Introduction to Artificial Intelligence

VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

COMP307/AIML420

Neural Networks: Tutorial

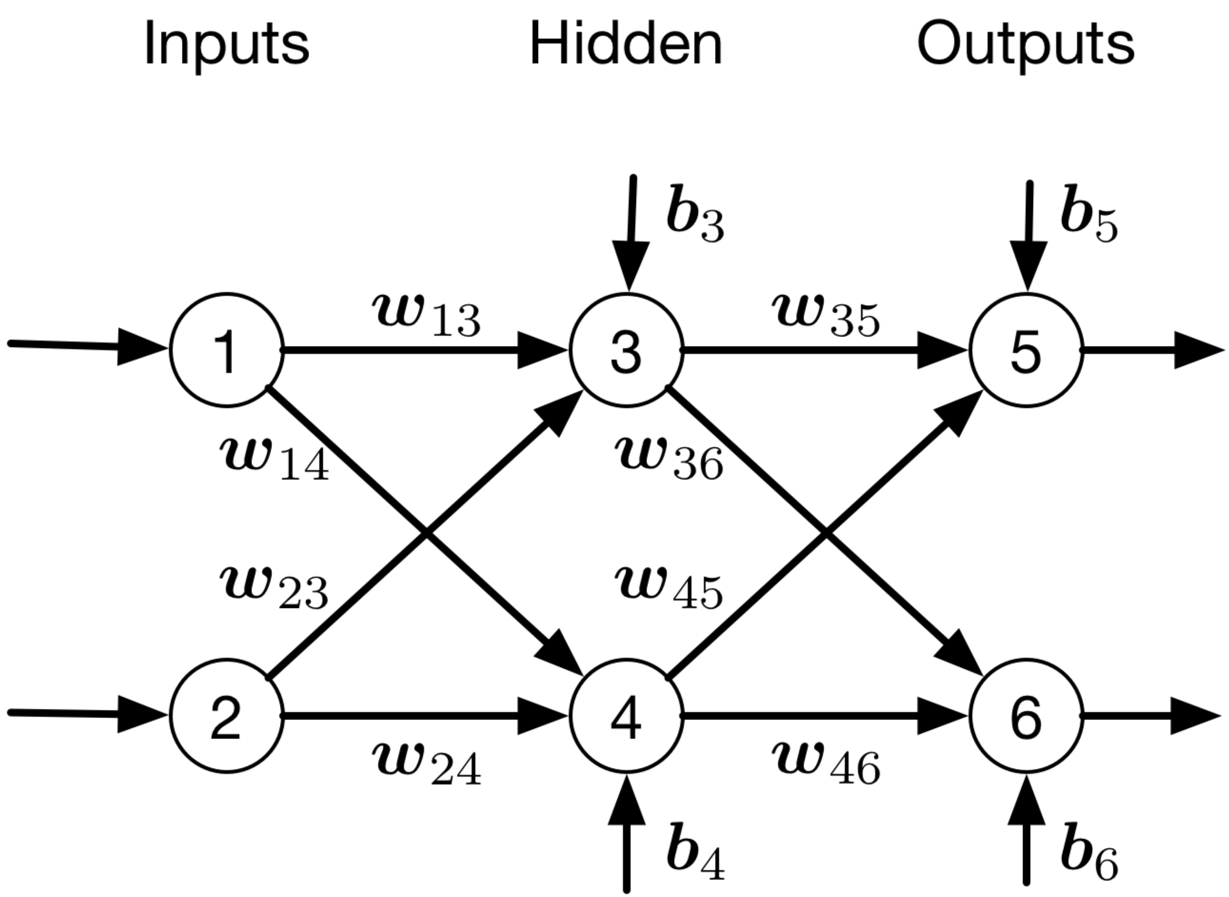
Dr Andrew Lensen

Andrew.Lensen@vuw.ac.nz

NN Example: Your Turn!

- 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	
O_3	
z_4	
O_4	
z_5	
O_5	
z_6	
O_6	

Class = ____

Useful Formulae: Feedforward

- Weighted sum (ws)of a node: $$z_j = \sum_i w_{ji} x_i + b_j$$

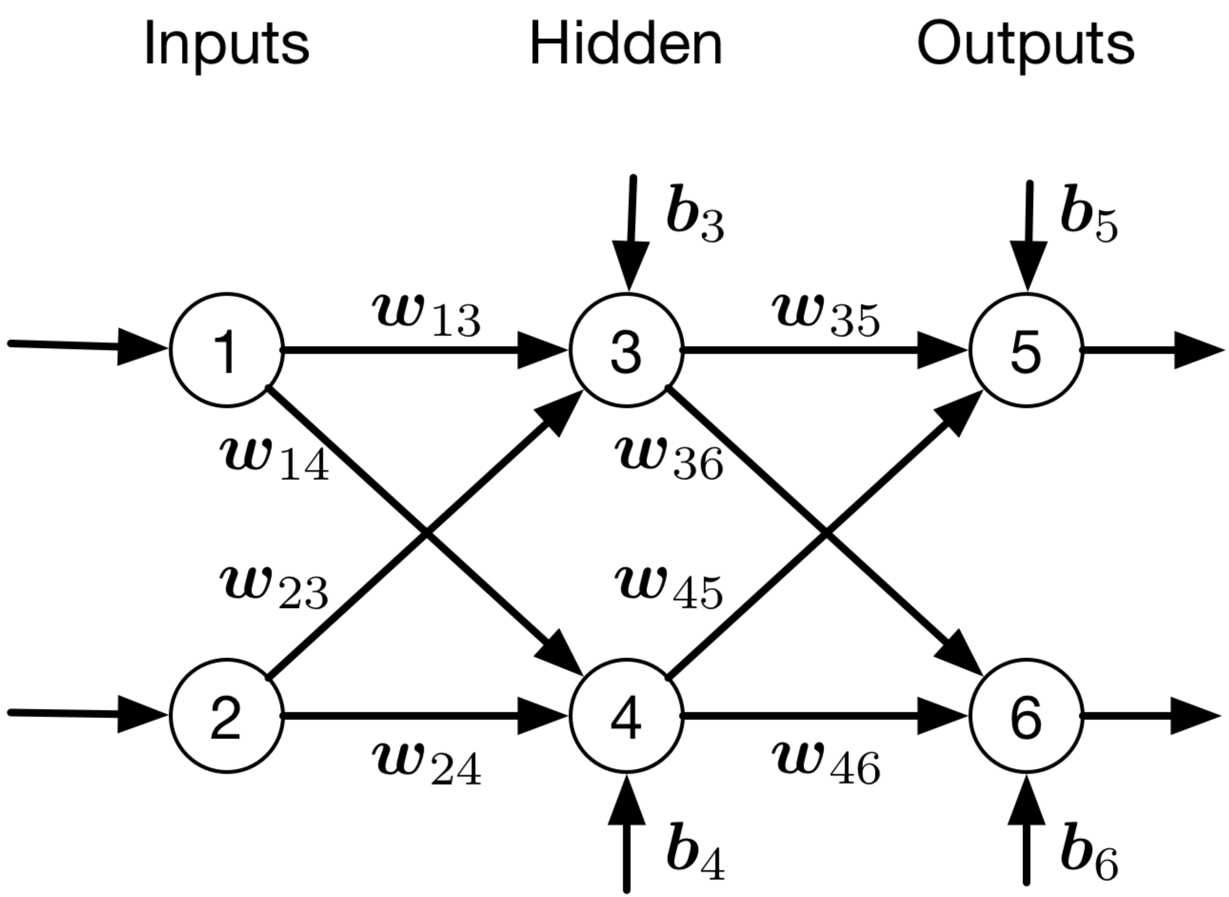
- Output of a node: $$O_j = \varphi(z_j)$$
  - Where φ is the activation function.

- Assume φ is the sigmoid function: $$\varphi(z_j) = \frac{1}{1 + e^{-z_j}}$$

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				



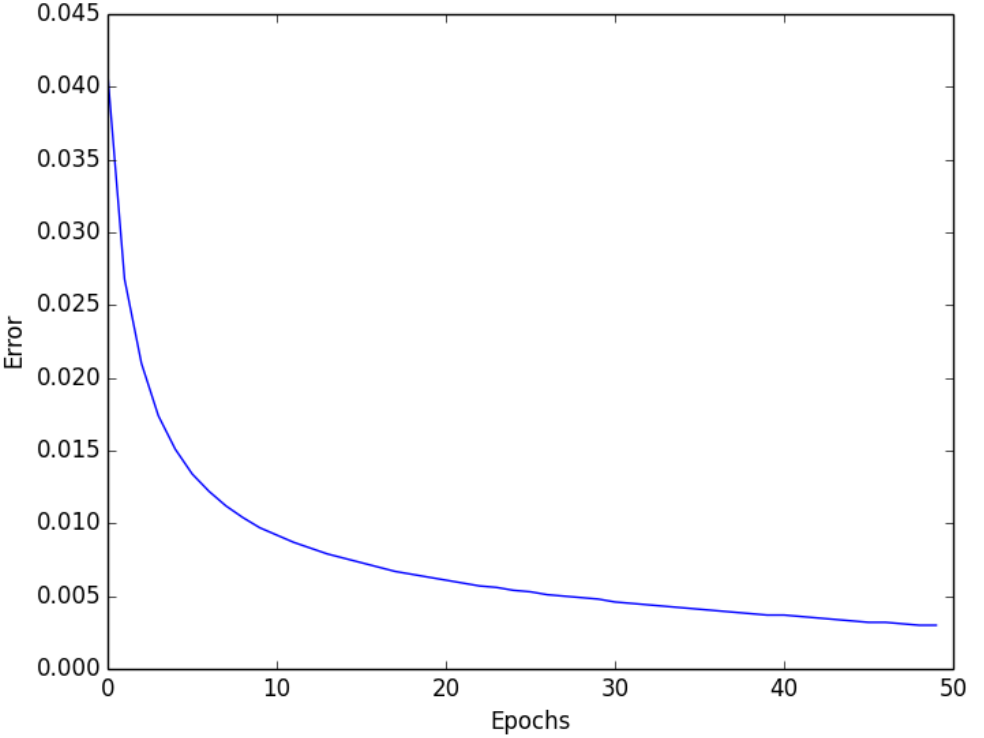
w_{13}	
w_{14}	
w_{23}	
w_{24}	
w_{35}	
w_{36}	
w_{45}	
w_{46}	
b_3	
b_4	
b_5	
b_6	

Useful Formulae: Backprop

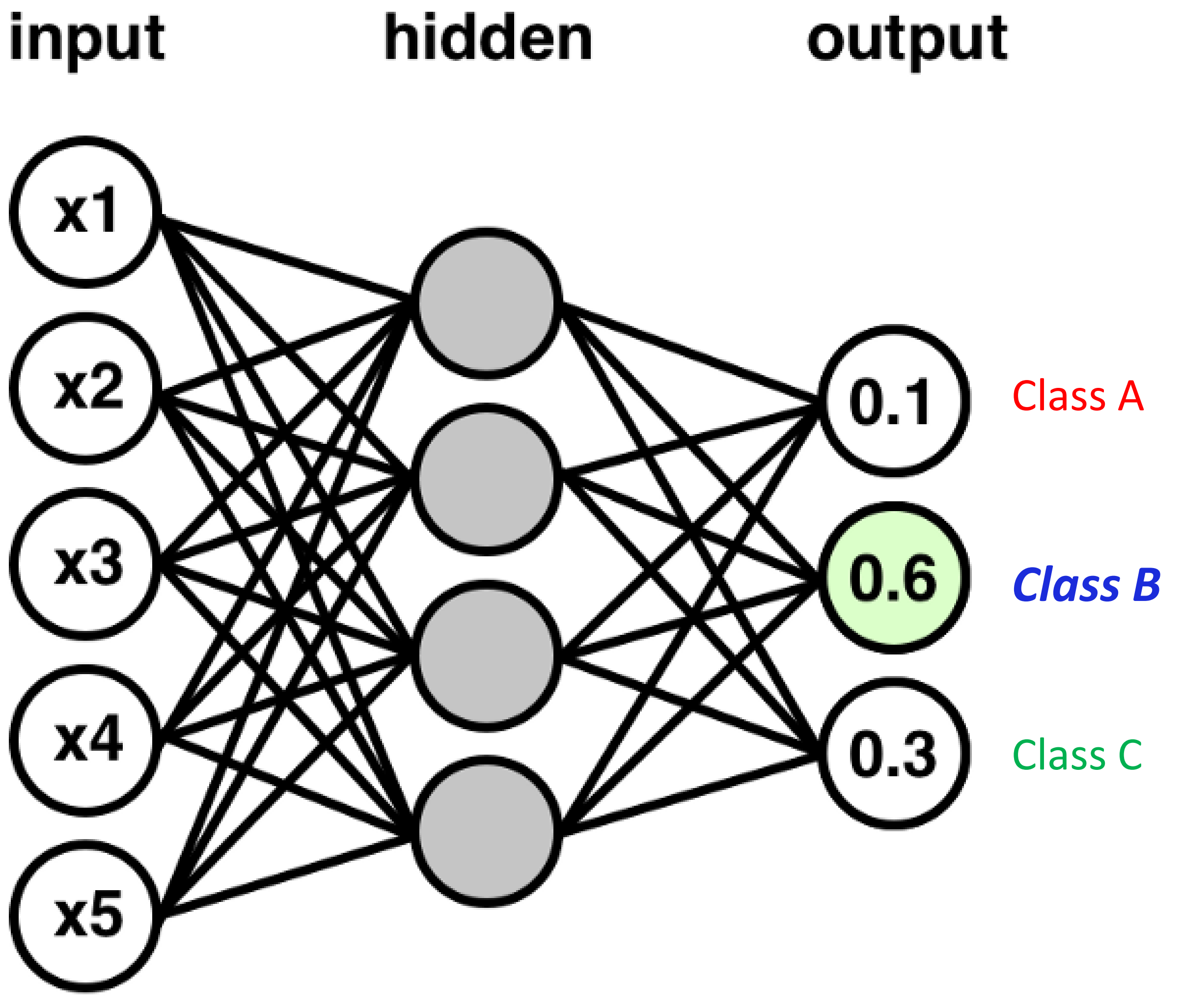
- Error term of an output node: $\beta_j = d_j - O_j$
- Error term of a hidden node: $\beta_j = \sum_k w_{j \to k} O_k (1 - O_k) \beta_k$
  - (For the sigmoid activation function)
- Amount to change a weight: $\Delta w_{i \to j} = \eta O_i O_j \left(1 - O_j\right) \beta_j$
- Amount to change a bias: $\Delta b_j = \eta O_j \left(1 - O_j\right) \beta_j$

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?)



NNs for (Multi-Class) Classification



Training a Neural Network

- **Initialise** the weights (randomly)

- **Feedforward**
  - For each example, calculate the predicted outputs o_z using the current weights
  - Calculate the total *error* $\sum_z (d_z - o_z)^2$

- 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

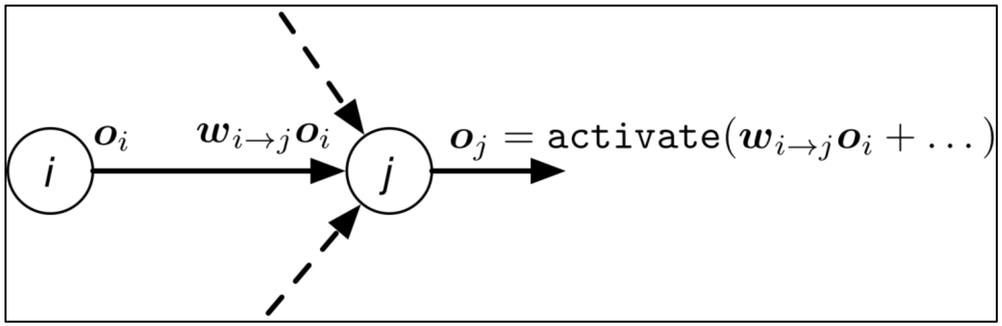
- Estimate the **contribution (gradient)** of each weight to the *error*, i.e. how much the error will be reduced by changing the weight (gradient)

- 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)
- Error of a single output node is $d_z - o_z$
  - d_z means "*desired*"
  - o_z means "output" (i.e. what we actually got)

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 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

- Let η be the learning rate ("eta"…)

- 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)
  - Sum up weight changes for all input examples
  - Change weights!