F20DL and F21DL: Part 2 Machine Learning
Lecture 7. Supervised Learning: Neural Networks

Katya Komendantskaya

Progress and plans

The second half of the course

has been devoted to detailed discussion of machine-learning algorithms employed in Data mining:

- W1: Bayesian rule, Bayesian learning, Bayesian nets
- W2: Unsupervised learning: clustering
- W3: Supervised learning: Decision trees and Linear Classifiers

Progress and plans

The second half of the course

has been devoted to detailed discussion of machine-learning algorithms employed in Data mining:

- W1: Bayesian rule, Bayesian learning, Bayesian nets
- W2: Unsupervised learning: clustering
- W3: Supervised learning: Decision trees and Linear Classifiers
- This week: Supervised Learning: Neural nets

Progress and plans

The second half of the course

has been devoted to detailed discussion of machine-learning algorithms employed in Data mining:

- W1: Bayesian rule, Bayesian learning, Bayesian nets
- W2: Unsupervised learning: clustering
- W3: Supervised learning: Decision trees and Linear Classifiers
- This week: Supervised Learning: Neural nets

- CW3, is due 28th November (available now)

Two most common computing devices:

	computer/bits	brain/neurons
processing units		
mode of computation	sequential	
mode of operating	programmed	

Two most common computing devices:

	computer/bits	**brain/neurons**
processing units	21.474.836.480	
mode of computation	sequential	
mode of operating	programmed	

Two most common computing devices:

	computer/bits	brain/neurons
processing units	21.474.836.480	100.000.000.000
mode of computation	sequential	
mode of operating	programmed	

Two most common computing devices:

	computer/bits	brain/neurons
processing units	21.474.836.480	100.000.000.000
mode of computation	sequential	distributed
mode of operating	programmed	

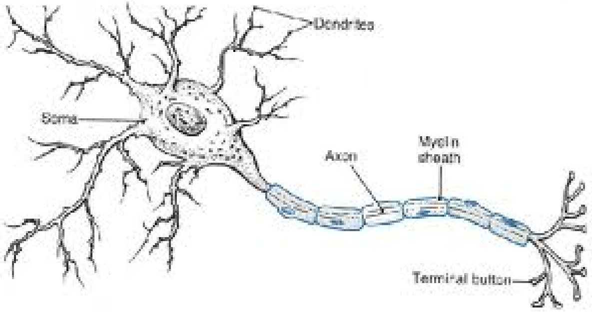
Two most common computing devices:

	computer/bits	**brain/neurons**
processing units	21.474.836.480	100.000.000.000
mode of computation	sequential	distributed
mode of operating	programmed	emergent

how does human brain work?

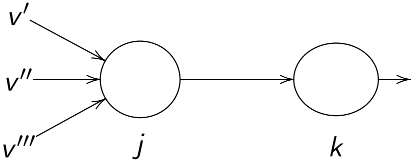
Brain neurons firing

Structure of neurons



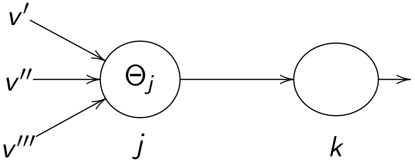
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



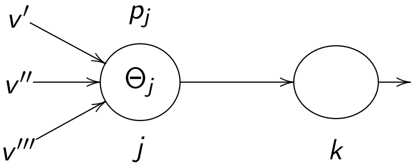
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t) v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



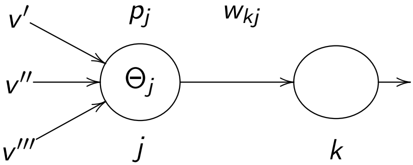
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



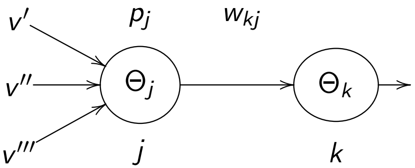
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t) v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



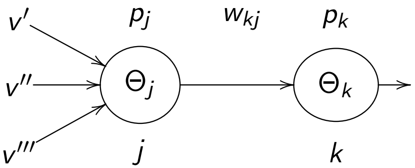
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



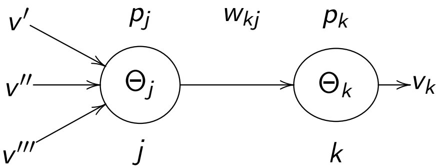
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



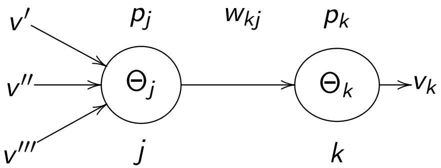
Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



Artificial Neurons

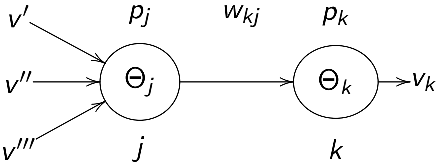
Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$



The following parameters can be trained: weights w_{kj}, biases (or Thresholds) Θ_k, Θ_j.

Artificial Neurons

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$

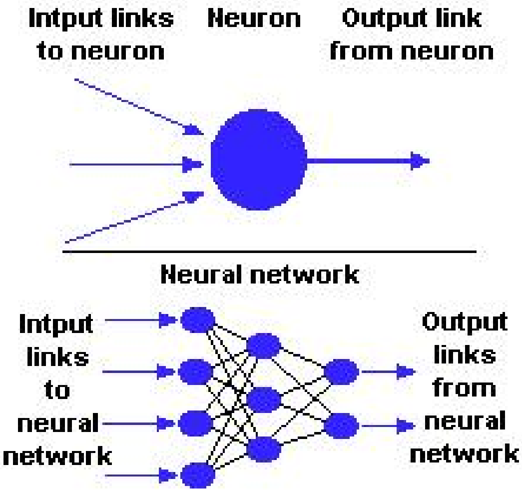


The following parameters can be trained: weights w_{kj}, biases (or Thresholds) Θ_k, Θ_j.

That's right, this reminds us the lecture on Linear classifiers!

Neural Network is...

a directed graph where each node and edge has the above parameters...



You change the NN behavior by:

- changing network's architecture/configuration of neurons;

You change the NN behavior by:

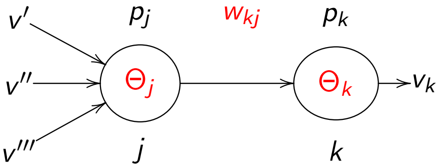
- changing network's architecture/configuration of neurons;
- changing the activation function;

You change the NN behavior by:

- changing network's architecture/configuration of neurons;
- changing the activation function;
- changing the weights;

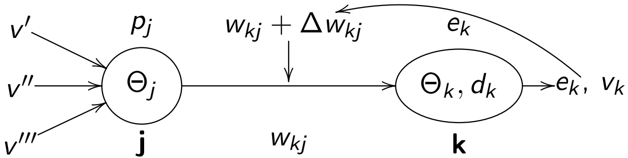
You change the NN behavior by:

- changing network's architecture/configuration of neurons;
- changing the activation function;
- changing the weights;
- changing the thresholds.



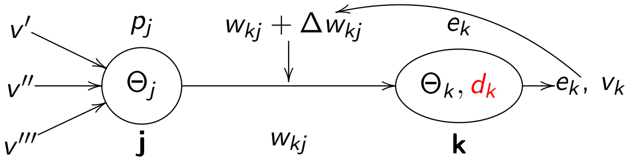
$v_k(t) = \psi(p_k(t))$

Error-Correction (Supervised) Learning



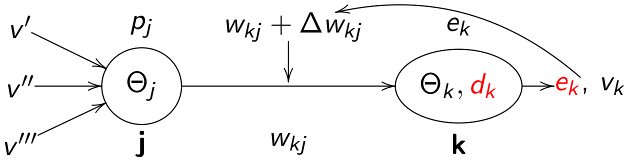
Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;



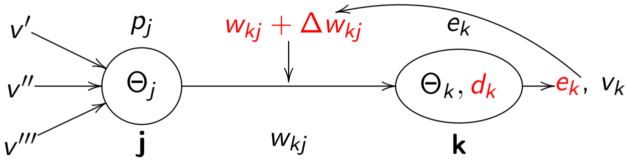
Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;
Error-signal: e.g. absolute error $e_k(t) = d_k(t) - v_k(t)$;



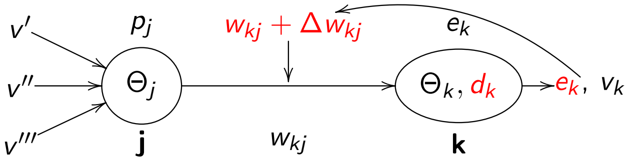
Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;
Error-signal: e.g. absolute error $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.



Error-Correction (Supervised) Learning

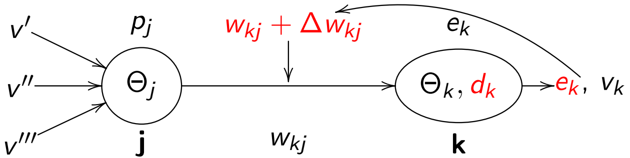
We embed a new parameter, **desired response** d_k into neurons;
Error-signal: e.g. absolute error $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.



This mode of learning is called *gradient descent*.

Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;
Error-signal: e.g. absolute error $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

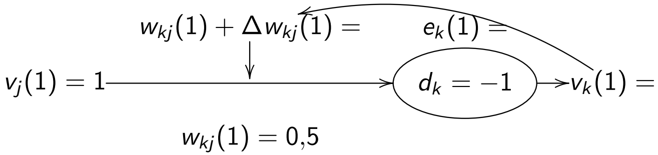


This mode of learning is called *gradient descent*.
NB: last lecture we had a formula $w_i := w_i + \eta \times \delta \times val(e, X_i)$
with $\delta = val(e, Y) - pval^{\overline{w}}(e, Y)$

Example: Error-Correction (Supervised) Learning

Neuron's potential: $p_k(t) = \sum_{j=1}^{n_k} w_{kj}(t)v_j(t) - \Theta_k$
Neuron's value: $v_k(t) = \psi(p_k(t))$
Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.
weight update: $w_{kj}(t + \Delta t) = w_{kj}(t) + \Delta w_{kj}(t)$

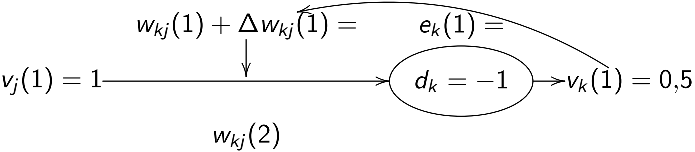
The example will work for Perceptrons:
Suppose $\Theta_k = 0$, $\eta = 1$, $\psi = id$. We start at time $t = 1$.



Example: Error-Correction (Supervised) Learning

Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

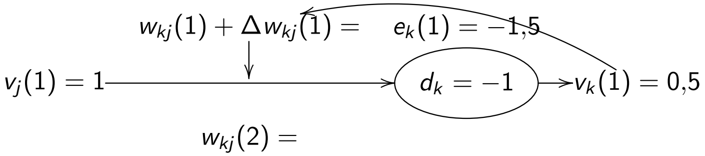
The example will work for Perceptrons:
Suppose $\Theta_k = 0$, $\eta = 1$. We start at time $t = 1$.



Example: Error-Correction (Supervised) Learning

Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

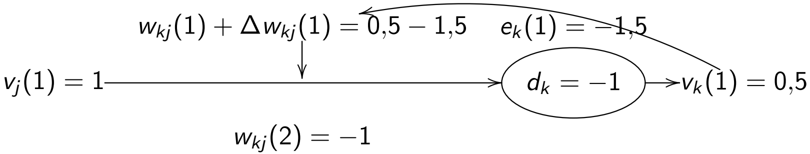
The example will work for Perceptrons:
Suppose $\Theta_k = 0$, $\eta = 1$. We start at time $t = 1$.



Example: Error-Correction (Supervised) Learning

Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

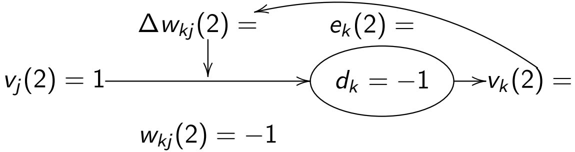
The example will work for Perceptrons:
Suppose $\Theta_k = 0$, $\eta = 1$. We start at time $t = 1$.



Example: Error-Correction (Supervised) Learning

Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\delta = \Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

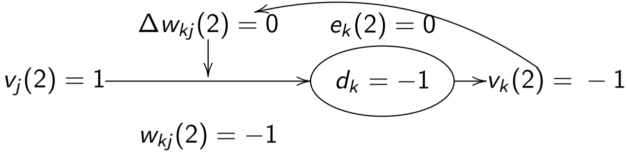
The example will work for Perceptrons:
Suppose $\Theta_k = 0$, $\eta = 1$. We start at time $t = 1$.



Example: Error-Correction (Supervised) Learning

Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\delta = \Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

The example will work for Perceptrons:
Suppose $\Theta_k = 0$, $\eta = 1$. We start at time $t = 1$.



You change the mode of learning...

- by changing the learning rate η.

You change the mode of learning...

- by changing the learning rate η.
- by changing the formula computing the error

You change the mode of learning...

- by changing the learning rate η.
- by changing the formula computing the error

Suppose the neuron predicts: $v(i)$ for input i. Measure of success is how different $v(i)$ is from the actual value $d(i)$

- absolute error $|d(i) - v(i)|$

You change the mode of learning...

- by changing the learning rate η.
- by changing the formula computing the error

Suppose the neuron predicts: $v(i)$ for input i. Measure of success is how different $v(i)$ is from the actual value $d(i)$

- absolute error $|d(i) - v(i)|$
- sum of squares error $(d(i) - v(i))^2$

You change the mode of learning...

- by changing the learning rate η.
- by changing the formula computing the error

Suppose the neuron predicts: $v(i)$ for input i. Measure of success is how different $v(i)$ is from the actual value $d(i)$

- absolute error $|d(i) - v(i)|$
- sum of squares error $(d(i) - v(i))^2$
- worst-case error $max_i|d(i) - v(i)|$

You change the mode of learning...

- by changing the learning rate η.
- by changing the formula computing the error

Suppose the neuron predicts: $v(i)$ for input i. Measure of success is how different $v(i)$ is from the actual value $d(i)$

- absolute error $|d(i) - v(i)|$
- sum of squares error $(d(i) - v(i))^2$
- worst-case error $max_i|d(i) - v(i)|$

That's right, this looks familiar – see the error estimation in the "Linear Classifiers" lecture

One Neuron is limited

Suppose we got a few:

$$v_1 \longrightarrow w_{n1} \longrightarrow n1 \longrightarrow v_{n1}$$

$$v_2 \longrightarrow w_{n2} \longrightarrow n2 \longrightarrow v_{n2}$$

$$v_3 \longrightarrow w_{n3} \longrightarrow n3 \longrightarrow v_{n3}$$

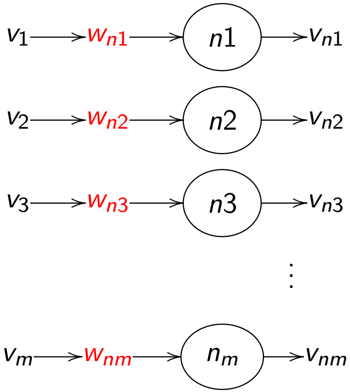
$$\vdots$$

$$v_m \longrightarrow w_{nm} \longrightarrow n_m \longrightarrow v_{nm}$$

What do we get? –

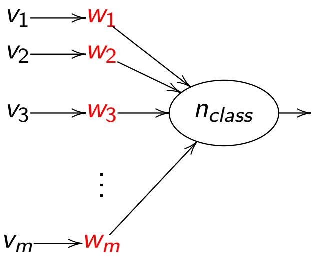
One Neuron is limited

Suppose we got a few:



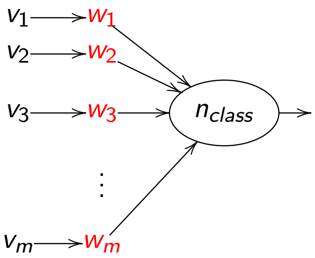
What do we get? – A function from $v_1, \ldots v_m$ to $v_{n1}, \ldots v_{nm}$ with (trained) parameters $w_{n1}, \ldots w_{nm}$. Already good to talk about relation of m attributes to m classes. What if we have just one class (as we used to have in our examples)?

One Neuron is limited



Now, just by my definition of neurons and activation functions, this network simulates the function: $f(v_1, v_2, \ldots, v_m) = \psi(\theta + v_1 w_1 + v_2 w_2 + \ldots + v_m w_m)$, where ψ is whatever activation function the neuron n_{class} has.

One Neuron is limited



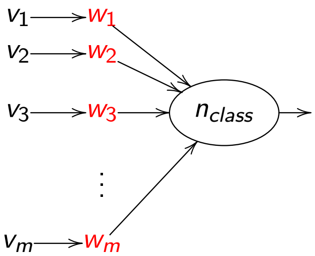
Now, just by my definition of neurons and activation functions, this network simulates the function:
$f(v_1, v_2, \ldots, v_m) = \psi(\theta + v_1 w_1 + v_2 w_2 + \ldots + v_m w_m)$, where ψ is whatever activation function the neuron n_{class} has.

Remember our old linear classifier formula?

$f(X_1, \ldots, X_n) = G(w_0 + w_1 X_1 + \ldots + w_n X_n)$, where $X_1, \ldots X_n$ were the (input) features/attributes.

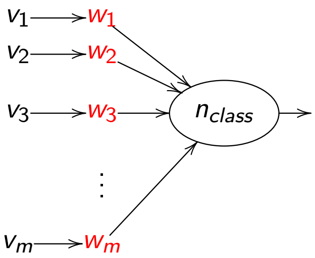
How to train this network?

- A neural network of this shape is called Perceptron:



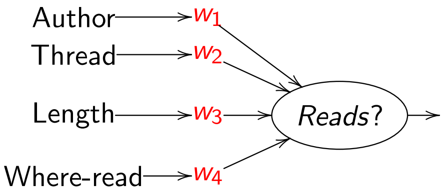
- It is exactly the linear classifier
  $n_{class}(X_1, \ldots, X_n) = G(w_0 + w_1 X_1 + \ldots + w_n X_n)$
- It may be a logistic classifier, if G is a sigmoid function

How to train this network?



Take a few examples of classification tasks with m features/attributes, and with a target value for each example. The target values will be desired response $d_{n_{class}}$. Choose an error function and error rate, and perform training as we did with one neuron.

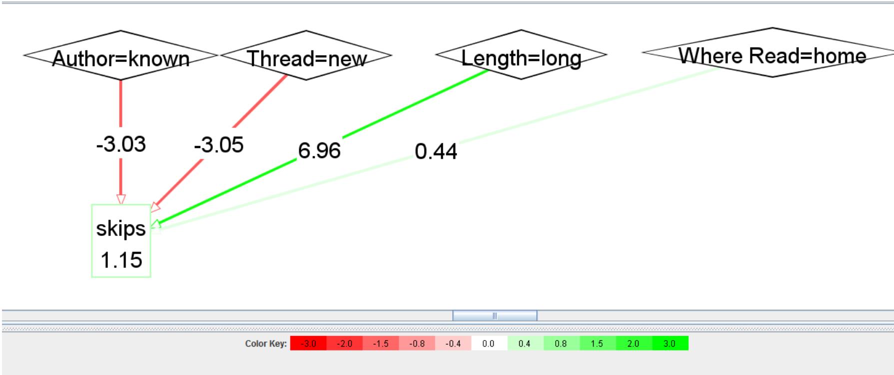
Example: email classification



Input 1 for Author = known, Thread = new, Length = Long, Where read = home, and 0 otherwise. Reads is represented by 1, and "Does not read " – by 0. Use this neural net to compute the optimal numeric values for w_1, w_2, w_3, w_4, so that the network outputs correct predictions for the given examples.

Example: mail classification

...live demo on the board...



Test 4: Facial emotion recognition set

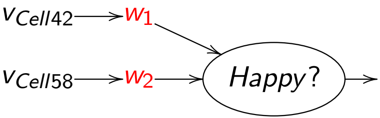
Take

Picture	Cell 33	Cell 42	Cell 48	Cell 58	Face expression
P1	White	Black	White	White	Happy
P3	White	White	White	Black	Sad
P9	Black	Black	Black	Black	Sad

Test 4, Part 1: Facial emotion recognition set

Manually train on:

Picture		v_{Cell42}		v_{Cell58}	d_{Happy}
P1		1		0	1
P3		0		1	0
P9		1		1	0

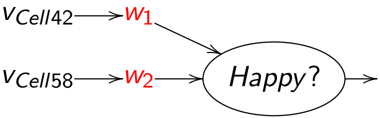


Manually test on:

Test 1: Happy face with noise in the picture: `1, 0, ???`

Test 2: Happy face with a beard: `1, 1, ???`

Training settings:



- Learning rate $\eta = 0{,}3$
- $\theta = 0$, suppose we do not train θ
- Activation function – just identity (no function applied), i.e.: $v_{Happy}(t) = p_{Happy}(t) = w_1(t) * v_{Cell42}(t) + w_2(t) * v_{Cell58}(t)$
- Random weights initialised at time$= 1$: $w_1(1) = 1$, $w_2(1) = 0$
- **Error-signal**: $e_{Happy}(t) = d_{Happy}(t) - v_{Happy}(t)$
- Error-correction learning rule: $\Delta w_1(t) = \eta * e_{Happy}(t) * v_{Cell42}(t)$, $\Delta w_2(t) = \eta * e_{Happy}(t) * v_{Cell58}(t)$; $w_1(t+1) = w_1(t) + \Delta w_1(t)$; $w_2(t+1) = w_2(t) + \Delta w_2(t)$
- Time counter: $(t = 1)$ send $P1$; $(t = 2)$ send $P3$, $(t = 3)$ send $P9$, end with computing $w_1(4)$, $w_2(4)$ at step $t = 4$.
- **Record all intermediate steps and parameters, be ready to answer questions**

Back to our Philosophical questions

Supervised Learning is about finding a good hypothesis in the hypothesis space, with the purpose of making (class) predictions.

Hypotheses were given by:

- decision trees,

Back to our Philosophical questions

Supervised Learning is about finding a good hypothesis in the hypothesis space, with the purpose of making (class) predictions.

Hypotheses were given by:

- decision trees,
- linear functions,

Back to our Philosophical questions

Supervised Learning is about finding a good hypothesis in the hypothesis space, with the purpose of making (class) predictions.

Hypotheses were given by:

- decision trees,
- linear functions,
- ???

Back to our Philosophical questions

Supervised Learning is about finding a good hypothesis in the hypothesis space, with the purpose of making (class) predictions.

Hypotheses were given by:

- decision trees,
- linear functions,
- ??? Neural Network Parameters

Next lecture: more complex Neural Net architectures