Introduction to Artificial Intelligence

COMP307/AIML420

Neural Networks: Tutorial

Dr Junhong(Jennifer) Zhao

jennifer.zhao@vuw.ac.nz

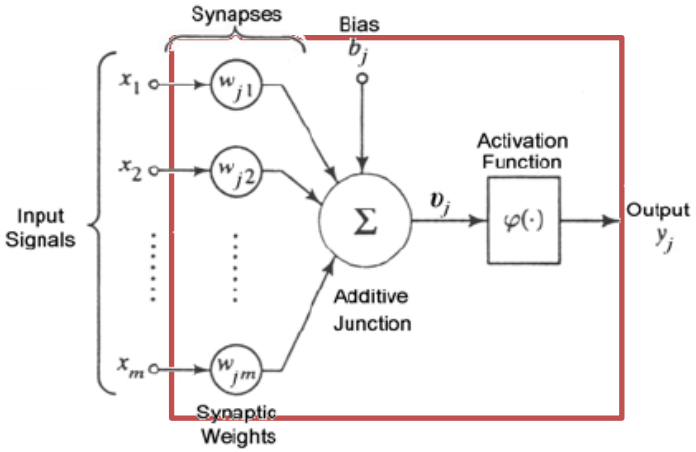
j.zhao@vuw.ac.nz

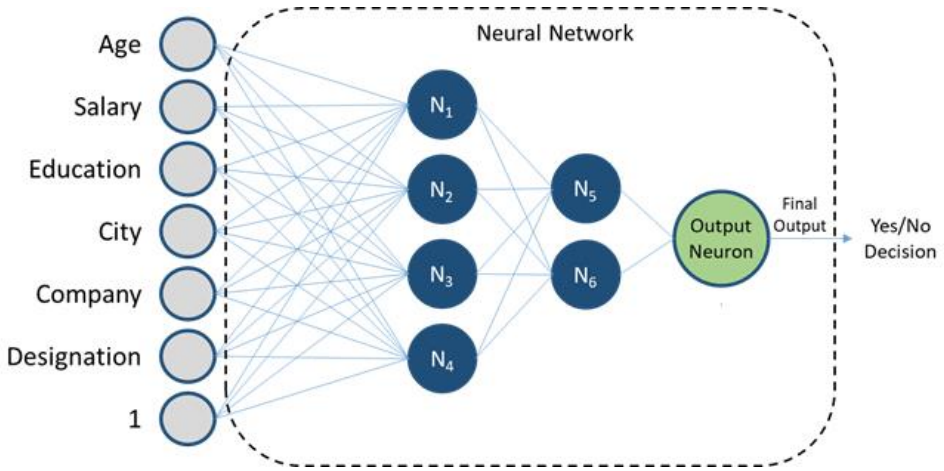
COMP307/AIML420 Week 4 (Tutorial)

1. Announcement
   - Assignment 1 (**15%**)
   - Due on 29th Mar (next Wed.)
     - Part 1- KNN
     - Part 2 - DT
     - Part 3 - Perceptron
   - Helpdesk (CO242B, 3-4pm)

2. Neural Networks
   - Perceptron
   - Feedforward NN
     - Forward propagation
     - Back Propagation

Neural Networks





- Number of inputs
- Number of outputs
- Fully/partially connected

- Number of layers
- Number of hidden nodes

Depends on the problems!

Neural Network Learning

- Applications:
  --- image
  --- text
  --- speech
  --- tabular

ID	TOTAL ACTIONS	ACTION 1	ACTION 2	TOTAL TIME
10	120	80	40	0:50:05
11	255	130	125	1:40:03
12	180	100	80	1:20:19
13	305	205	100	1:58:58

- Three steps for learning

$y = f(x)$

Define function → goodness of function → Optimize for the best function

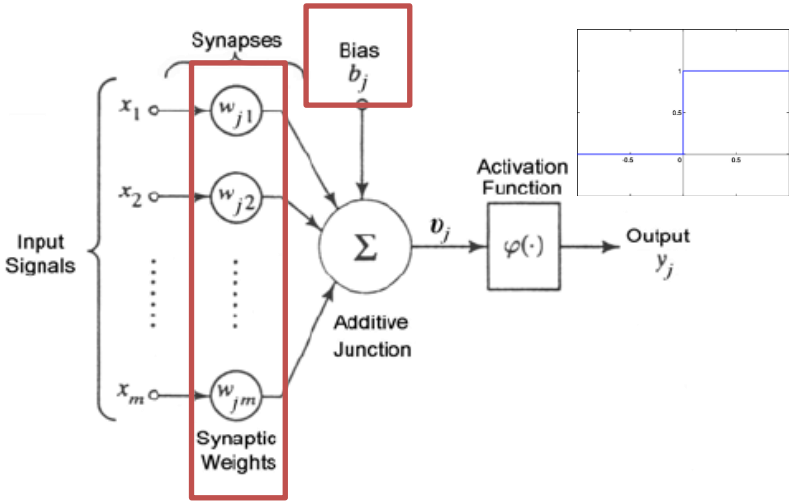
- neural networks
- decision tree

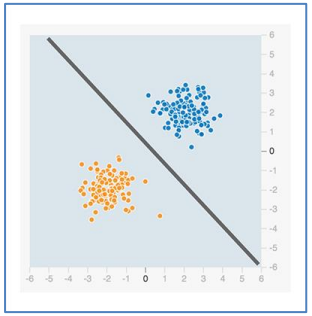
- Metrics:
  - Accuracy
  - Squared Error

- Data
- Optimization algorithm (back propagation)

Perceptron for Classification

- What we have :
  - Dataset (features, labels, training/testing/validation dataset)
  - Metrics: Classification error (higher accuracy ->better)
    - *error* $\sum_z d - y$
  - Perceptron structure





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

- Perceptron learning

--- To learn weights and bias

Perceptron

- Goal of optimizing weights and bias?

  important features <-> larger weights and bias

Identity	Location	Income	Occupation

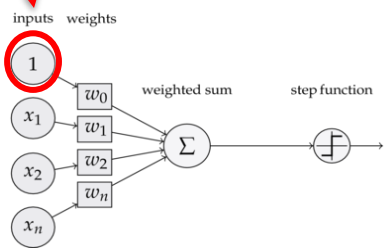
Credit card approval / not

- To simplify notation, we can transform the bias to a weight $w_{j0} = b_j$, by dummy an extra input node $x_0 = 1$

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

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

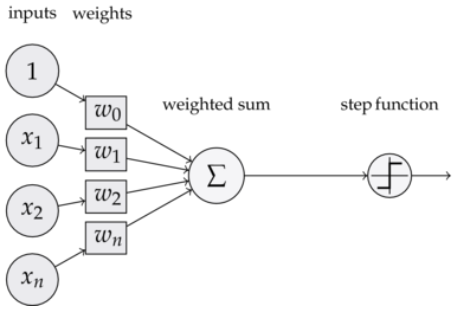
$$y_j = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} w_{ji}x_i > 0, \\ 0, & \text{otherwise} \end{cases}$$



Training a Perceptron

1. Get one instance from the dataset

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



2. Random initialize a set of weights $[w_0,w_1,w_2,w_3]=[0.1, 0.02, 0.1, -0.25]$

*3. Sum up *feature values and weights*, and get the predicted class label

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

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

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

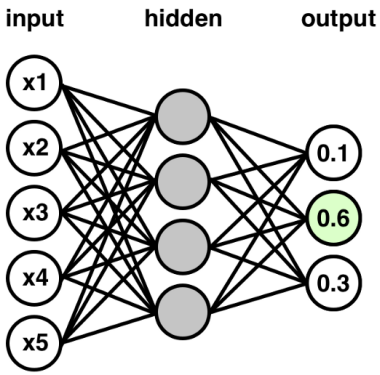
d -> desired label, *y* -> predicted label

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

*5. Go to step 3 for the next instance (until meet the stopping criterion)

NN for classification

- What we have
  - Dataset (features, labels, training/testing/validation dataset)
  - Metrics: Squared Error
    - total *error* $\sum_z (d_z - O_z)^2$
  - Pre-defined Neural network structure (features=input, labels=output, hidden layer =1, hidden size = 4)



- Train NN network

  - optimized NN network with high classification accuracy

Training a Neural Network

- **Initialise** the weights (randomly)

- **Forward propagation**
  - For each example/instance, calculate the predicted outputs using the current weights
  - Calculate the total *error/loss*

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

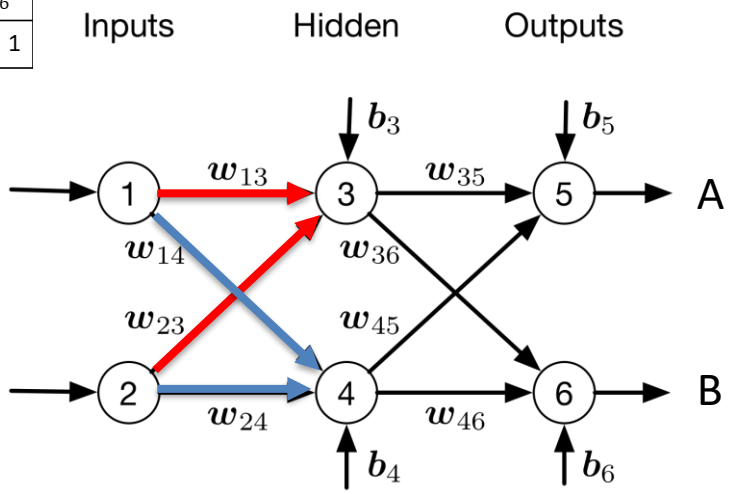
- Otherwise, **back propagation** to adjust the weights to make the error *smaller*.
  - Uses gradient descent (GD)

Feedforward NN Example

- Calculate the outputs from inputs (forward propagation)

X_1	X_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

d_5	d_6
0	1



$$z_j = \sum_i w_{ji} x_i + b_j$$

$Z_3 = w_{13}*X_1 + w_{23}*X_2 + b_3$
$= 0.72*0.90 + 0.10*(-0.2) + 0.01$
$= 0.64$

$Z_4 = w_{14}*X_1 + w_{24}*X_2 + b_4$
$= (-0.31)*0.90 + (-0.92)*(-0.2) + 0.38$
$= 0.29$

$$O_j = \varphi(z_j) = \frac{1}{1+e^{-z_j}}$$

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

Class = ? B

1. Weighted sum of a node z_j :
2. Calculate the output of a node O_j :
   - Assume φ is the sigmoid
3. Transmit the output to the next layer of neurons

$y_5 = O_5$
$y_6 = O_6$

total *loss* $\sum_z (d_z - O_z)^2$

Back Propagation (BP) Algorithm

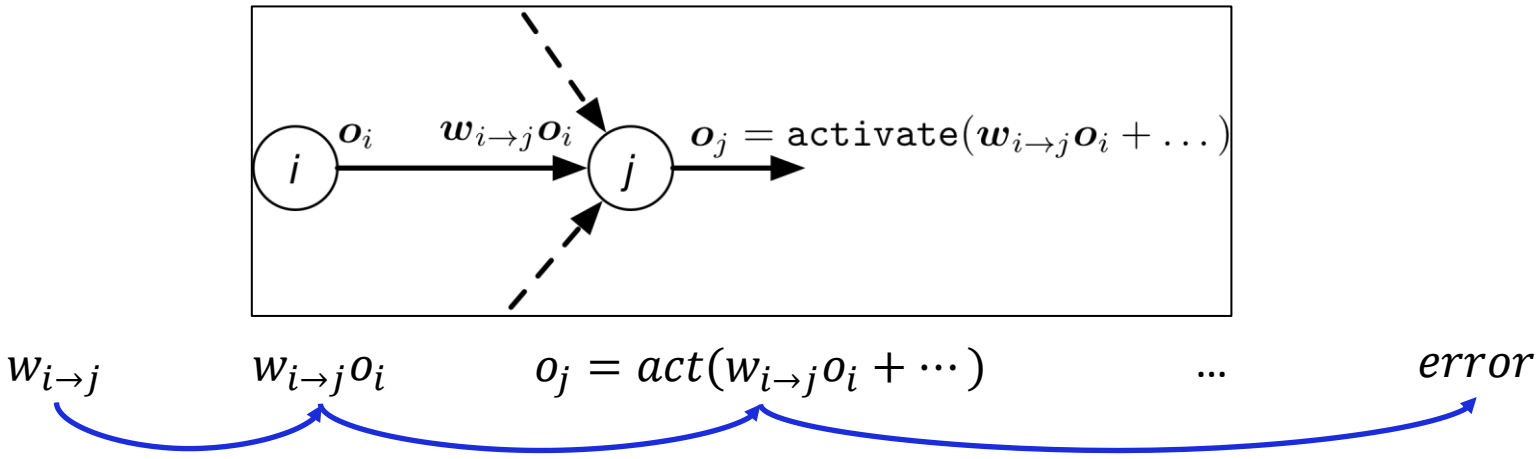
- **Weights updating (gradient descent)**

  – 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 proportional to its contribution to error-reducing
    - How big a change should we make a weight $w_{i\to j}$?
      – Big changes <-> big gradient = big contributions to the final error reducing

      $\Delta w_{i\to j} \sim \text{gradient}$
  – We calculate the gradient backward layer by layer from last layer to the first hidden layer

Back Propagation (BP) Algorithm



- When changing $w_{i\to j}$, the error change should be:
  - Proportional to the output: o_i
  - Proportional to the *slope* of the activation function at node j: $slope_j$
  - Proportional to error term of j (β_j)
- β_j -> "error term", is how "significant" for node j adjustment for the final error

BP Algorithm Implementation

- Initialise all weights (+bias) to small random values

- Until total error is small enough, repeat:
  - For each input example:
    - 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
    - Compute (+store) the weight changes for all weights

      $\Delta w_{i\to j} = \eta o_i o_j \left(1 - o_j\right) \beta_j$ (proportional to all 3 factors), Let η be the learning rate
  - Change weights

    $w_{i\to j} = w_{i\to j} + \Delta w_{i\to j}$

Weight Update Frequency

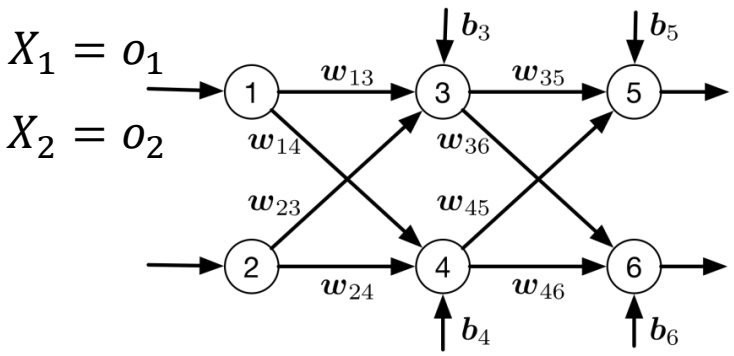
- For updating a weight w
- 4 training instances
- Online learning
  - Instance 1, calculate Δw_1, $w \mathrel{+}= \Delta w_1$
  - Instance 2, calculate Δw_2, $w \mathrel{+}= \Delta \mathrm{w}_2$
  - Instance 3, calculate Δw_3, $w \mathrel{+}= \Delta \mathrm{w}_3$
  - Instance 4, calculate Δw_4, $w \mathrel{+}= \Delta w_4$
- Offline learning
  - Instance 1, calculate Δw_1, w unchanged
  - Instance 2, calculate Δw_2, w unchanged
  - Instance 3, calculate Δw_3, w unchanged
  - Instance 4, calculate Δw_4, w unchanged
  - $w \mathrel{+}= (\Delta w_1 + \Delta w_2 + \Delta w_3 + \Delta w_4)$

BP NN Example

- Calculate the new weights and biases (backprop):

O_3	O_4	O_5	O_6	d_5	d_6	η	w_{13}	w_{14}	w_{23}	w_{24}	w_{35}	w_{36}	w_{45}	w_{46}
0.65	0.57	0.62	0.82	0	1	0.1	0.72	-0.31	0.10	-0.92	-0.37	0.43	-0.19	0.78

X_1	X_2
0.90	-0.20



$$\beta_z = d_z - o_z$$

Output layer

$$\beta_j = \sum_k w_{j\to k} o_k (1 - o_k) \beta_k$$

Hidden layer

$$\Delta w_{i\to j} = \eta o_i o_j (1 - o_j) \beta_j$$

$$\beta_5 = d_5 - o_5 = 0 - 0.62 = -0.62$$

$$\beta_6 = d_6 - o_6 = 1 - 0.82 = 0.18$$

$\Delta w_{35} = \eta o_3 o_5 (1 - o_5) \beta_5$=0.1* 0.65 * 0.62* (1-0.62)* (-0.62) = -0.0095

$\beta_3 = w_{3\to 5} o_5 (1 - o_5) \beta_5 + w_{3\to 6} o_6 (1 - o_6) \beta_6$

= -0.37*0.62* (1-0.62)* (-0.62) + 0.43* 0.82*(1-0.82)*0.18 =

0.0540 + 0.0114 = 0.0654

$\Delta w_{13} = \eta o_1 o_3 (1 - o_3) \beta_3$=0.1* 0.9* 0.65* (1-0.65)* 0.0654 = 0.0013

NN learning Example

(1) Feed Forward Pass

$I_1, \quad O_1 = I_1 ;$

$I_2, \quad O_2 = I_2 ;$

$I_3 = O_1 * W_{13} + O_2 * W_{23} + b_3, \quad O_3 = f(I_3) = \frac{1}{1+e^{-I_3}} ;$

$I_4 = O_1 * W_{14} + O_2 * W_{24} + b_4, \quad O_4 = f(I_4) = \frac{1}{1+e^{-I_4}} ;$

$I_5 = O_3 * W_{35} + O_4 * W_{45} + b_5, \quad O_5 = f(I_5) = \frac{1}{1+e^{-I_5}} ;$

$I_6 = O_3 * W_{36} + O_4 * W_{46} + b_6, \quad O_6 = f(I_6) = \frac{1}{1+e^{-I_6}} ;$

(2) Back Propagation:

$\beta_5 = d_5 - O_5 ;$

$\beta_6 = d_6 - O_6 ;$

$\beta_3 = W_{35} * O_5 * (1 - O_5) * \beta_5 + W_{36} * O_6 * (1 - O_6) * \beta_6 ;$

$\beta_4 = W_{45} * O_5 * (1 - O_5) * \beta_5 + W_{46} * O_6 * (1 - O_6) * \beta_6 ;$

$\Delta W_{35} = \eta * O_3 * O_5 * (1 - O_5) * \beta_5 ;$

$\Delta W_{36} = \eta * O_3 * O_6 * (1 - O_6) * \beta_6 ;$

$\Delta W_{45} = \eta * O_4 * O_5 * (1 - O_5) * \beta_5 ;$

$\Delta W_{46} = \eta * O_4 * O_6 * (1 - O_6) * \beta_6 ;$

$\Delta W_{13} = \eta * O_1 * O_3 * (1 - O_3) * \beta_3 ;$

$\Delta W_{14} = \eta * O_1 * O_4 * (1 - O_4) * \beta_4 ;$

$\Delta W_{23} = \eta * O_2 * O_3 * (1 - O_3) * \beta_3 ;$

$\Delta W_{24} = \eta * O_2 * O_4 * (1 - O_4) * \beta_4 ;$

$\Delta b_5 = \eta * O_5 * (1 - O_5) * \beta_5 ;$

$\Delta b_6 = \eta * O_6 * (1 - O_6) * \beta_6 ;$

$\Delta b_3 = \eta * O_3 * (1 - O_3) * \beta_3 ;$

$\Delta b_4 = \eta * O_4 * (1 - O_4) * \beta_4 ;$

(3) Update Weights

$W_{13} = W_{13} + \Delta W_{13} ;$

$W_{14} = W_{14} + \Delta W_{14} ;$

$W_{23} = W_{23} + \Delta W_{23} ;$

$W_{24} = W_{24} + \Delta W_{24} ;$

$W_{35} = W_{35} + \Delta W_{35} ;$

$W_{36} = W_{36} + \Delta W_{36} ;$

$W_{45} = W_{45} + \Delta W_{45} ;$

$W_{46} = W_{46} + \Delta W_{46} ;$

$b_3 = b_3 + \Delta b_3 ;$

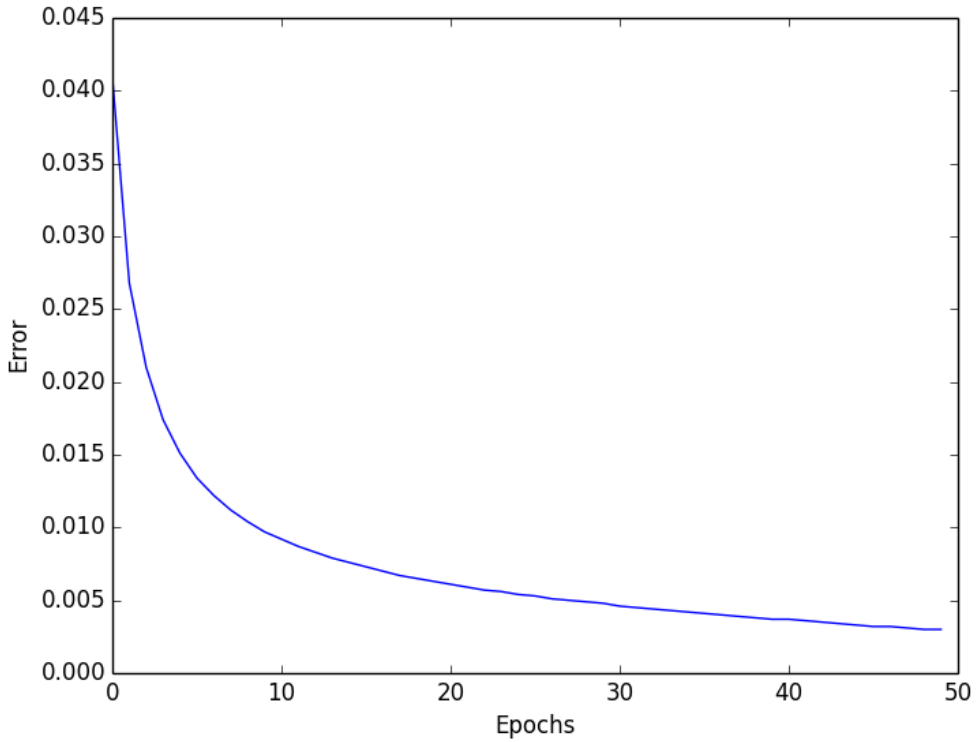
$b_4 = b_4 + \Delta b_4 ;$

$b_5 = b_5 + \Delta b_5 ;$

$b_6 = b_6 + \Delta b_6 ;$

Notes on BP Algorithm

- *1 Epoch*: all input examples (entire training set, batch, …)
- Training may require *thousands* of epochs. A convergence curve will help to decide when to stop (over-fitting?)



Summary

- Perceptron
- Feedforward NN
  - Forward propagation
  - Back Propagation

- Next week

  --- Neural Engineering

  --- Evolutionary Computation