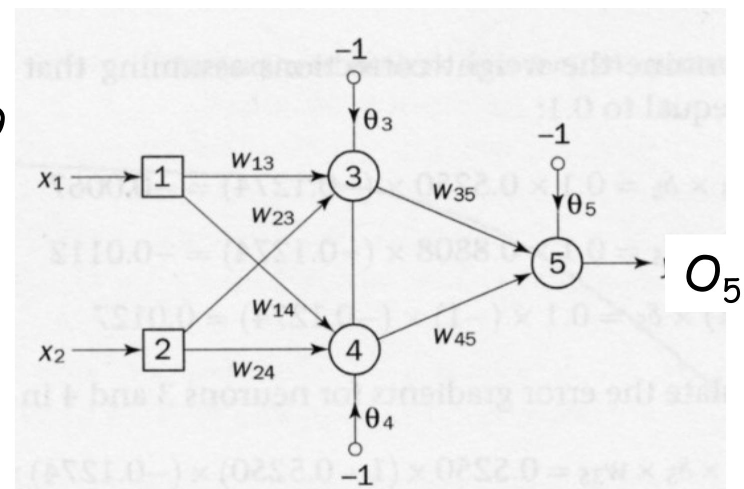


Step 4: Update Weights (con't)

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad \text{where } \Delta w_{ij} = -\eta\, \delta_j\, x_i$$

- Update all weights (assume a constant learning rate η = 0.1)

  - $w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$
  - $w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$
  - $w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$
  - $w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$
  - $w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.00669 = -1.20669$
  - $w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.01122 = 1.08878$
  - $\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$
  - $\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$
  - $\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$



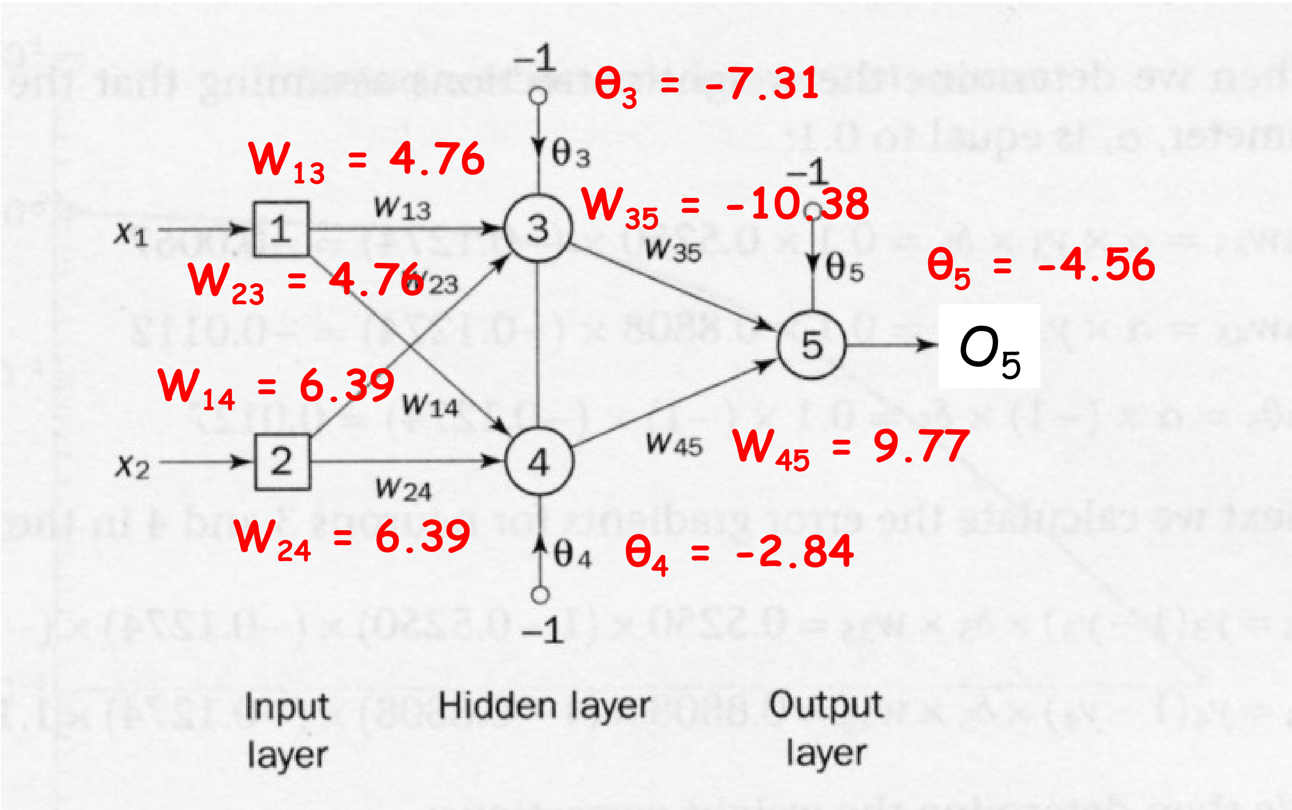
Step 4: Iterate through data

- after adjusting all the weights, repeat the forward pass and back pass for the next data point until all data points are examined
- repeat this entire exercise until the cost function is minimised
  - Eg. $C = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}({T^k}_i \ln({O^k}_i))$

where
n = nb of training examples
K = nb of classed

The Result…

- After 224 epochs, we get:
  - (1 epoch = going through all data once)



Error is minimized

Inputs		Target Output	Actual Output
x_1	x_2	T	O
1	1	false (0)	0.0155
0	1	true (1)	0.9849
1	0	true (1)	0.9849
0	0	false (0)	0.0175

$\Longrightarrow$

i	real distribution (false, true)	predicted distribution (false, true)
1	(1, 0)	(0.9845, 0.0155)
2	(0, 1)	(0.0151, 0.9849)
3	(0, 1)	(0.0151, 0.9849)
4	(1, 0)	(0.9825, 0.0175)

K=2 classes (false, true)

$$C = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}(T^k{}_i \ln(O^k{}_i)) \quad = -\frac{1}{n}\Big($$

$(1)ln(0.9845) + (0)ln(0.0155)$ // for i=1

$+ (0)ln(0.0151) + (1)ln(0.9849)$ // for i=2

$+ (0)ln(0.0151) + (1)ln(0.9849)$ // for i=3

$+ (1)ln(0.9825) + (0)ln(0.0175)$ // for i=4

$\Big) = 0.01592$

May be a local minimum…

Types of Gradient Descent

- Batch Gradient Descent (GD)
  - updates the weights after 1 epoch
  - can be costly (time & memory) since we need to evaluate the whole training dataset before we take one step towards the minimum.

- Stochastic Gradient Descent (SGD)
  - updates the weights after each training example
  - often converges faster compared to GD
  - but the error function is not as well minimized as in the case of GD
  - to obtain better results, shuffle the training set for every epoch

- MiniBatch Gradient Descent:
  - compromise between GD and SGD
  - cut your dataset into sections, and update the weights after training on each section

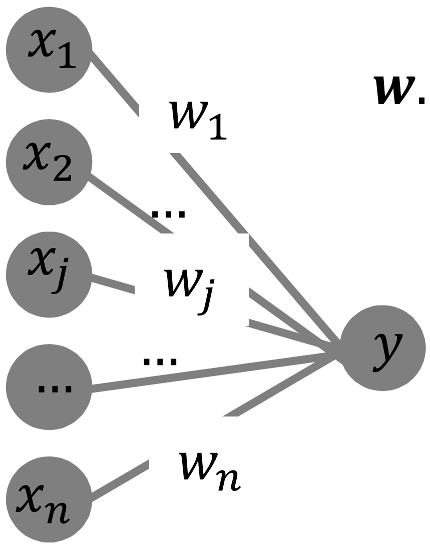
Remember your Linear Algebra

- Dot product (inner product) of 2 vectors

$$\boldsymbol{a}.\boldsymbol{b} = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$= \sum_{i=1}^{m} a_i b_i = (a_1 b_1 + a_2 b_2 + \cdots + a_m b_m)$$

so what?



$$\boldsymbol{w}.\boldsymbol{x} = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$= \sum_{j=1}^{n} w_j x_j = (w_1 x_1 + w_2 x_2 + \dots + w_n x_n)$$

$$= y$$

Remember your Linear Algebra

- matrix-vector product

$$\underset{m \times n}{\begin{bmatrix} w_{11} & \cdots & w_{1j} & \cdots & w_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & w_{ij} & \cdots & w_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & w_{mj} & \cdots & w_{mn} \end{bmatrix}} \underset{n \times 1}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{bmatrix}} = \underset{\mathrm{m} \times 1}{\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_m \end{bmatrix}}$$

$$j = 1 \rightarrow n$$
$$i = 1 \rightarrow m$$

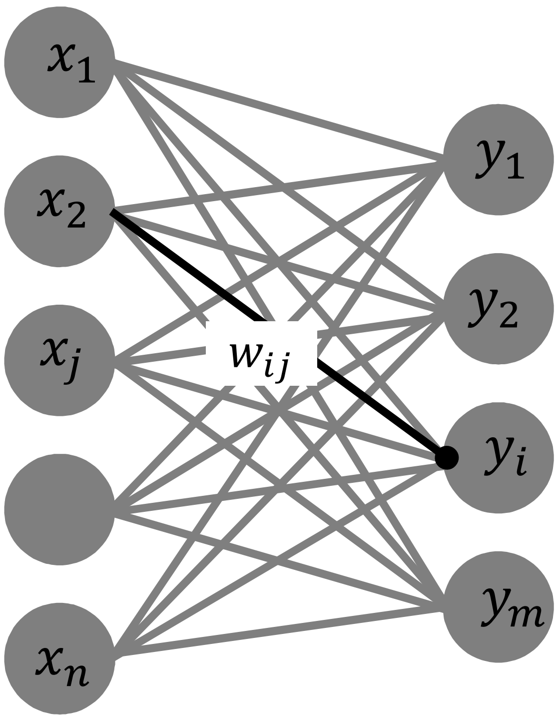
where:

$$y_i = dot\ product\ of\ i^{th}\ row\ of\ \boldsymbol{W}\ with\ \boldsymbol{x}$$

$$y_i = \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} w_{ij} x_{\boldsymbol{j}}$$

Note that the formula in the video was wrong. Please use this formula.

so what?



m *hidden nodes*

n *input nodes*

$w_{ij} = weight\ from\ node\ x_j\ to\ y_i$

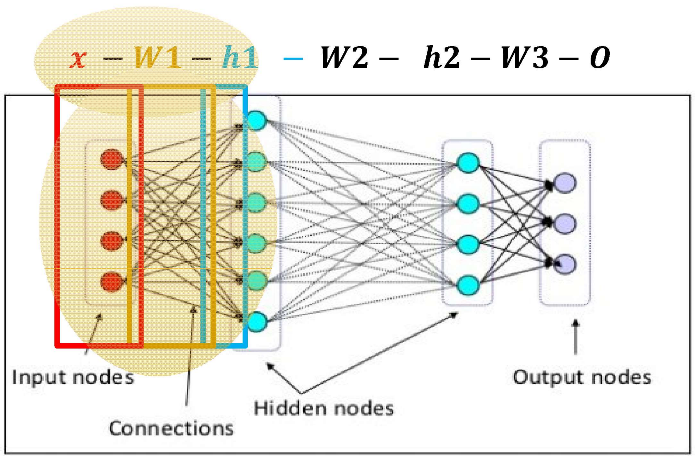
$$y_i = sigmoid\left(\sum_{j=1}^{n} w_{ij} x_j\right)$$

Note that the formula in the video was wrong. Please use this formula.

$$\begin{bmatrix} w_{11} & \cdots & w_{1j} & \cdots & w_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & w_{ij} & \cdots & w_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & w_{mj} & \cdots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_m \end{bmatrix}$$

Matrix Notation - Example

Note that the formula in the video was wrong. Please use this formula.



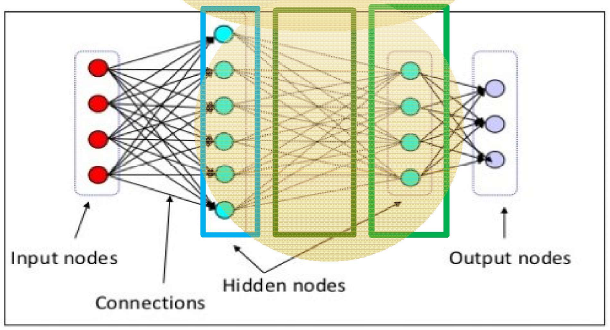
$$h1_i = sigmoid(net_{h1i}) = sigmoid\left(\sum_{j=1}^{4} W1_{ij}x_j\right)$$

$$\begin{array}{ccccc} & 6 \times 4 & 4 \times 1 & & 6 \times 1 \end{array}$$

$$h1_i = sigmoid\left(\begin{bmatrix} w1_{11} & w1_{12} & w1_{13} & w1_{14} \\ w1_{21} & w1_{22} & w1_{23} & w1_{24} \\ w1_{31} & w1_{32} & w1_{33} & w1_{34} \\ w1_{41} & w1_{42} & w1_{43} & w1_{44} \\ w1_{51} & w1_{52} & w1_{53} & w1_{54} \\ w1_{61} & w1_{62} & w1_{63} & w1_{64} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}\right) = \text{sigmoid}\left(\begin{bmatrix} net_{h11} \\ net_{h12} \\ net_{h13} \\ net_{h14} \\ net_{h15} \\ net_{h16} \end{bmatrix}\right) = \begin{bmatrix} h1_1 \\ h1_2 \\ h1_3 \\ h1_4 \\ h1_5 \\ h1_6 \end{bmatrix}$$

Repeat on next level

$\boldsymbol{x} - \boldsymbol{W1} - \boldsymbol{h1} - \boldsymbol{W2} - \boldsymbol{h2} - \boldsymbol{W3} - \boldsymbol{O}$



Note that the formula in the video was wrong. Please use this formula.

$$h2_i = sigmoid(net_{h2i}) = sigmoid\left(\sum_{j=1}^{6} w2_{ij} h1_j\right)$$

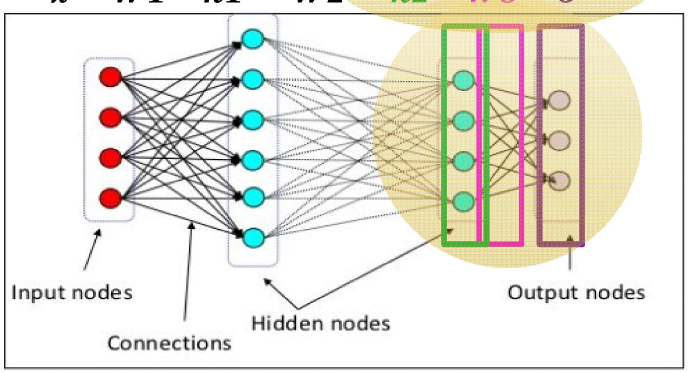
$$\begin{array}{ccc} 4 \times 6 & 6 \times 1 & \\ \end{array}$$

$$h2_i = sigmoid\left(\underbrace{\begin{bmatrix} w2_{11} & w2_{12} & w2_{13} & w2_{14} & w2_{15} & w2_{16} \\ w2_{21} & w2_{22} & w2_{23} & w2_{24} & w2_{25} & w2_{26} \\ w2_{31} & w2_{32} & w2_{33} & w2_{34} & w2_{35} & w2_{36} \\ w2_{41} & w2_{42} & w2_{43} & w2_{44} & w2_{45} & w2_{46} \end{bmatrix}}_{4 \times 6} \underbrace{\begin{bmatrix} h1_1 \\ h1_2 \\ h1_3 \\ h1_4 \\ h1_5 \\ h1_6 \end{bmatrix}}_{6 \times 1}\right) = \text{sigmoid}\left(\begin{bmatrix} net_{h21} \\ net_{h22} \\ net_{h23} \\ net_{h24} \end{bmatrix}\right) = \underbrace{\begin{bmatrix} h2_1 \\ h2_2 \\ h2_3 \\ h2_4 \end{bmatrix}}_{4 \times 1}$$

Repeat on next level

Note that the formula in the video was wrong. Please use this formula.

$$\boldsymbol{x - W1 - h1 - W2 - h2 - W3 - O}$$



$$O_i = sigmoid(net_{Oi}) = sigmoid\left(\sum_{j=1}^{4} W3_{ij} h2_j\right)$$

$$\begin{array}{ccc} 3 \times 4 & 4 \times 1 & & 3 \times 1 \end{array}$$

$$O_i = sigmoid\left(\begin{bmatrix} w3_{11} & w2_{12} & w2_{13} & w2_{14} \\ w3_{21} & w2_{22} & w2_{23} & w2_{24} \\ w3_{31} & w2_{32} & w2_{33} & w2_{34} \end{bmatrix} \begin{bmatrix} h2_1 \\ h2_2 \\ h2_3 \\ h2_4 \end{bmatrix}\right) = \mathrm{sigmoid}\left(\begin{bmatrix} net_{O1} \\ net_{O2} \\ net_{O3} \end{bmatrix}\right) = \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}$$

Neural Networks

- Disadvantage:
  - result is not easy to understand by humans (set of weights compared to decision tree)... it is a black box

- Advantage:
  - robust to noise in the input (small changes in input do not normally cause a change in output) and graceful degradation

Today

1. Introduction to ML
2. Naïve Bayes Classification
   a. Application to Spam Filtering
3. Decision Trees
4. (Evaluation
5. Unsupervised Learning)
6. Neural Networks
   a. Perceptrons
   b. Multi Layered Neural Networks

Up Next

Part 4: Search