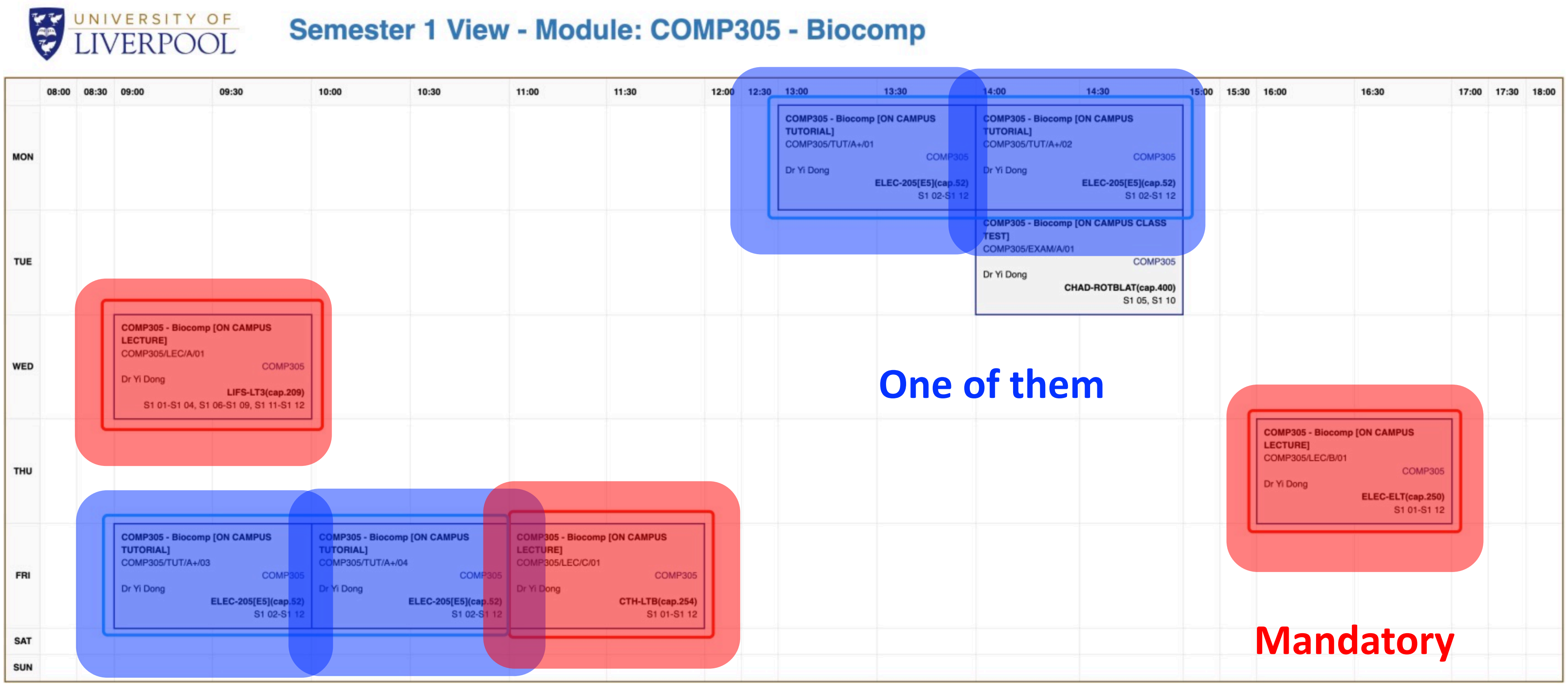
Comp305

Biocomputation

Lecturer: Yi Dong

Comp305 Module Timetable



There will be **26-30** lectures, thee per week. The lecture slides will appear on Canvas. Please use Canvas to access the lecture information. There will be **9** tutorials, one per week.

Lecture/Tutorial Rules

Questions are welcome as soon as they arise, because

1. Questions give feedback to the lecturer;

2. Questions help your understanding;

3. Your questions help your classmates, who might experience difficulties with formulating the same problems/doubts in the form of a question.

Comp305 Part I.

Artificial Neural Networks

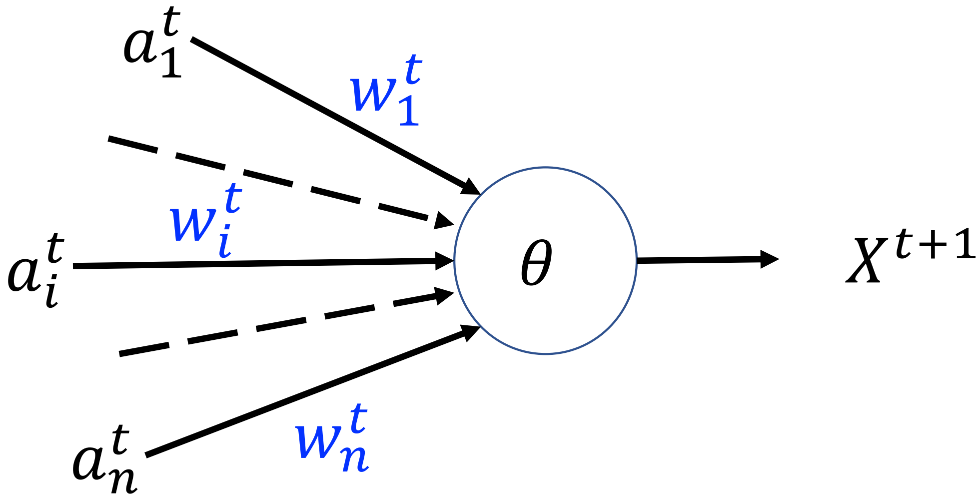
Topic 4.

Normalized Hebb’s Rule (Oja’s Rule)

Topic of Today’s Lecture

Oja’s rule and associative learning.

Hebb's Rule (1949)

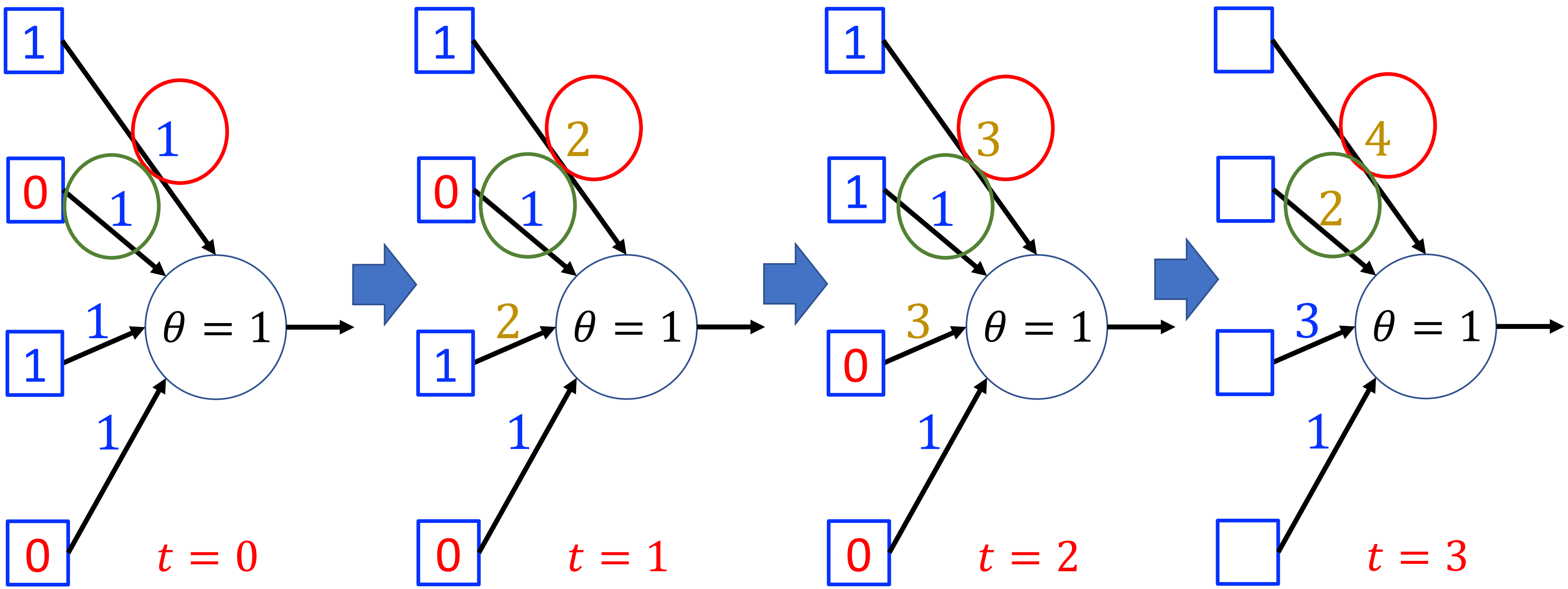


$$w_i^{t+1} = w_i^t + \Delta w_i^t,$$

Where

$$\Delta w_i^t = C a_i^t X^{t+1}$$

Example: w_1 and w_2



All the weights increase **monotonously**. Finally, each weight will become large enough such that any activated input can fire the neuron alone.

Normalized Hebb's Rule

$$w_i^{t+1} = w_i^t + \Delta w_i^t,$$

Where

$$\Delta w_i^t = C a_i^t X^{t+1}$$

Normalization:

$$\|\boldsymbol{w}^{\boldsymbol{t}}\| = \mathbf{1}$$

- Also known as **Oja's rule**.
- By normalization, the weights will not monotonously increase, but converge after, which reflects the **predisposition to different inputs**. It plays an important role in ***unsupervised learning*** or ***self-organisation***.

Formulation of Normalized Hebb's Rule (Oja's rule)

1. Set the neuron threshold value θ and the learning rate C.
2. Set ***random initial values*** for the weights of connections w_i^t.
3. **Normalization**.
4. Give instant input values a_i^t by the input units.
5. Compute the instant state of the neuron $S^t = \sum_i w_i^t a_i^t$
5. Compute the instant output of the neuron X^{t+1}

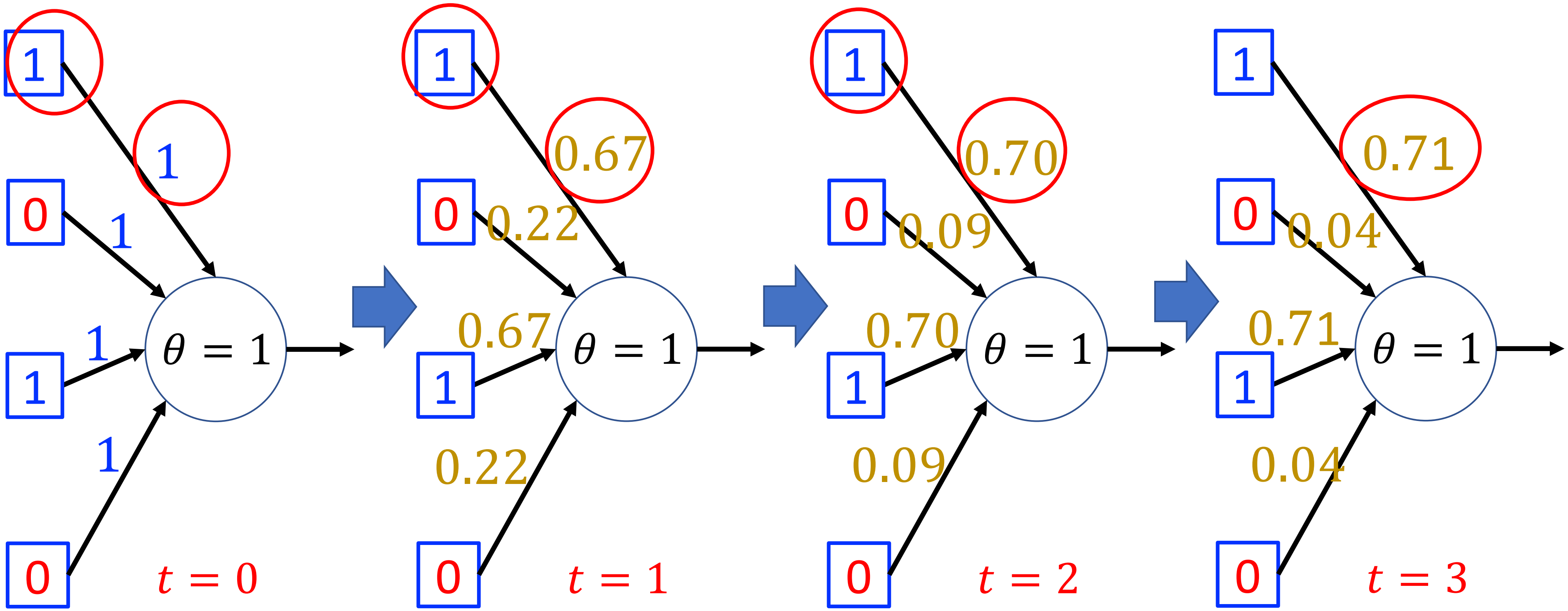
$$X^{t+1} = g(S^t) = H(S^t - \theta) = \begin{cases} 1, & S^t \geq \theta; \\ 0, & S^t < \theta. \end{cases}$$

6. Compute the instant corrections to the weights $\Delta w_i^t = C a_i^t X^{t+1}$
7. Update the weights of connections $w_i^{t+1} = w_i^t + \Delta w_i^t$
8. Go to the step 3.

Formulation of Normalized Hebb's Rule

1. Set the neuron threshold value θ and the learning rate C.
2. Set ***random initial values*** for the weights of connections w_i^t.
3. **Normalization.**
   I. Compute the 2-norm of the vector w^t
   $$\|w^t\|_2 = \sqrt{\sum_i \left(w_i^t\right)^2}$$
   II. Normalize the weight of each connection w_i^t
   $$w_i^t = \frac{1}{\|w^t\|_2} w_i^t$$
   III. Check the following **convergence criteria** with a given small positive real δ
   $$\max_i \left| w_i^t - w_i^{t-1} \right| \le \delta$$

An Example of **Normalized** Hebb's Rule



Intuition: Weights will **NOT** increase monotonously. The most frequent firing pattern will be remembered and finally dominate.

Why does Normalized Hebb's Rule Converge?

- The exact dynamics of the Oja's rule have been solved by Wyatt and Elfaldel 1995

- It shows that the weight $w \rightarrow \delta^*$, where δ^* is the eigenvector of the **largest eigenvalue $\boldsymbol{\alpha}^*$** of the matrix based on average $\bar{a}$.

- Without normalization (the original Hebb's rule) the weight of each connection grows exponentially with $\boldsymbol{\alpha_i}$. With normalization (Oja's rule) only the largest $\boldsymbol{\alpha}^*$ can survive.

Jr, JL Wyatt, and Ibrahim M. Elfadel. "Time-domain solutions of Oja's equations." Neural Computation 7.5 (1995): 915-922.

Observations of the Running Example

- The data set $D = \{[1,0,1,0]\}$. That is, we only use one input to train the neuron.

- Thus, the weights of the neuron $[0.71, 0.04, 0.71, 0.04]$ reflects the characteristic of this single input $[1,0,1,0]$.

Observations of the Running Example

- The data set $D = \{[1,0,1,0]\}$. That is, we only use one input to train the neuron.

- Thus, the weights of the neuron $[0.71,0.04,0.71,0.04]$ reflects the characteristic of this single input $[1,0,1,0]$.

$$C\bar{a}^T\bar{a} = 1 \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Where the largest eigenvalue of this matrix is 2, and the corresponding eigenvector with the 2-norm as 1 is $[0.7071,0,0.7071,0]$.

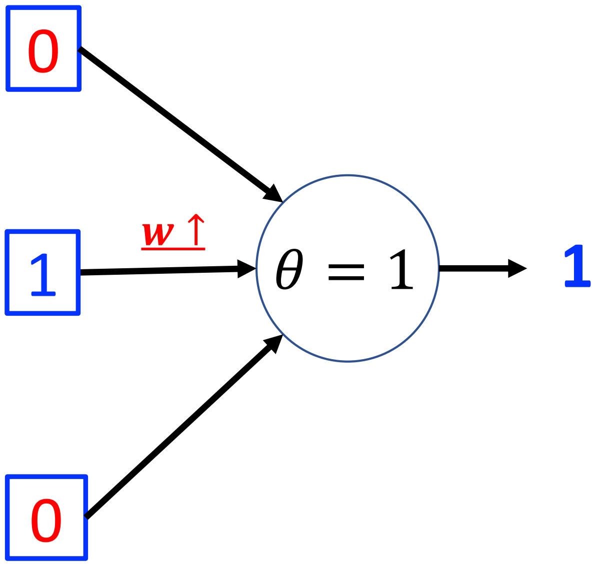
Observations of the Running Example

- The data set $D = \{[1,0,1,0]\}$. That is, we only use one input to train the neuron.

- Thus, the weights of the neuron $[0.71,0.04,0.71,0.04]$ reflects the characteristic of this single input $[1,0,1,0]$.

- The weight does not monotonically increase anymore. Instead, the more important the input is, the closer to 1 the corresponding weight gets.

Hebb’s Rules and Associative Learning

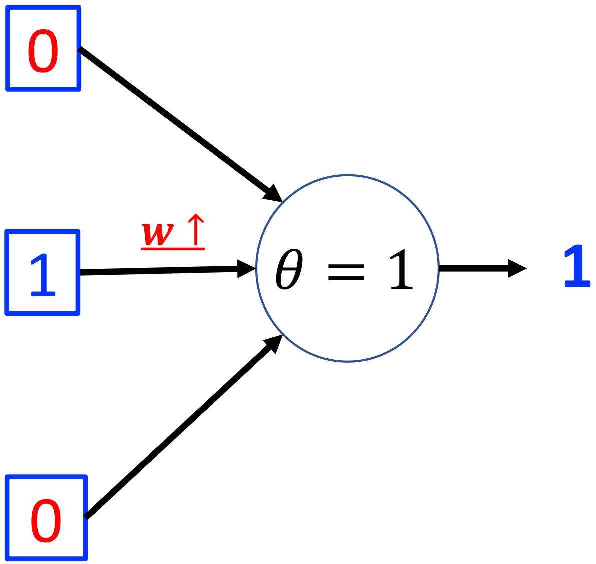


“… Cells that fire together, wire together…”

Applying *Hebb’s rules (including the original one and the normalized one)* on a neural network, we know that

the weight of connection increases between nodes having activated outputs simultaneously.

Hebb’s Rules and Associative Learning

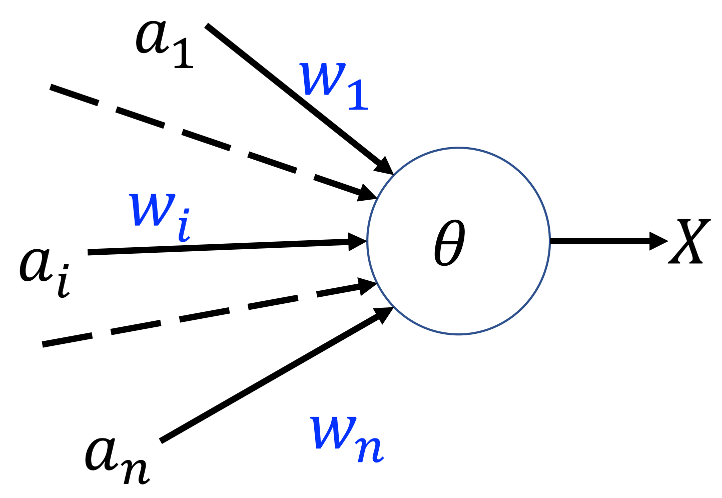


“… Cells that fire together, wire together…”

Thus,

the weight of connections between neurons eventually comes to represent the correlation between their outputs, and that fact is used to solve *association problems*.

Hebb's Rules and Associative Learning

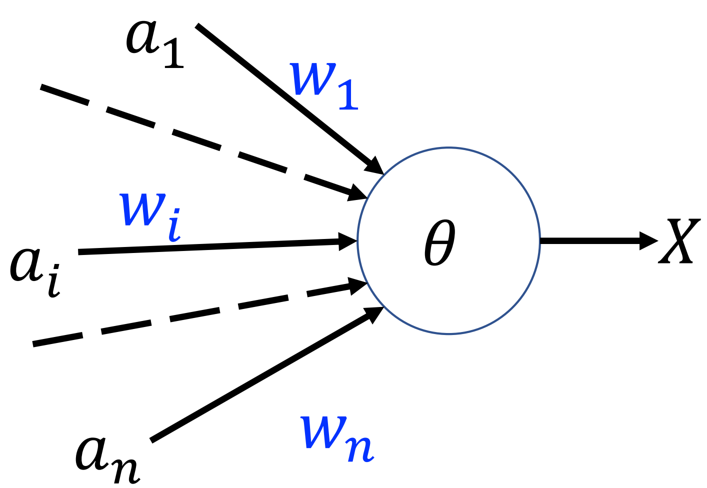


"… Cells that fire together, wire together…"

As we did before, let
$$a = (a_1, a_2, \cdots, a_n)$$
be the input vector.

Let
$$W = (w_1, w_2, \cdots, w_n)$$
be the weight vector of the connections between the input vector a and the output X.

Hebb's Rules and Associative Learning



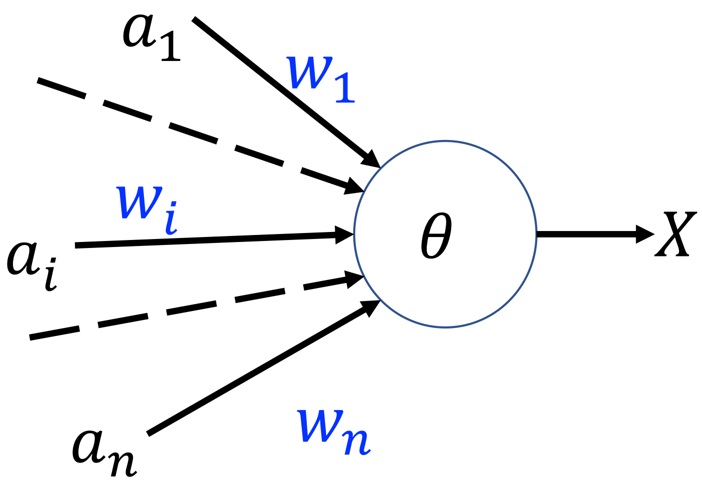
"… Cells that fire together, wire together…"

Input vector: $a = (a_1, a_2, \cdots, a_n)$
Weight vector: $W = (w_1, w_2, \cdots, w_n)$

Then the instant state of the output neuron is defined as dot product:

$$S = W \cdot a^T = \sum_{i=1}^{n} w_i a_i$$

Hebb's Rules and Associative Learning



a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

"... Cells that fire together, wire together..."

Input vector: $a = (a_1, a_2, \cdots, a_n)$
Weight vector: $W = (w_1, w_2, \cdots, w_n)$

Then the instant state of the output neuron is defined as dot product:

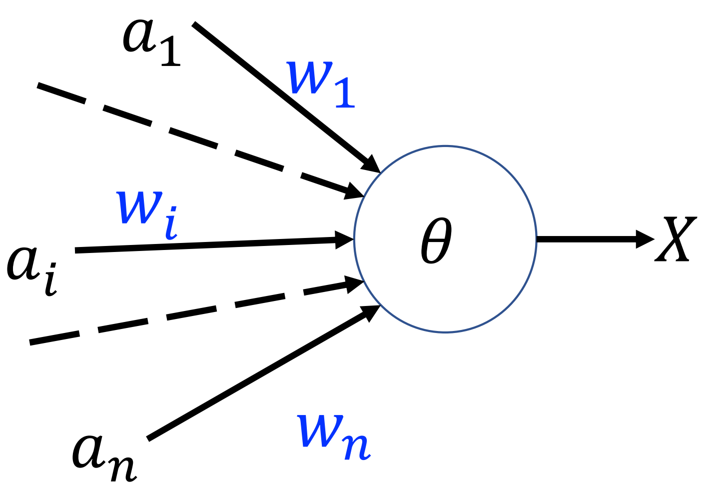
$$S = W \cdot a^T = \sum_{i=1}^{n} w_i a_i$$

This weighted sum will be greater if the vectors

***a* and *W* are similar,**

i.e., close to each other.

Hebb’s Rules and Associative Learning



a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

“… Cells that fire together, wire together…”

Input vector: $a = (a_1, a_2, \cdots, a_n)$
Weight vector: $W = (w_1, w_2, \cdots, w_n)$

Then the instant state of the output neuron is defined as dot product:

$$S = W \cdot a^T = \sum_{i=1}^{n} w_i a_i$$

This weighted sum will be greater if the vectors

***a* and *W* are similar,**

i.e., close to each other.

$\boxed{\text{Training data}} \longleftrightarrow \boxed{\text{Neuron (weight)}}$

Hebb's Rules and Associative Learning

Input:

a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

Weight (trained)

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

Weight (theory)

w_1^4	w_2^4	w_3^4	w_4^4
0.7071	0	0.7071	0

"… Cells that fire together, wire together…"

This weighted sum S will be greater if the vectors

a and W are similar,

i.e., close to each other.

Training data ⟷ Neuron (weight)

- As seen in previous lectures, training of a neuron with normalised Hebb's learning rule results in a vector of weights of connections similar to the training input vector.
- Thus, **the neuron "remembers" the training pattern via increasing weight of corresponding connections**.

Hebb's Rules and Associative Learning

Input:

a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

Weight (trained)

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

Weight (theory)

w_1^4	w_2^4	w_3^4	w_4^4
0.7071	0	0.7071	0

"… Cells that fire together, wire together…"

This weighted sum S will be greater if the vectors

a and W are similar,

i.e., close to each other.

Training data ⟷ Neuron (weight)

- If the trained network is presented with an input vector $\boldsymbol{a}'$ similar to the training one $\boldsymbol{a}$, it should be also similar to the weight of the network $\boldsymbol{W}$, and result in the similar neuron output.

Hebb’s Rules and Associative Learning

Input:

a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

Weight (trained)

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

Weight (theory)

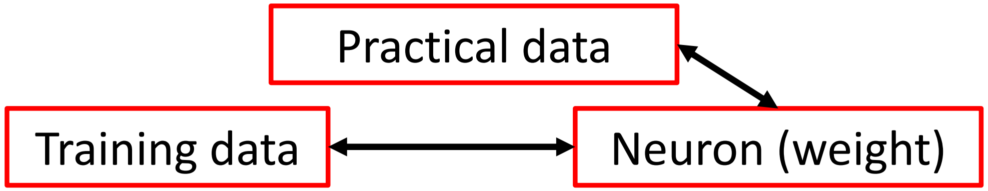
w_1^4	w_2^4	w_3^4	w_4^4
0.7071	0	0.7071	0

“… Cells that fire together, wire together…”

This weighted sum S will be greater if the vectors

a and W are similar,

i.e., close to each other.



- If the trained network is presented with an input vector $\boldsymbol{a}'$ similar to the training one $\boldsymbol{a}$, it should be also similar to the weight of the network $\boldsymbol{W}$, and result in the similar neuron output $\boldsymbol{a}$.

Hebb's Rules and Associative Learning

Input:

a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

Weight (trained)

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

Weight (theory)

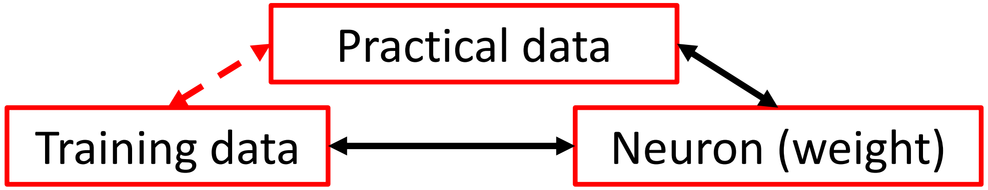
w_1^4	w_2^4	w_3^4	w_4^4
0.7071	0	0.7071	0

"… Cells that fire together, wire together…"

This weighted sum S will be greater if the vectors

a and W are similar,

i.e., close to each other.



- If the trained network is presented with an input vector $\boldsymbol{a'}$ similar to the training one $\boldsymbol{a}$, it should be also similar to the weight of the network $\boldsymbol{W}$, and result in the similar neuron output.
- One may say that , **the network "recognises" the new input or "associates" it with the training one.**

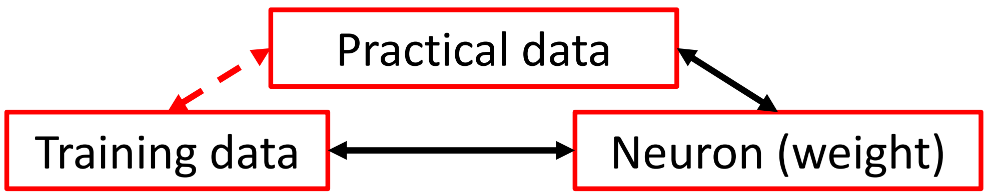
Associative Learning

"… Cells that fire together, wire together…"

This weighted sum S will be greater if the vectors

a and W are similar,

i.e., close to each other.

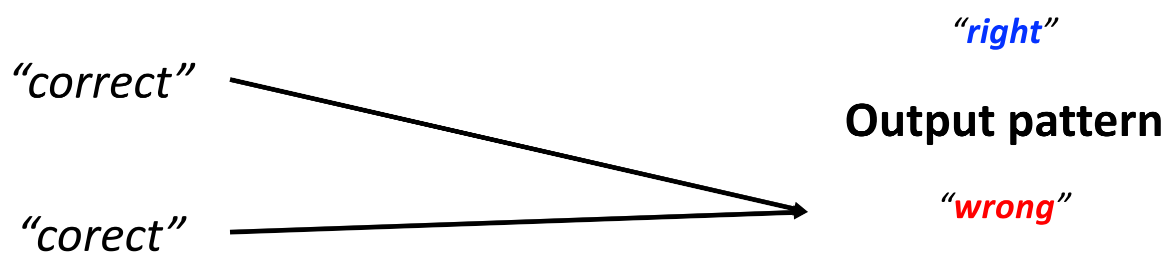


Definition: Association is the task of mapping patterns to patterns.

- An ***associative memory*** is to learn and remember the mapping between two unrelated pattens.
- For instance, a person may learn a word and its meaning before (learn the mapping between ***word*** and ***meaning***). When the person reads the word that is spelled incorrectly, she or he may know it has the same meaning with the correct word by association (remember the mapping).

Key Points

- We do not label any input in the data set during learning. The weight updating in each iteration only involves **the input, the output and the current weight**.
- We do not care what the output pattern means. We **care which inputs are considered similar** by the network. That is, **unsupervised** learning.



They point to the same pattern? ☺

Unsupervised Learning

- Unsupervised learning is a type of machine learning in which the algorithm **is not provided with any pre-assigned labels or scores for the training data**. As a result, unsupervised learning algorithms must first **self-discover any naturally occurring patterns** in that training data set.

- It will be much clearer when we extend the network with single output to the network with multiple outputs.

From Single Output to Multiple outputs

a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

When there are multiple outputs, weight vector becomes weight matrix.

$$w = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$$

n: number of inputs
m: number of outputs

What can we expect for a multi-output network?

- For a single output network, we can strengthen the link between an input pattern and an output by normalized Hebb's rule. The network can therefore recognize other similar inputs;

From Single Output to Multiple outputs

a_1^4	a_2^4	a_3^4	a_4^4
1	0	1	0

w_1^4	w_2^4	w_3^4	w_4^4
0.71	0.02	0.71	0.02

When there are multiple outputs, weight vector becomes weight matrix.

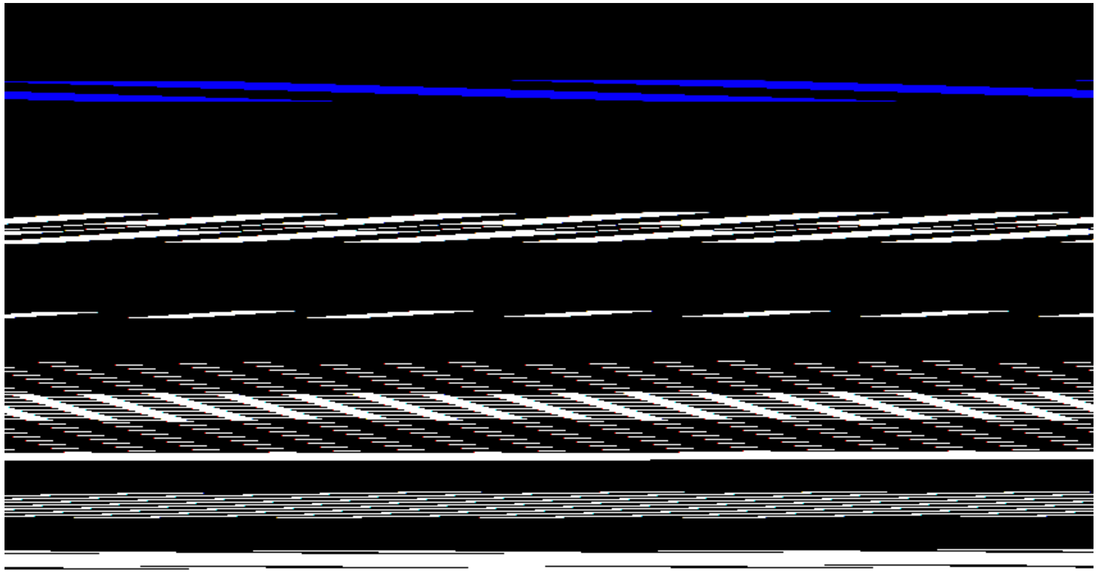
$$w = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$$

n: number of inputs
m: number of outputs

What can we expect for a multi-output network?

- For a single output network, we can strengthen the link between an input pattern and an output by normalized Hebb's rule. The network can therefore recognize other similar inputs;
- For a multi-output network, it is supposed to recognize multiple types of patterns. Each input could be recognized as either one type of pattern, or none of them. (**Clustering**)

Self-Organizing Map



[www.wikidata.org/wiki/Q1136838]