Introduction to Artificial Intelligence

VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

COMP307/AIML420

Neural Networks: Tutorial

Dr Fangfang Zhang

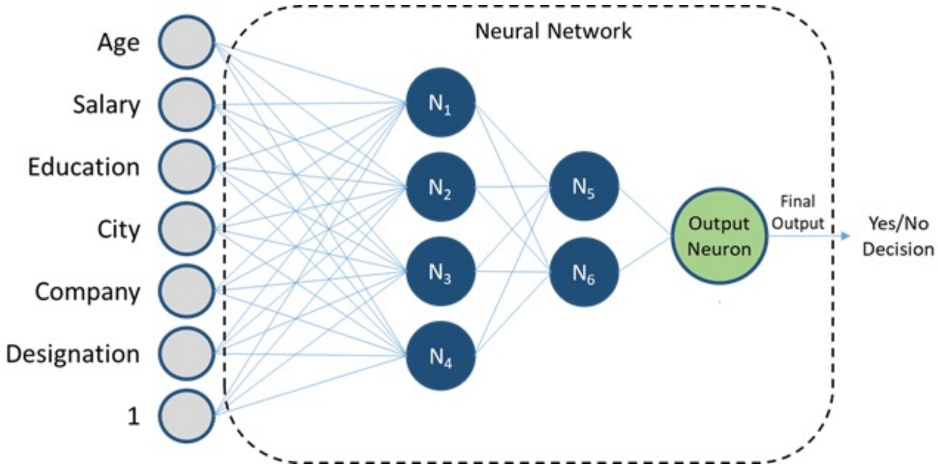
fangfang.zhang@ecs.vuw.ac.nz

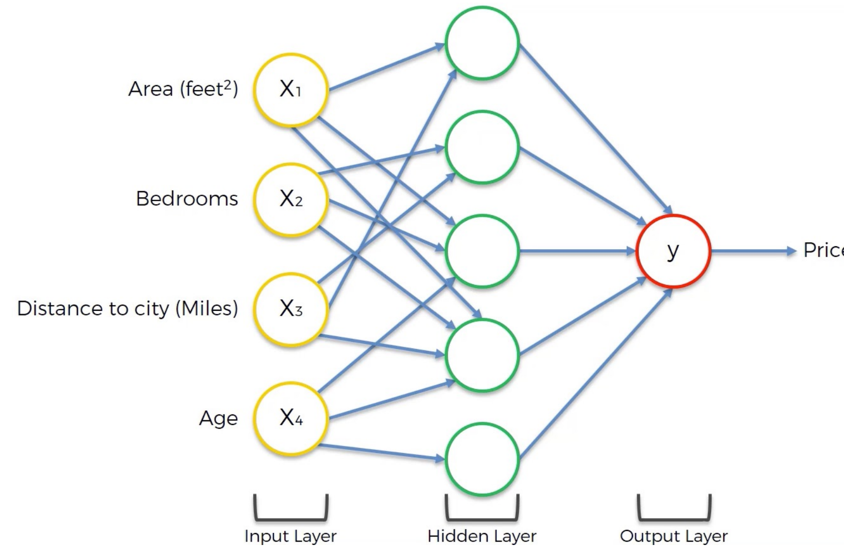
COMP307/AIML420 Week 4 (Tutorial)

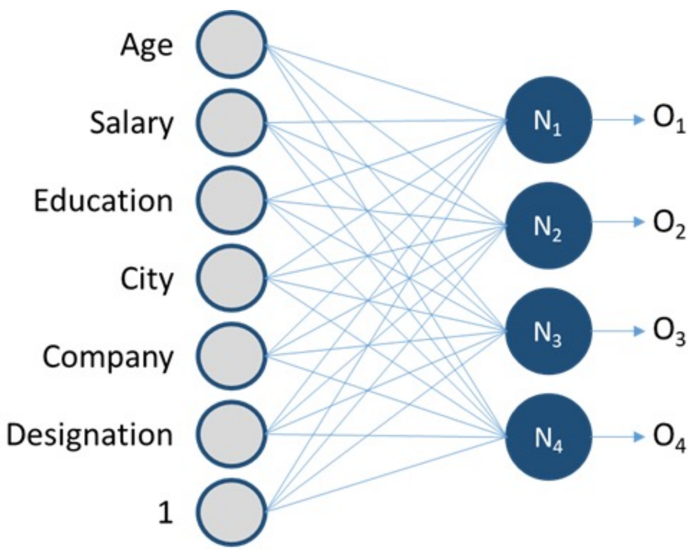
1. Announcements
   – Assignment 1 (**15%**)
   – Due on 30th Mar.
   – Helpdesk (2 hours) this Thurs.---next Wed.
   – Part 1 and Part 2
   – Part 3

2. Neural Networks
   - Perceptron (Part 3)
   - Back Propagation

Neural Networks







- Flexible structure
- Number of inputs
- Number of layers
- Fully/partially connected
- Number of outputs

Depends on the problems!

Ups and downs of Neural Networks

- 1958: Perceptron
- 1969: Perceptron has limitations
- 1980s: Multi-layer perceptron

  Do not have a significant difference from DNN today
- 1986: Backpropagation

  Improve the efficiency of NN learning
- 1989: 1 hidden layer is “good enough”, why deep?
- 2011: start to be popular
- ……

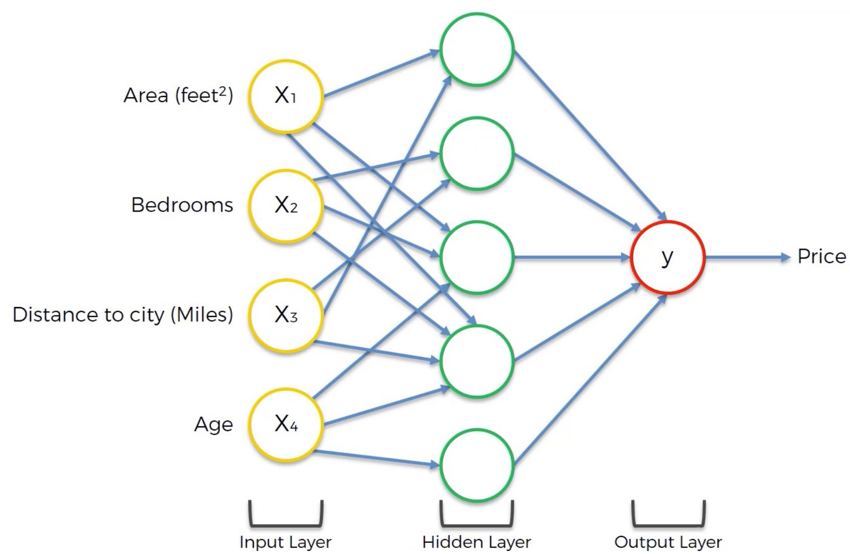
Perceptron

- Three steps for learning

Define function → goodness of function → pick the best function

- neural networks
- decision tree

- accuracy



- Different dataset

--- image data

--- text data

--- speech data

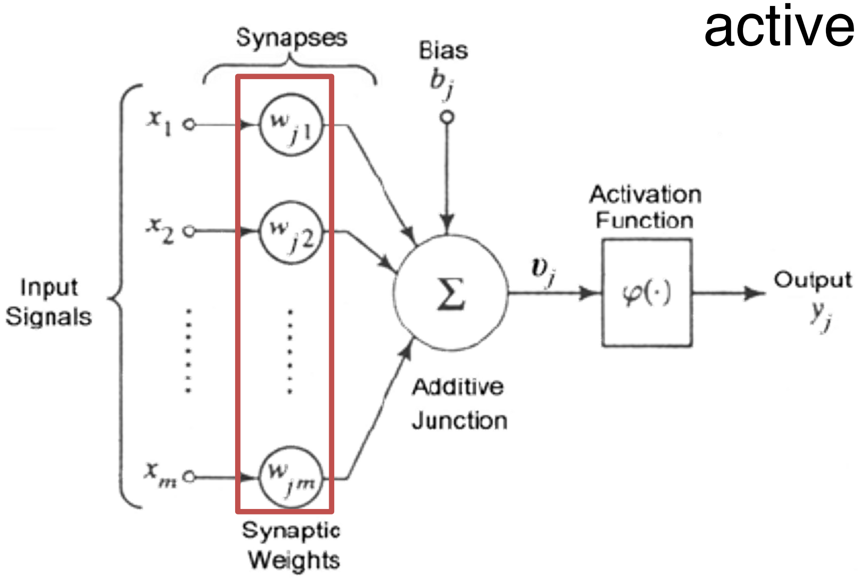
--- tabular data (with numbers)

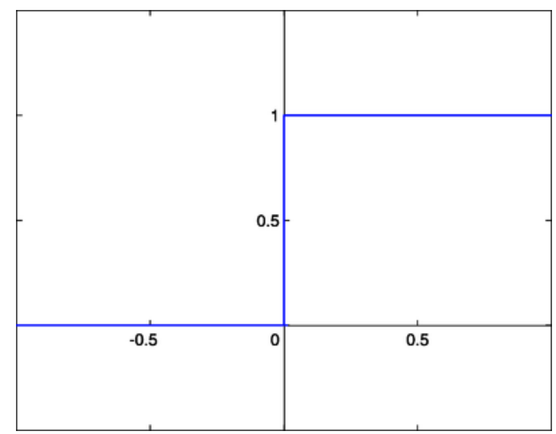
Perceptron

- What we have (classification task):

  Dataset (features, labels)

  Neural network structure (features=input, labels=output, active function)





Active function

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} w_{ji} x_i + b_j > 0, \\ 0, & \text{otherwise} \end{cases}$$

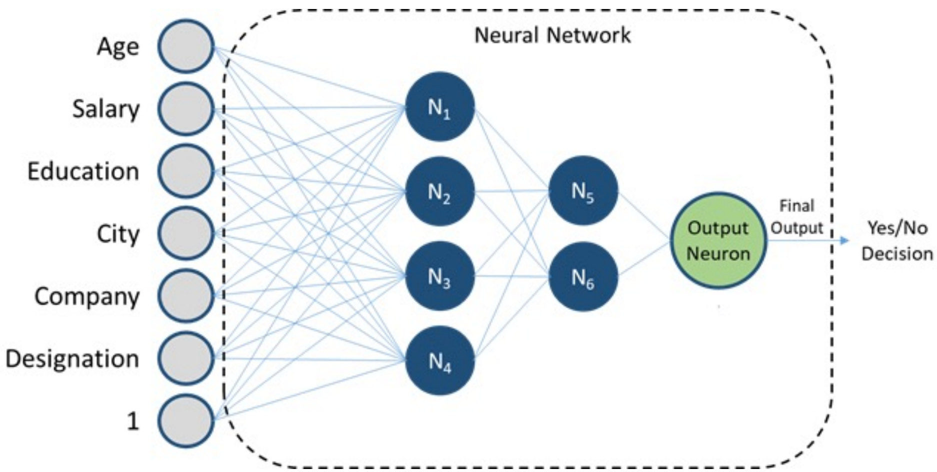
- Perceptron learning

  --- **learning weights and bias**

Perceptron

- How to get the **optimal** weights and bias?
  --- important features may have larger weights
- To simplify notation, we can transform the bias to a weight $w_{j0} = b_j$, where $x_0 = 1$ (dummy feature) always holds

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} w_{ji}x_i + b_j > 0, \\ 0, & \text{otherwise} \end{cases}$$



$$\boxed{b_j = w_{j0} \cdot 1 = w_{j0}x_0}$$

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} w_{ji}x_i > 0, \\ 0, & \text{otherwise} \end{cases}$$

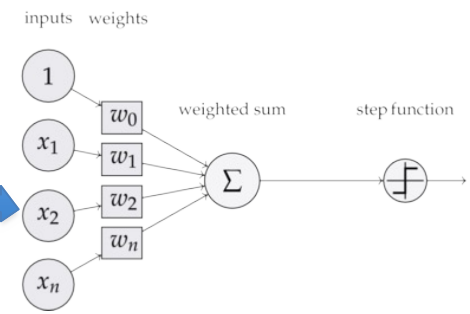
Perceptron

dataset

length	width	weight	class
12	7	43	A (1)
1	24	51	B (0)

1. Define a dummy feature (for bias)
2. Random initialise a set of weights (how many?)
3. Get the first instance in the dataset
4. Sum up *feature values and weights*, and get predicted class label along with active function

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} w_{ji} x_i > 0, \\ 0, & \text{otherwise} \end{cases}$$



5. Adjust the weights (class labels, i.e., 1, 0)

$$w_i \leftarrow w_i + (d - y)x_i, i = 0, 1, 2, \ldots, m$$

d is the desired class label, *y* is the predicted class label

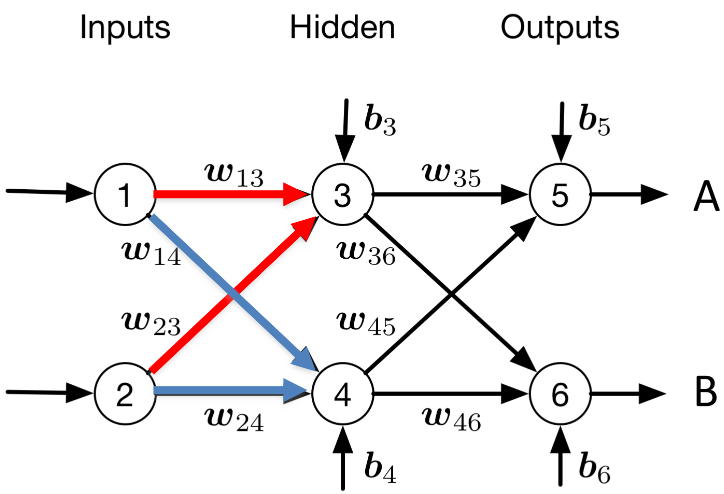
- *d = 1, y = 1 or d = 0, y = 0* (same) *nothing*
- d = 1, y = 0, d – y > 0, increase (different)
- d = 0, y = 1, d – y < 0, decrease (different)

6. Go to step 4, and use the next instance (until meet the stop criterion)

NN Example

- Calculate the outputs of this network (feedforward): to 2dp

I_1	I_2	w_{13}	w_{14}	w_{23}	w_{24}	w_{35}	w_{36}	w_{45}	w_{46}	b_3	b_4	b_5	b_6
0.90	-0.20	0.72	-0.31	0.10	-0.92	-0.37	0.43	-0.19	0.78	0.01	0.38	-0.13	0.78



$Z_3 = w_{13}*I_1+w_{23}*I_2 + b_3$
$= 0.72*0.90 + 0.10*(-0.2) + 0.01$
$= 0.64$

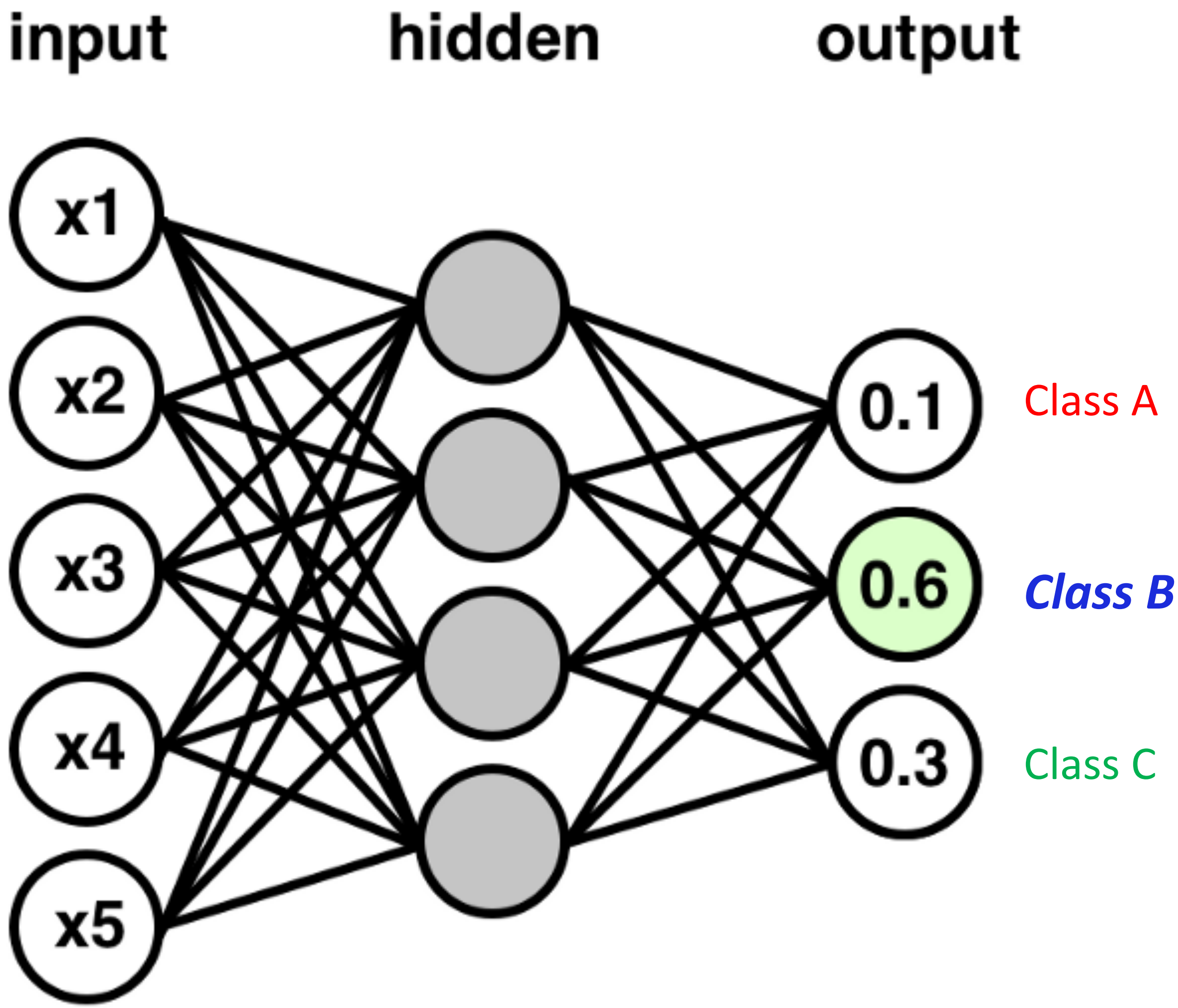
$Z_4 = w_{14}*I_1+w_{24}*I_2 + b_4$
$= (-0.31)*0.90 + (-0.92)*(-0.2)$
$+ 0.38$
$= 0.29$

z_3	0.64
O_3	0.65
z_4	0.29
O_4	0.57
z_5	-0.48
O_5	0.62
z_6	1.50
O_6	0.82

1. Weighted sum of a node: $$z_j = \sum_i w_{ji}x_i + b_j$$
2. Output of a node:
   - Where φ is the activation function
3. Assume φ is the sigmoid function: $$O_j = \varphi(z_j) = \frac{1}{1+e^{-z_j}}$$

Class = ? B

NNs for (Multi-Class) Classification



Training a Neural Network

- **Initialise** the weights (randomly)

- **Feedforward**
  - For each example/instance, calculate the predicted outputs o_z using the current weights
  - Calculate the total *error* $\sum_z (d_z - o_z)^2$
  - d_z means "*desired*"
  - o_z means "output" (i.e. what we actually got)

- If the error is small enough, we can stop.

- Otherwise, we use **back propagation** to adjust the weights to make the error *smaller*.
  - Uses gradient descent (GD)

Back Propagation (BP) Algorithm

- **Improve the efficiency of NN learning**
- **How to update the weights**

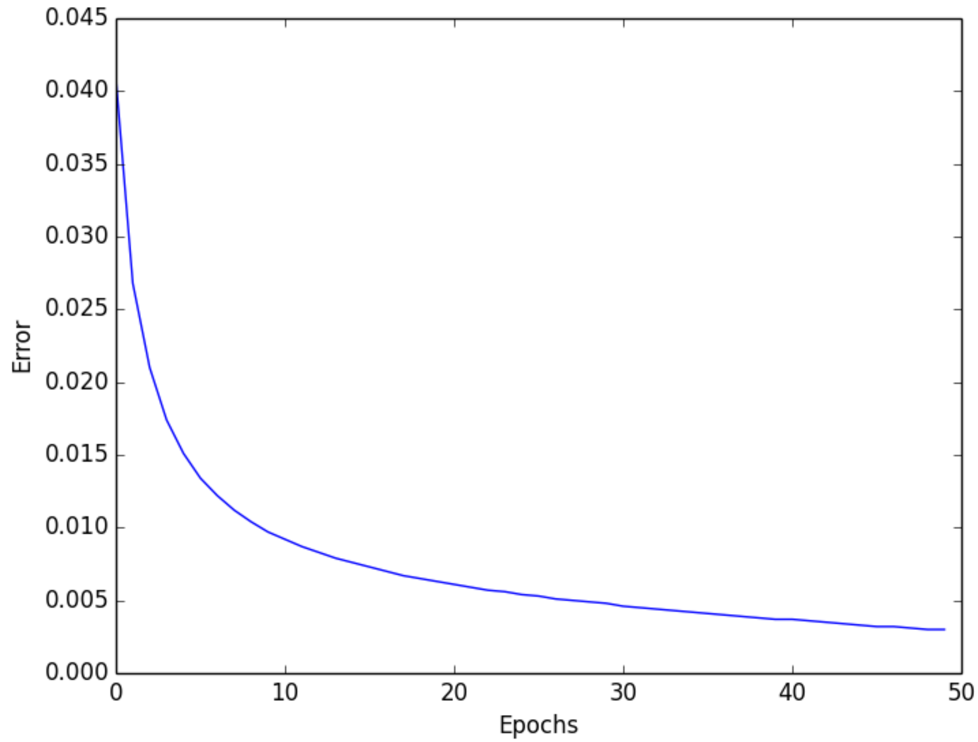
- Estimate the **contribution (gradient)** of each weight to the *error*, i.e. how much the error will be reduced by changing the weight (gradient)

- Change each weight (simultaneously) proportional to its contribution to reduce the error as much as possible
  - Move in the direction of the steepest gradient

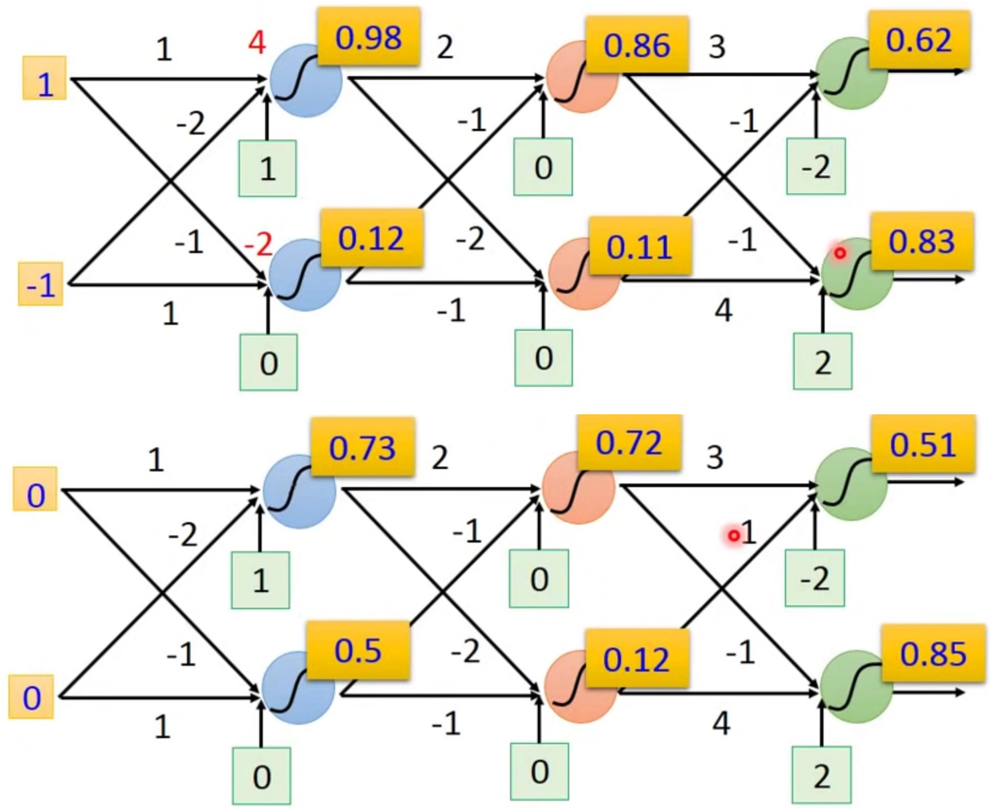
- We calculate the contribution/gradient backwards (from the last/output layer to the first hidden layer)

Notes on BP Algorithm

- *1 Epoch*: all input examples (entire training set, batch, …)
- A target of 0 or 1 cannot reasonably be reached. Usually interpret an output > 0.9 or > 0.8 as '1'
- Training may require *thousands* of epochs. A convergence curve will help to decide when to stop (over-fitting?)



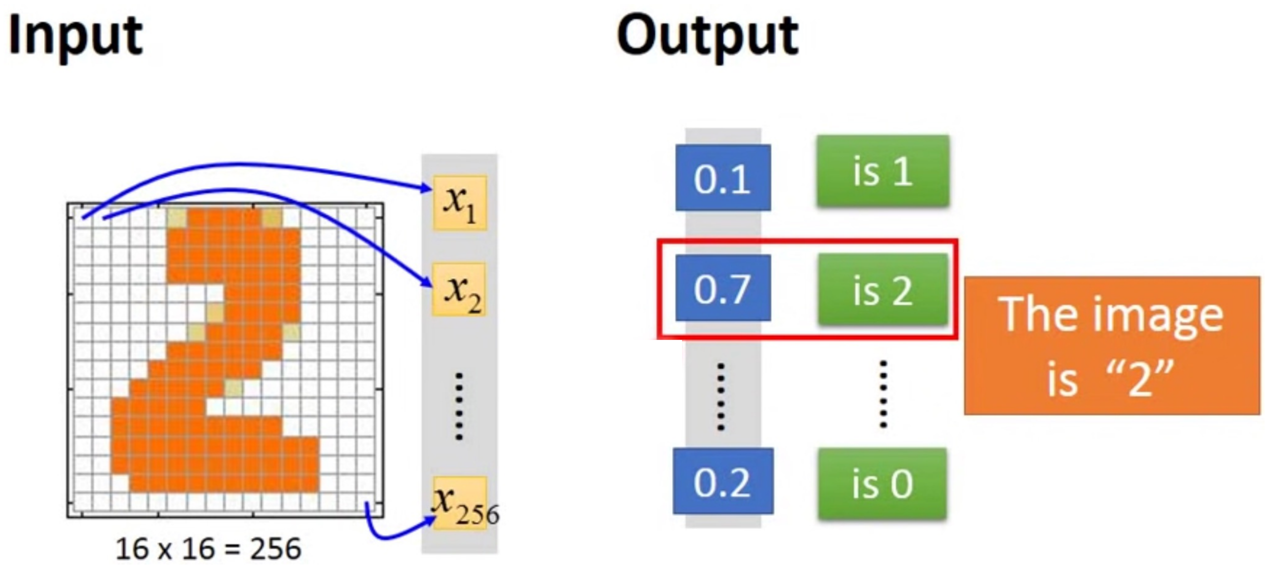
Back Propagation



This is a function.
Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

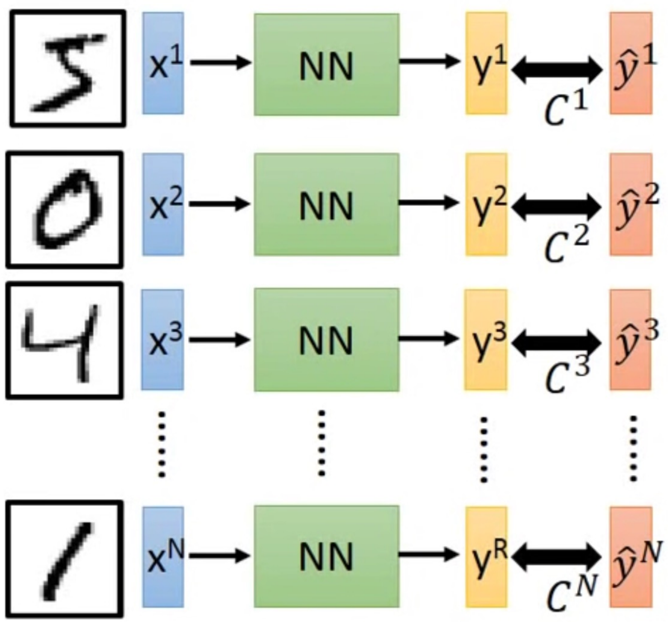
Back Propagation



Back Propagation

Total Loss

For all training data …



Total Loss:

$$\boxed{L = \sum_{n=1}^{N} C^n}$$

Find ***a function in function set*** that minimizes total loss L

Find ***the network parameters θ^**** that minimize total loss L

Back Propagation

- Chain Rule

Case 1 $\quad y = g(x) \quad z = h(y)$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

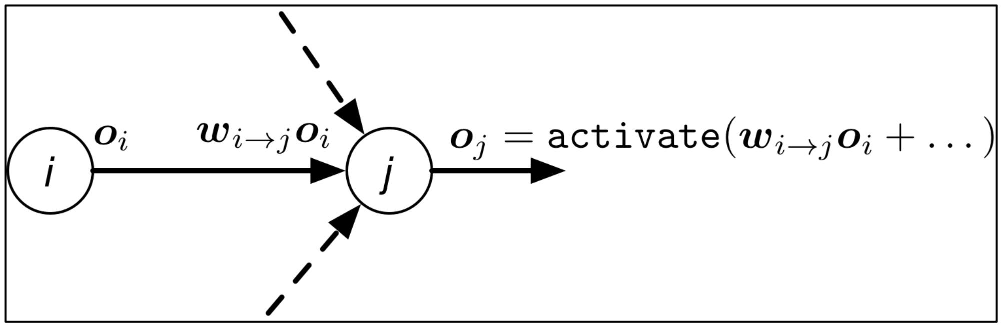
Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

$$\begin{tikzcd}
 & \Delta x \arrow[dr] & \\
\Delta s \arrow[ur] \arrow[dr] & & \Delta z \\
 & \Delta y \arrow[ur] &
\end{tikzcd}
\qquad \frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

Back Propagation (BP) Algorithm

- How big a change should we make to weight $w_{i\to j}$?
  - Make a big change if will improve error a lot (big contribution)
  - Make a small change if there is little effect on error (small contribution)



$w_{i\to j}$ $\quad$ $w_{i\to j} o_i$ $\quad$ $o_j = act(w_{i\to j} o_i + \cdots)$ $\quad$ … $\quad$ $error$

- β_j is how "*beneficial*" a change is for node j ("error term")
- When changing $w_{i\to j}$, the error change should be:
  - Proportional to the output: o_i (larger output = more effect)
  - Proportional to the *slope* of the activation function at node j: $slope_j$
  - Proportional to error term of j (β_j)

BP Algorithm Implementation

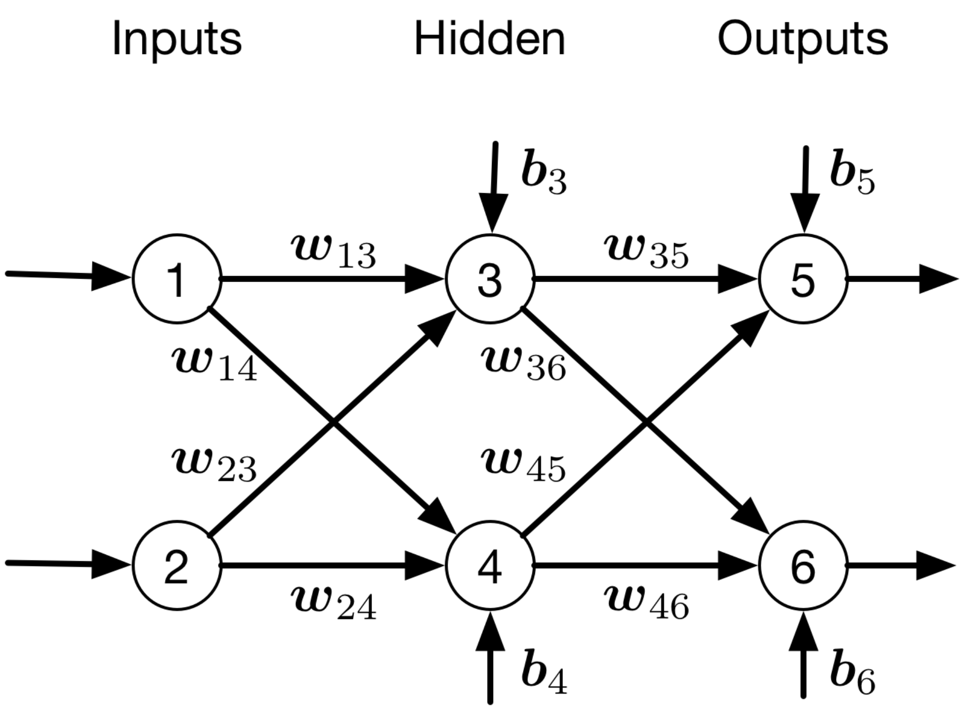
- Initialise all weights (+bias) to small random values
- Until total error is small enough, repeat:
  - For each input example:
    - Feed forward pass to get predicted outputs
    - Compute $\beta_z = d_z - o_z$ for each output node
    - Compute $\beta_j = \sum_k w_{j\to k} o_k (1 - o_k) \beta_k$ for each hidden node (working backwards from last to first layer)
    - Compute (+store) the weight changes for all weights

      $\Delta w_{i\to j} = \eta o_i o_j (1 - o_j) \beta_j$ (proportional to all 3 factors), Let η be the learning rate
  - Sum up weight changes for all input examples
  - Change weights!

NN Example: Your Turn!

- Calculate the new weights and biases (backprop): to 2dp

d_5	d_6	η	β_3	β_4	β_5	β_6
0	1	1				



w_{13}	
w_{14}	
w_{23}	
w_{24}	
w_{35}	
w_{36}	
w_{45}	
w_{46}	
b_3	
b_4	
b_5	
b_6	

Useful Formulae: Backprop

- Error term of an output node: $\beta_j = d_j - O_j$

- Error term of a hidden node: $\beta_j = \sum_k w_{j \to k} O_k (1 - O_k) \beta_k$
  - (For the sigmoid activation function)

- Amount to change a weight: $\Delta w_{i \to j} = \eta O_i O_j \left(1 - O_j\right) \beta_j$

- Amount to change a bias: $\Delta b_j = \eta O_j \left(1 - O_j\right) \beta_j$

Summary

- Perceptron
- Back Propagation

- Next week

  --- Neural Engineering (next Monday)

  --- Evolutionary Computation (next Tuesday)