COMP24112: Machine Learning

Chapter 7: Artificial Neural Network II

Dr. Tingting Mu

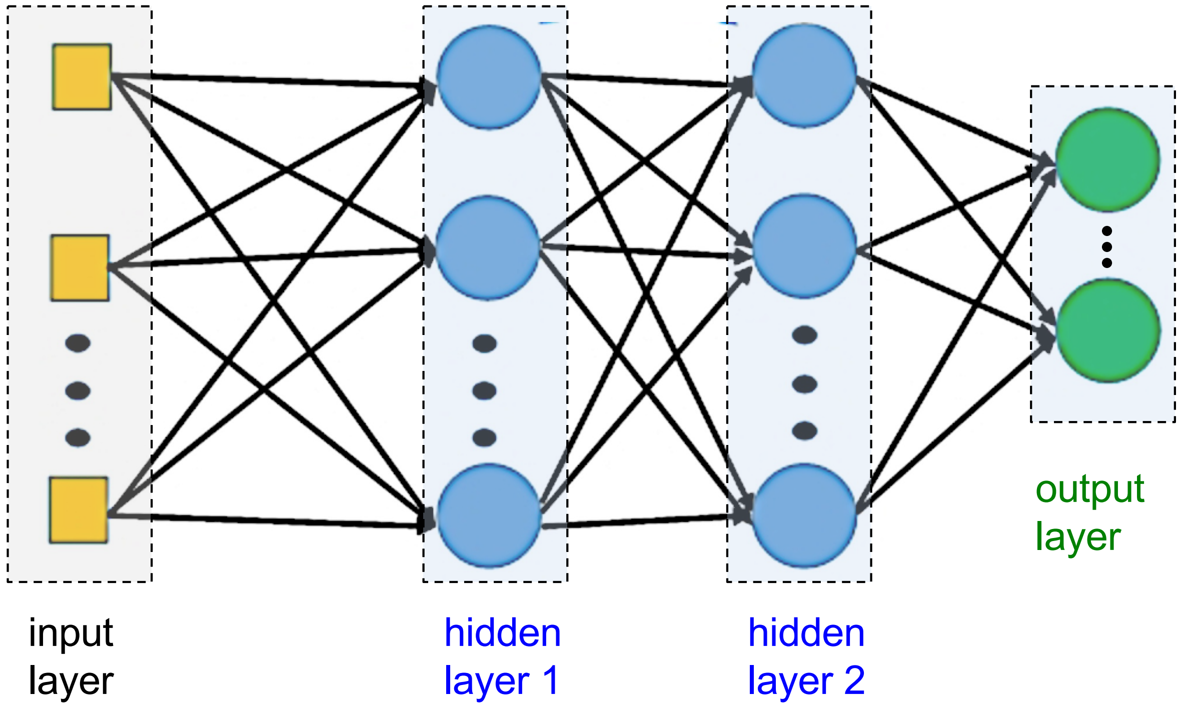
Email: tingting.mu@manchester.ac.uk

Content

- Neural network training
  - Hebbian learning
  - Gradient based
- The perceptron algorithm

Neural Network Training

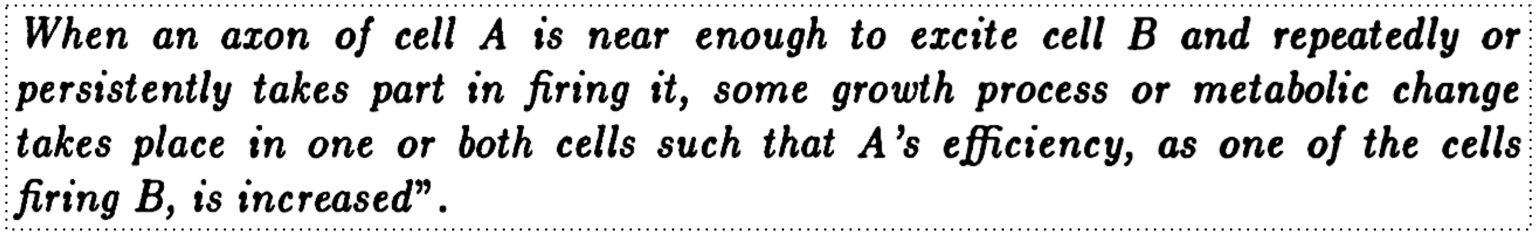
- Neural network training is the process of finding the optimal setting of the neural network weights.



MANCHESTER
1824
The University of Manchester

Hebbian Learning Rule

- Purely inspired by biological discovery (neuroscientific theory) by Donald Hebb, 1949
  - Original description:



When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased".

  - Re-formulated rules:
    - ◎ If the neurons on either side of a synapse are **activated simultaneously**, then the strength of that **synapse** should be selectively **increased**.
    - ◎ If the above activation is **asynchronous**, then that **synapse** is **weakened or eliminated**.

Hebbian Learning Rule

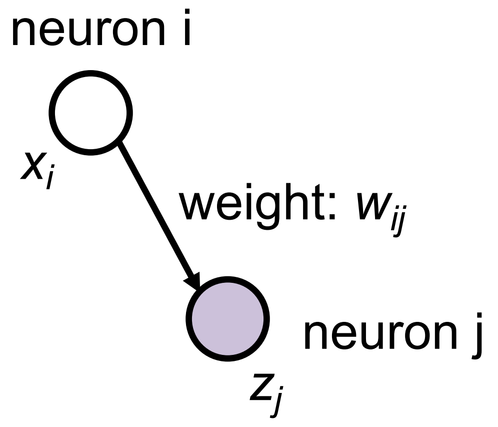
- The weight change is formulated as

$$\Delta w_{ij} = F(x_i, z_j)$$

- The simplest form of the function *F*

$$\Delta w_{ij} = \eta x_i z_j$$

- Can be unstable.



For this neuron:
pre-synaptic signal: x_i
post synaptic signal: z_j

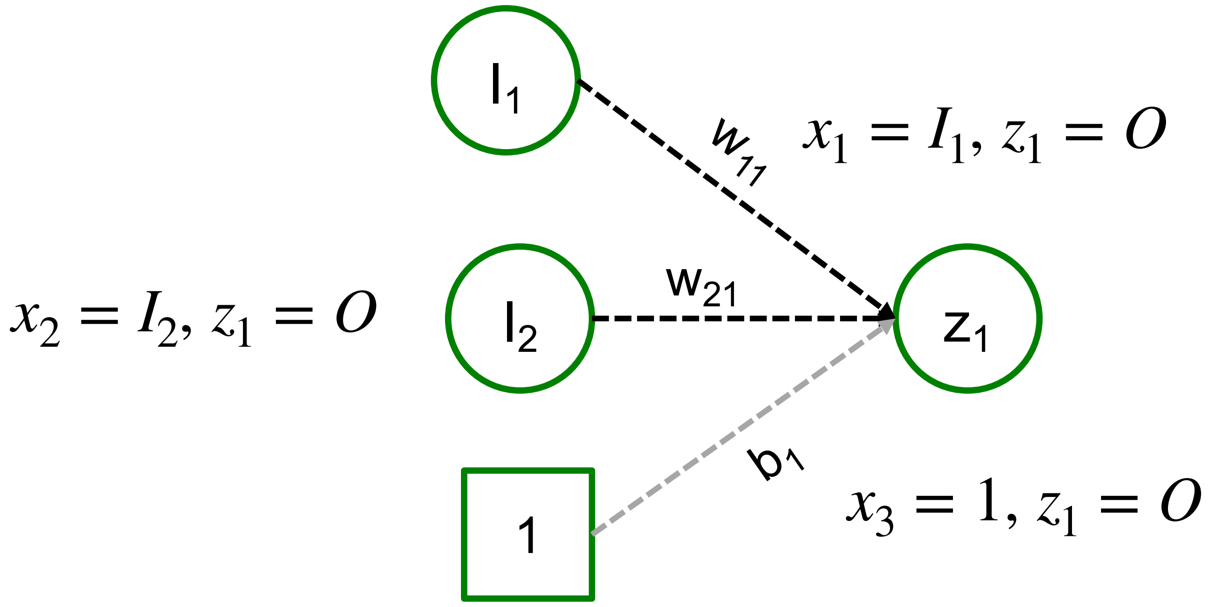
Example

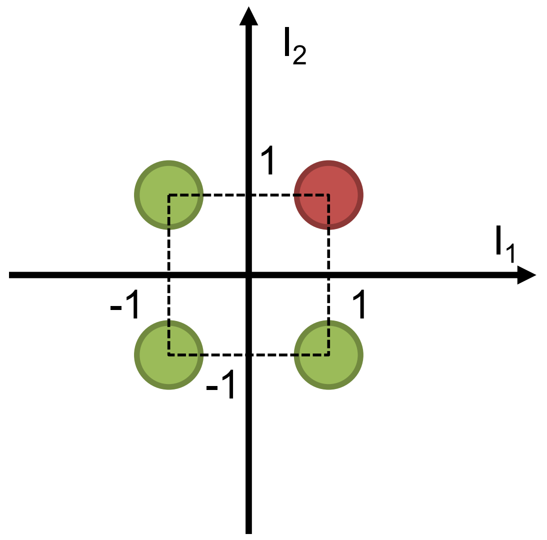
- Apply Hebbian learning rule to train a two-input one-output single layer perceptron to simulate AND gate. **Update the weights using one sample in every iteration, with $\eta = 1$.**

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)} = w_{ij}^{(t-1)} + \eta x_i z_j$$

Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

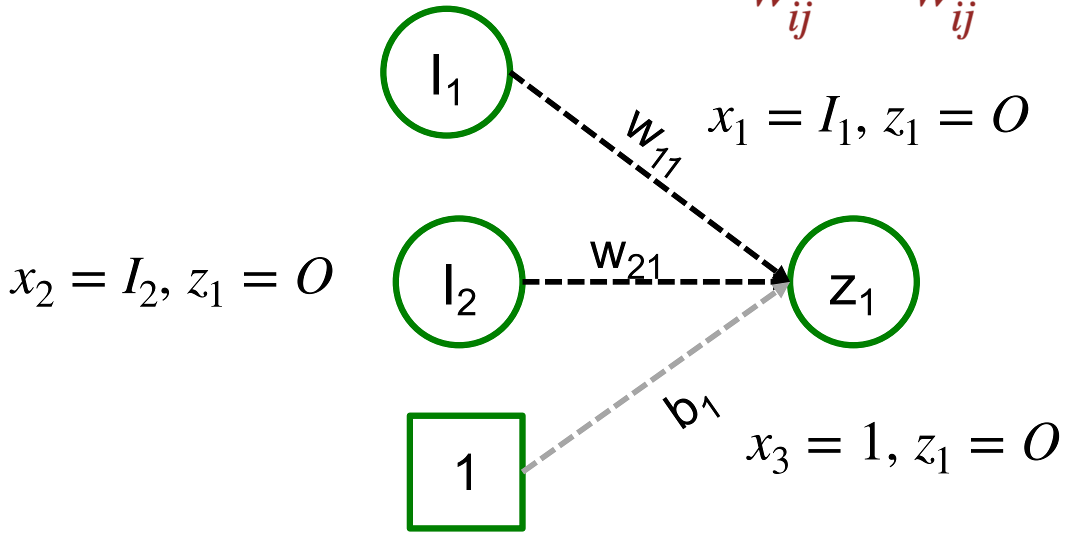
- Three network weights to decide: w_{11}, w_{12}, b





Example: Update I

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)} = w_{ij}^{(t-1)} + \eta x_i z_j$$



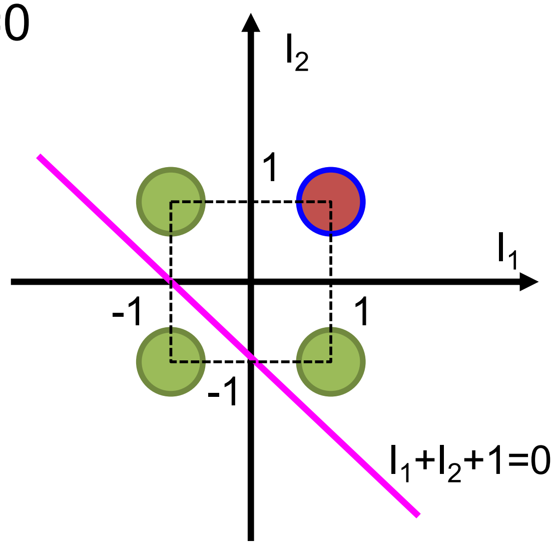
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: w_{11}=0, w_{12}=0, b=0
- **Update using sample (1, 1) - input, 1- output**

$$w_{11}^{(1)} = w_{11}^{(0)} + I_1 z_1 = 0 + 1 \times 1 = 1$$

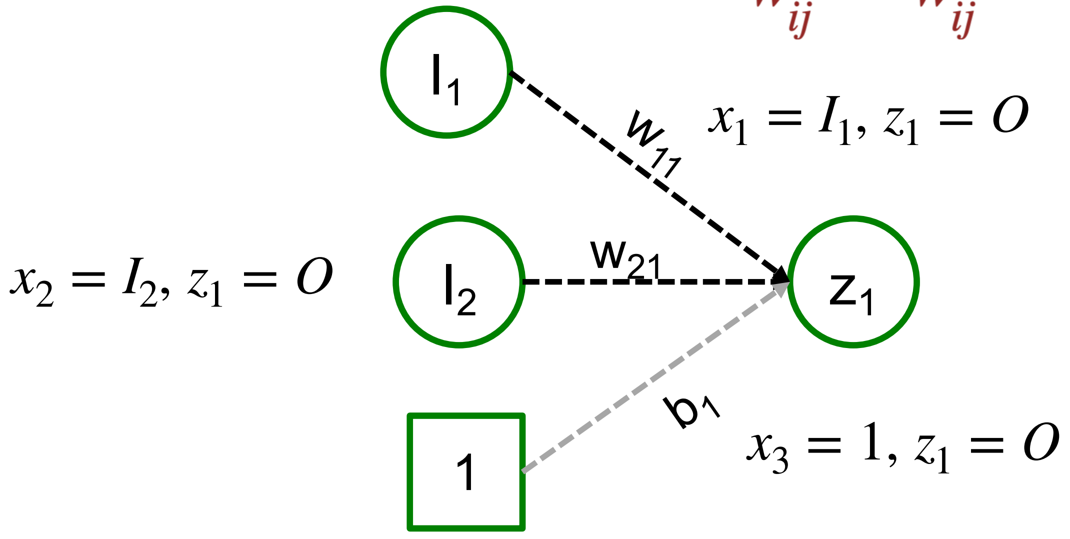
$$w_{21}^{(1)} = w_{21}^{(0)} + I_2 z_1 = 0 + 1 \times 1 = 1$$

$$b_1^{(1)} = b_1^{(0)} + 1 \times z_1 = 0 + 1 \times 1 = 1$$



Example: Update I

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)} = w_{ij}^{(t-1)} + \eta x_i z_j$$



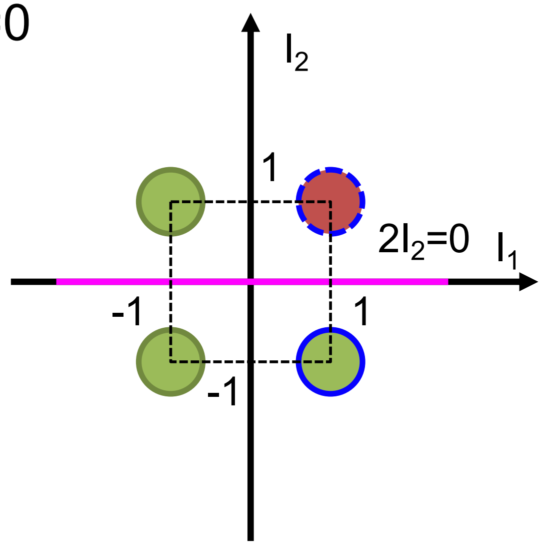
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: w_{11}=0, w_{12}=0, b=0
- **Update using sample (1, -1) - input, -1- output**

$$w_{11}^{(2)} = w_{11}^{(1)} + I_1 z_1 = 1 + 1 \times (-1) = 0$$

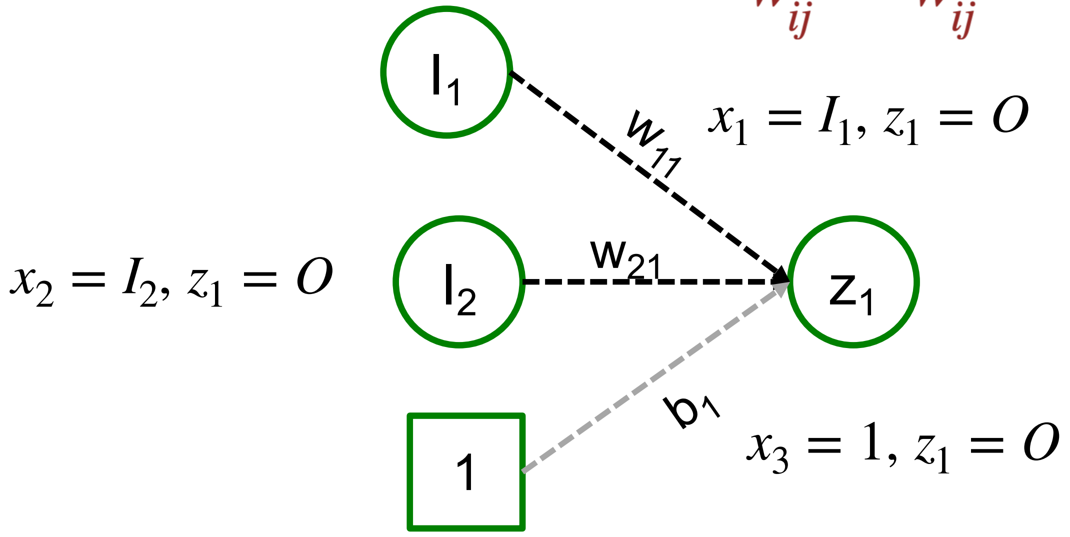
$$w_{21}^{(2)} = w_{21}^{(2)} + I_2 z_1 = 1 + (-1) \times (-1) = 2$$

$$b_1^{(2)} = b_1^{(1)} + 1 \times z_1 = 1 + 1 \times (-1) = 0$$



Example: Update I

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)} = w_{ij}^{(t-1)} + \eta x_i z_j$$



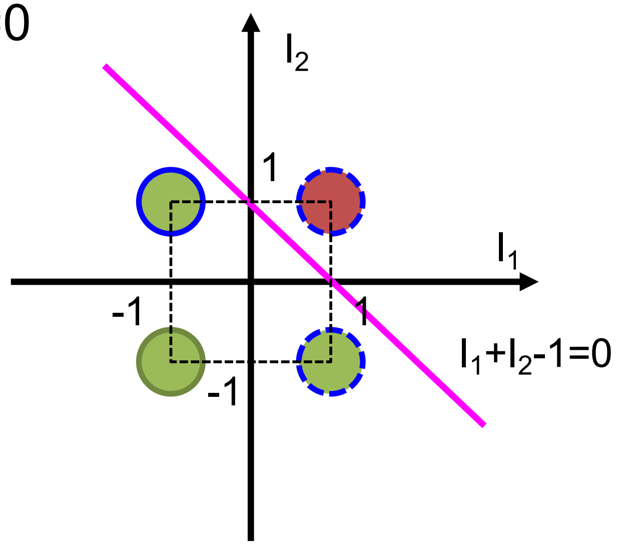
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: w_{11}=0, w_{12}=0, b=0
- **Update using sample (-1, 1) - input, -1- output**

$$w_{11}^{(3)} = w_{11}^{(2)} + I_1 z_1 = 0 + (-1) \times (-1) = 1$$

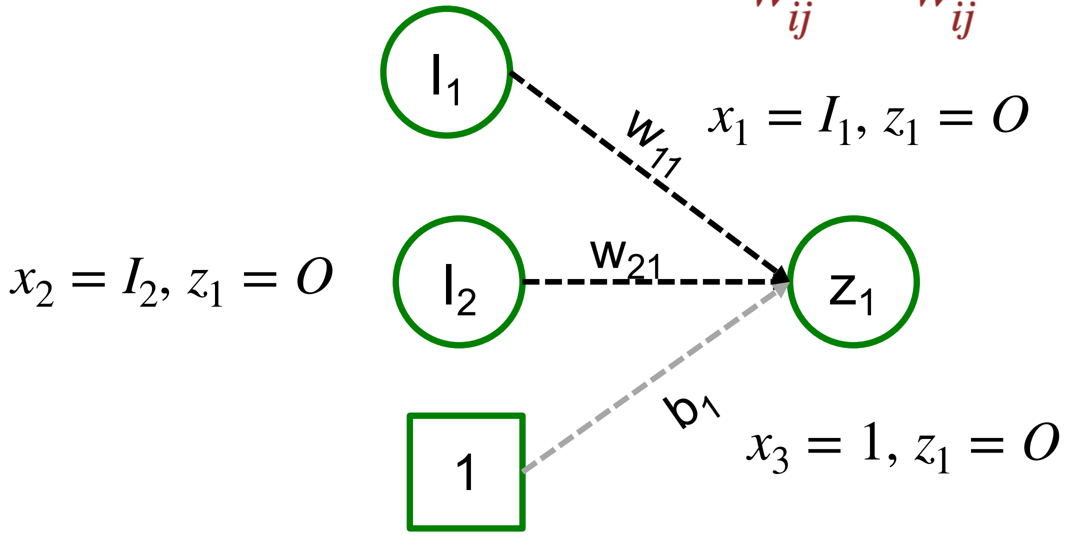
$$w_{21}^{(3)} = w_{21}^{(2)} + I_2 z_1 = 2 + 1 \times (-1) = 1$$

$$b_1^{(3)} = b_1^{(2)} + 1 \times z_1 = 0 + 1 \times (-1) = -1$$



Example: Update I

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)} = w_{ij}^{(t-1)} + \eta x_i z_j$$



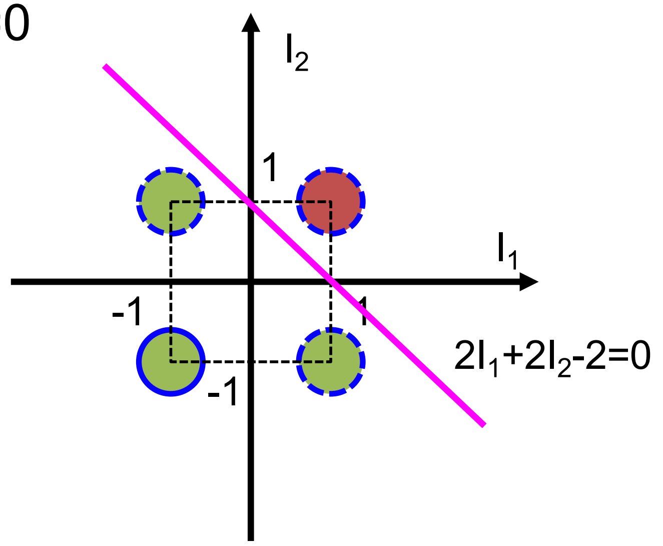
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: w_{11}=0, w_{12}=0, b=0
- **Update using sample (-1, -1) – input, -1-output**

$$w_{11}^{(4)} = w_{11}^{(3)} + I_1 z_1 = 1 + (-1) \times (-1) = 2$$

$$w_{21}^{(4)} = w_{21}^{(3)} + I_2 z_1 = 1 + (-1) \times (-1) = 2$$

$$b_1^{(4)} = b_1^{(3)} + 1 \times z_1 = -1 + 1 \times (-1) = -2$$



Example: Update II

Now change sample order used for updating the weights!

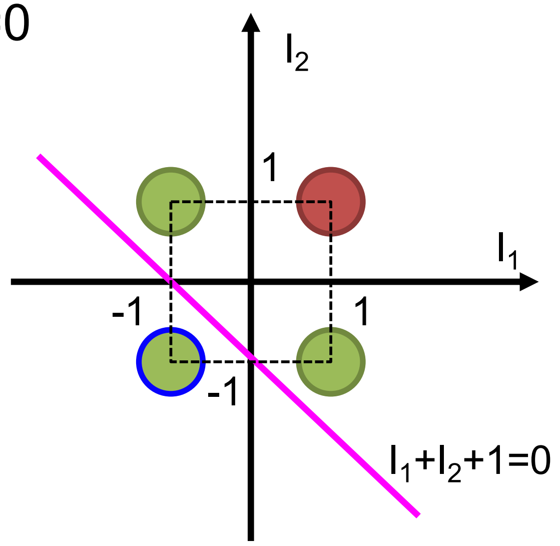
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: w_{11}=0, w_{12}=0, b=0
- **Update using sample (-1, -1) - input, 1-output**

$$w_{11}^{(1)} = w_{11}^{(0)} + I_1 z_1 = 0 + (-1) \times 1 = -1$$

$$w_{21}^{(1)} = w_{21}^{(0)} + I_2 z_1 = 0 + (-1) \times 1 = -1$$

$$b_1^{(1)} = b_1^{(0)} + 1 \times z_1 = 0 + 1 \times 1 = 1$$



Example: Update II

Now change sample order used for updating the weights!

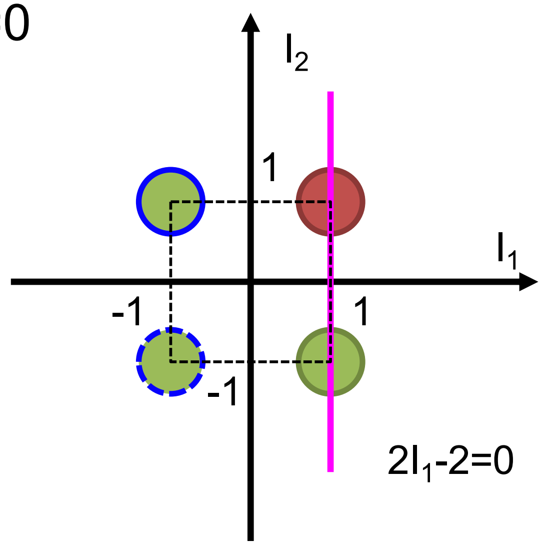
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: $w_{11}=0$, $w_{12}=0$, $b=0$
- **Update using sample (-1, 1) – input, -1-output**

$$w_{11}^{(2)} = w_{11}^{(1)} + I_1 z_1 = 1 + (-1) \times (-1) = 2$$

$$w_{21}^{(2)} = w_{21}^{(2)} + I_2 z_1 = 1 + 1 \times (-1) = 0$$

$$b_1^{(2)} = b_1^{(1)} + 1 \times z_1 = -1 + 1 \times (-1) = -2$$



Example: Update II

Now change sample order used for updating the weights!

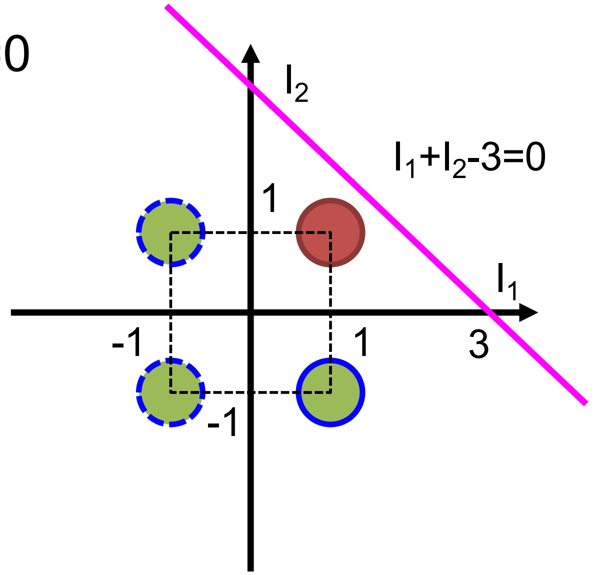
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: $w_{11}=0$, $w_{12}=0$, $b=0$
- **Update using sample (1, -1) – input, -1-output**

$$w_{11}^{(3)} = w_{11}^{(2)} + I_1 z_1 = 2 + 1 \times (-1) = 1$$

$$w_{21}^{(3)} = w_{21}^{(2)} + I_2 z_1 = 0 + (-1) \times (-1) = 1$$

$$b_1^{(3)} = b_1^{(2)} + 1 \times z_1 = -2 + 1 \times (-1) = -3$$



Example: Update II

Now change sample order used for updating the weights!

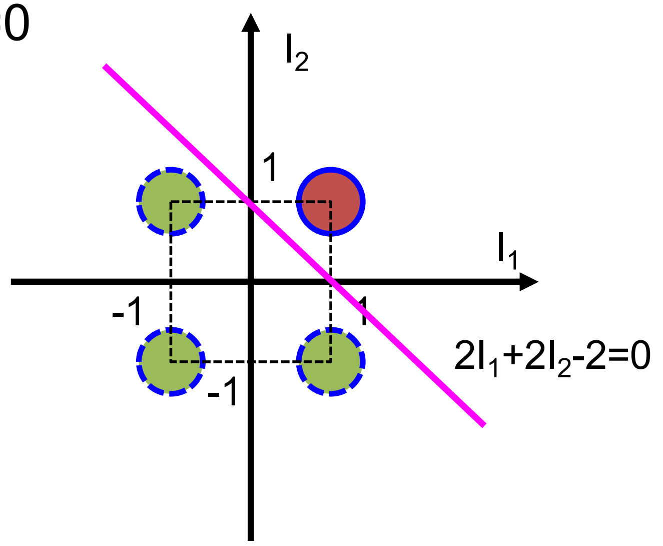
Input 1	Input 2	Output
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Start from a random weight guess: $w_{11}=0$, $w_{12}=0$, $b=0$
- **Update using sample (1, 1) – input, 1-output**

$$w_{11}^{(4)} = w_{11}^{(3)} + I_1 z_1 = 1 + 1 \times 1 = 2$$

$$w_{21}^{(4)} = w_{21}^{(3)} + I_2 z_1 = 1 + 1 \times 1 = 2$$

$$b_1^{(4)} = b_1^{(3)} + 1 \times z_1 = -3 + 1 \times 1 = -2$$



Gradient Based Training

A commonly used approach:

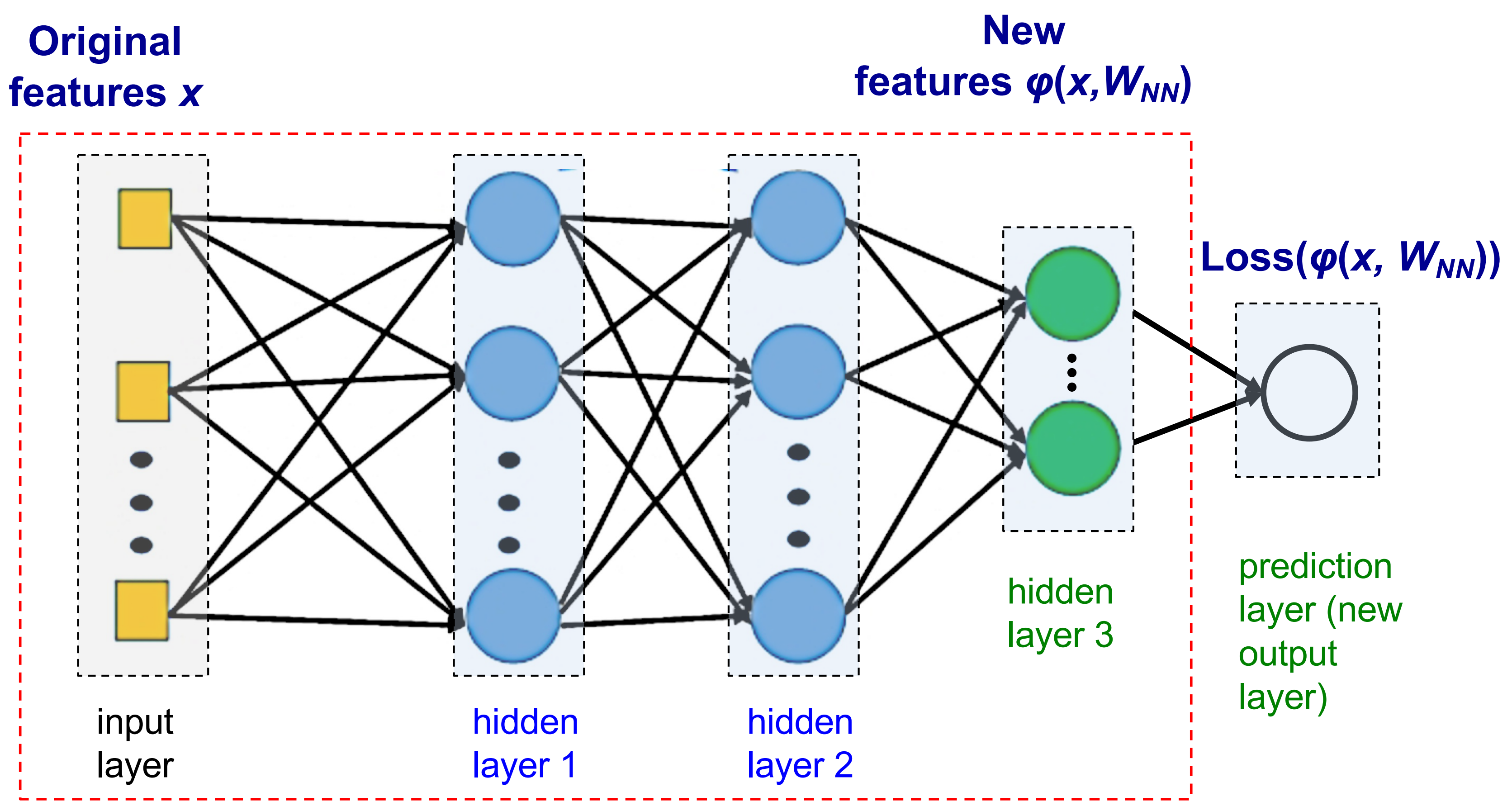
- Treat $\boldsymbol{z}=\boldsymbol{\varphi}(\boldsymbol{x},\boldsymbol{W}_{NN})$ as the new features, to be used as the input of a linear predictor.

- A loss function is minimised to find the optimal neural network weights $\boldsymbol{W}_{NN}$ together with the weights of the linear predictor $\boldsymbol{W}_P$.

- Training (optimisation) methods: stochastic gradient descent, mini-batch gradient descent.

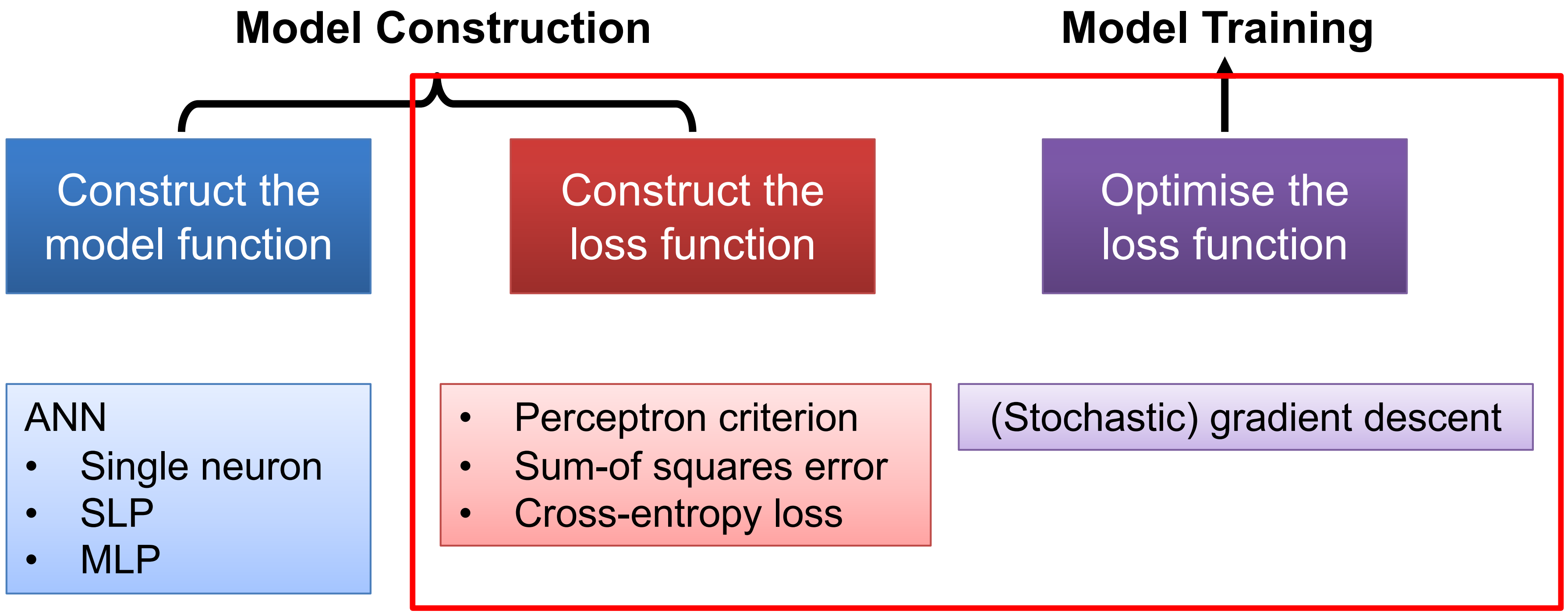
- More accurate and stable than Hebbian learning rules in practice.

Architecture Illustration


Original features x
New features $\varphi(x, W_{NN})$
$\text{Loss}(\varphi(x, W_{NN}))$
input layer
hidden layer 1
hidden layer 2
hidden layer 3
prediction layer (new output layer)

A neural network can be viewed as a powerful feature extractor to compute an effective representation for the sample, which helps the prediction task.

Training Pipeline


Model Construction
Model Training
Construct the model function
Construct the loss function
Optimise the loss function

ANN
- Single neuron
- SLP
- MLP

- Perceptron criterion
- Sum-of squares error
- Cross-entropy loss

(Stochastic) gradient descent

- **Case Study: Perceptron Algorithm**

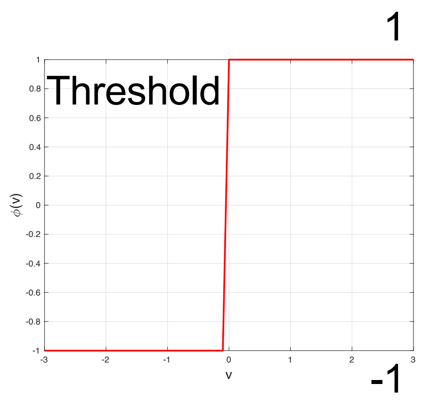
Training a single neuron model by minimising the perceptron criterion using stochastic gradient descent.

$$y = \varphi\left(\sum_{i=1}^{d} w_i x_i + b\right)$$

Perceptron of Rosenblatt

- When the activation function is set as the threshold function, the model is still linear, and it is known as the perceptron of Rosenblatt.

$$\text{output} = \begin{cases} 1 & \text{if} \quad \mathbf{w}^T\tilde{\mathbf{x}} \geq 0 \\ -1 & \text{if} \quad \mathbf{w}^T\tilde{\mathbf{x}} < 0 \end{cases}$$



Activation function:

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ -1 & if \quad v < 0 \end{cases}$$

- The perceptron algorithm was invented in 1958 by Frank Rosenblatt (1928-1971, Psychologist).
- The perceptron algorithm is for binary classification, and it occupies an import place in the history of pattern recognition algorithms.

The Perceptron Algorithm

- Update the weights using one sample at a time:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta \boxed{y_i \tilde{\mathbf{x}}_i}, \quad \eta > 0 \text{ is learning rate.}$$

Update using a **misclassified** sample in each iteration!

- An example implementation:

```
Initialise the weights (stored in $w^{(0)}$) to random numbers in range -1 to +1.
For t = 1 to NUM_ITERATIONS
    For each training sample ($x_i$,$y_i$)
        Calculate activation using current weight (stored in $w^{(t)}$).
        If the sample is misclassified, update weight (stored in $w^{(t+1)}$) by
        the above perceptron learning rule.
    end
end
```

Perceptron Criterion

- **Perceptron criterion** assesses classification error.

$$O(\mathbf{w}) = -\sum_{i \in \text{Misclassified Set}} y_i\left(\mathbf{w}^T\tilde{\mathbf{x}}_i\right)$$

If a sample is correctly classified, applies an error penalty of zero; if incorrectly classified, applies an error penalty of the following quantity:

$$\boxed{\left|y_i\left(\mathbf{w}^T\tilde{\mathbf{x}}_i\right)\right| = -y_i\left(\mathbf{w}^T\tilde{\mathbf{x}}_i\right)}$$

- The perceptron algorithm can be derived by minimising the perceptron criterion computed using the training samples.

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) is used to minimise the perceptron criterion.
- Calculate the error using one **misclassified** sample:

$$O_i(\mathbf{w}) = -y_i \mathbf{w}^T \tilde{\mathbf{x}}_i$$

- Calculate its gradient: $\dfrac{\partial O_i(\mathbf{w})}{\partial \mathbf{w}} = -y_i \tilde{\mathbf{x}}_i$

- SGD update: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta y_i \tilde{\mathbf{x}}_i$

Example

MANCHESTER 1824
The University of Manchester

- Perceptron learning rule:

Update using one misclassified sample in each iteration:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta y_i \tilde{\mathbf{x}}_i$$

- What weight changes do the following cases produce?

if... (true label = -1, activation output = -1).... then	No change
if... (true label = +1, activation output = +1).... then	No change
if... (true label = -1, activation output = +1).... then	Add $-\eta\tilde{\mathbf{x}}_i$
if... (true label = +1, activation output = -1).... then	Add $+\eta\tilde{\mathbf{x}}_i$