Natural Language Understanding

Lecture 4: Perceptrons

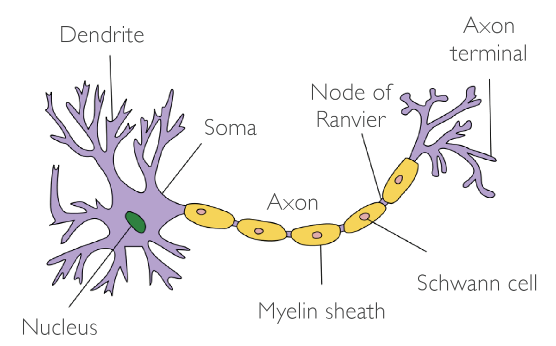
Adam Lopez

Some slides by Mirella Lapata and Frank Keller

21 January 2019

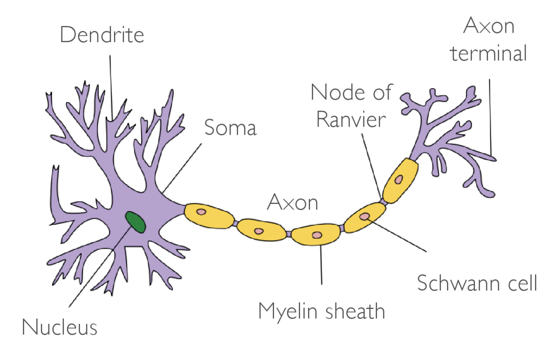
School of Informatics
University of Edinburgh
alopez@inf.ed.ac.uk

Biological neural networks



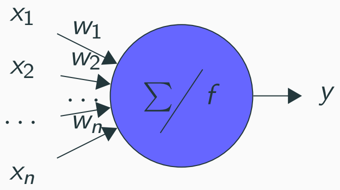
- Neuron receives inputs and combines these in the cell body.
- If the input reaches a threshold, then the neuron may fire (produce an output).
- Some inputs are excitatory, while others are inhibitory.

Biological neural networks

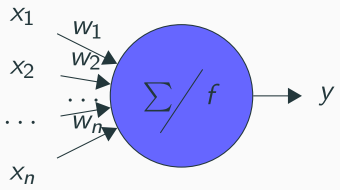


Artificial neural networks are even more cartoony than this picture, so will focus strictly on the mathematics, rather than woolly brain metaphors. (Be very skeptical of these metaphors whether you see them in a press releases or a hyped-up machine learning paper.)

The perceptron: a very simple artificial neuron



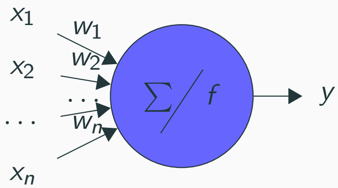
The perceptron: a very simple artificial neuron



Input function:

$$u(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

The perceptron: a very simple artificial neuron



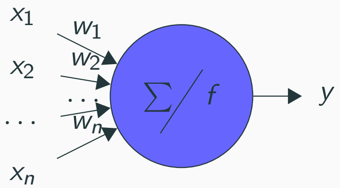
Input function:

$$u(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

Activation function: threshold

$$y = f(u(\mathbf{x})) = \begin{cases} 1, & \text{if } u(\mathbf{x}) > \theta \\ 0, & \text{otherwise} \end{cases}$$

The perceptron: a very simple artificial neuron



Input function:

$$u(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

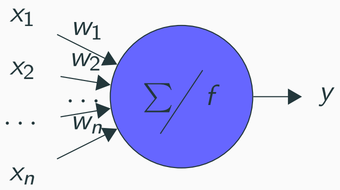
Activation function: threshold

$$y = f(u(\mathbf{x})) = \begin{cases} 1, & \text{if } u(\mathbf{x}) > \theta \\ 0, & \text{otherwise} \end{cases}$$

Activation state:

0 or 1 (-1 or 1)

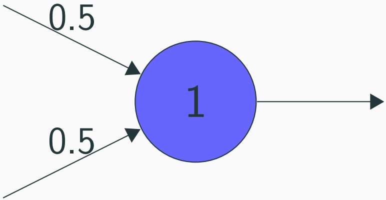
The perceptron: a very simple artificial neuron



- Inputs are in the range [0, 1], where 0 is "off" and 1 is "on".
- Weights can be any real number (positive or negative).

Perceptrons can represent logic functions

Perceptron for AND

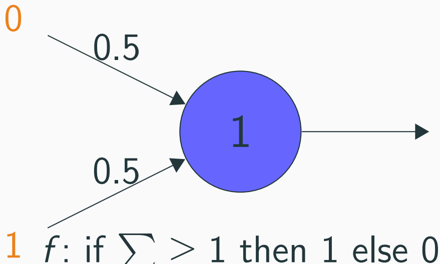


f: if $\sum \geq 1$ then 1 else 0

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND

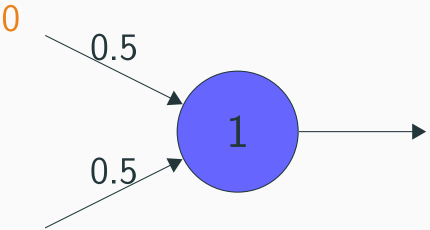


f: if $\sum \geq 1$ then 1 else 0

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND



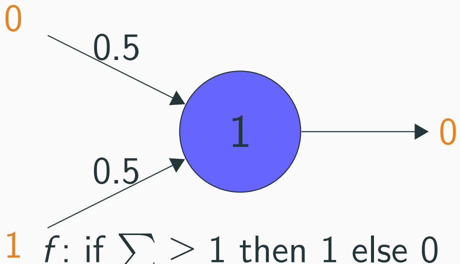
f: if $\sum \geq 1$ then 1 else 0

$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5 < 1$

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND



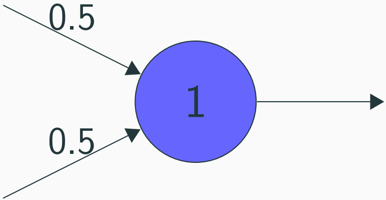
f: if $\sum \geq 1$ then 1 else 0

$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5 < 1$

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND

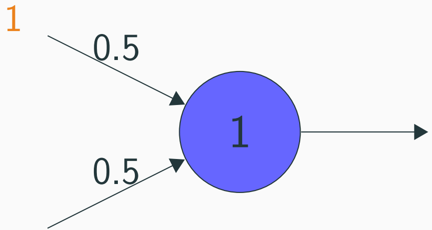


f: if $\sum \geq 1$ then 1 else 0

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND

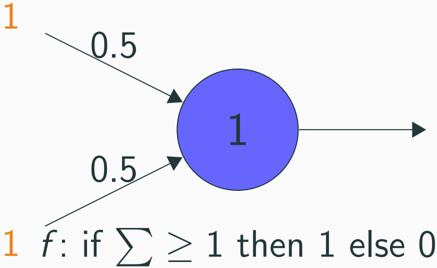


f: if $\sum \geq 1$ then 1 else 0

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND



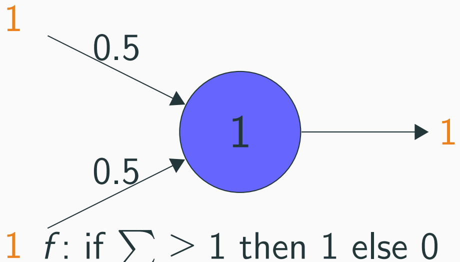
f: if $\sum \geq 1$ then 1 else 0

$$1 \cdot 0.5 + 1 \cdot 0.5 = 1 = 1$$

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for AND



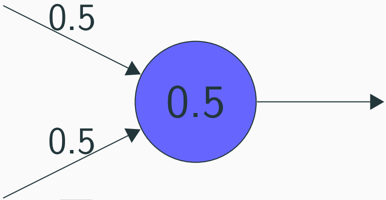
f: if $\sum \geq 1$ then 1 else 0

$$1 \cdot 0.5 + 1 \cdot 0.5 = 1 = 1$$

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrons can represent logic functions

Perceptron for OR

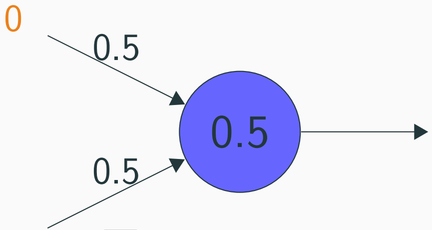


f: if $\sum \geq 0.5$ then 1 else 0

x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

Perceptrons can represent logic functions

Perceptron for OR

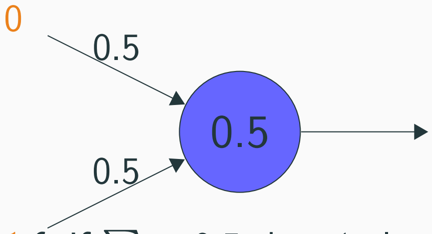


1 f: if $\sum \geq 0.5$ then 1 else 0

x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

Perceptrons can represent logic functions

Perceptron for OR



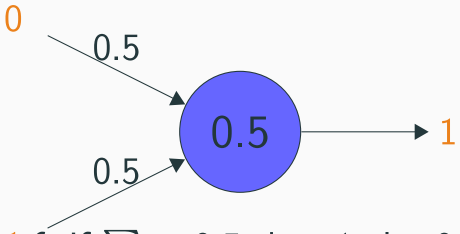
f: if $\sum \geq 0.5$ then 1 else 0

$$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5 = 0.5$$

x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

Perceptrons can represent logic functions

Perceptron for OR



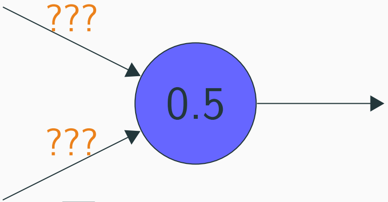
f: if $\sum \geq 0.5$ then 1 else 0

$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5 = 0.5$

x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

How would you represent NOT(OR)?

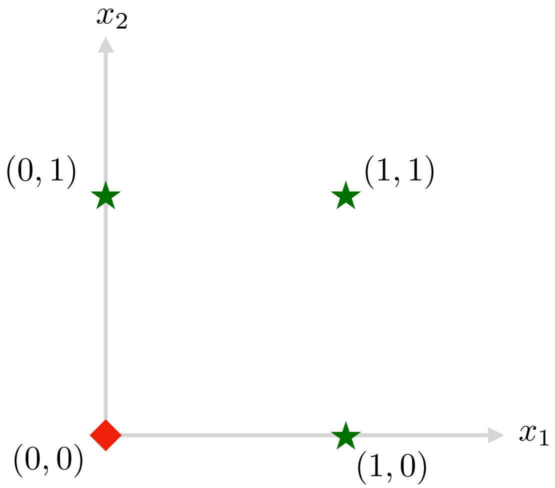
Perceptron for NOT(OR)



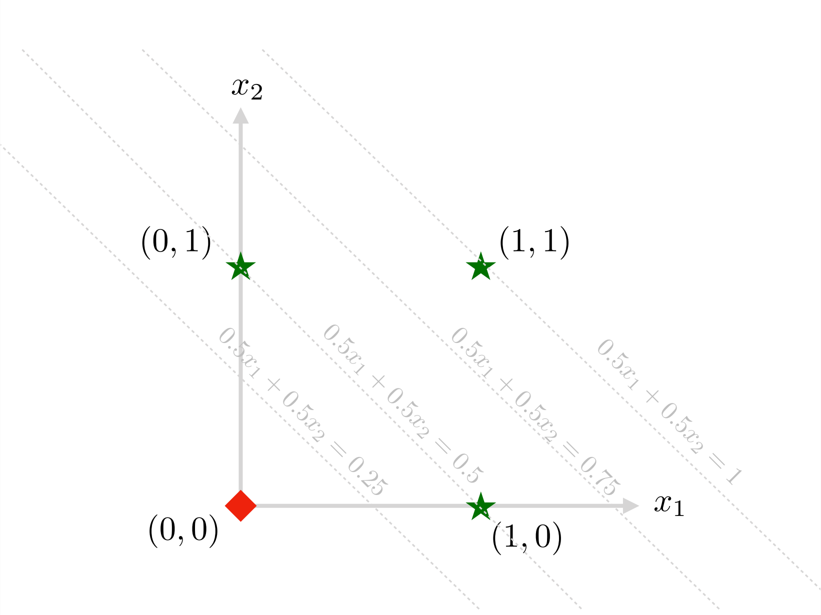
f: if $\sum \geq$??? then 1 else 0

x_1	x_2	x_1 OR x_2
0	0	1
0	1	0
1	0	0
1	1	0

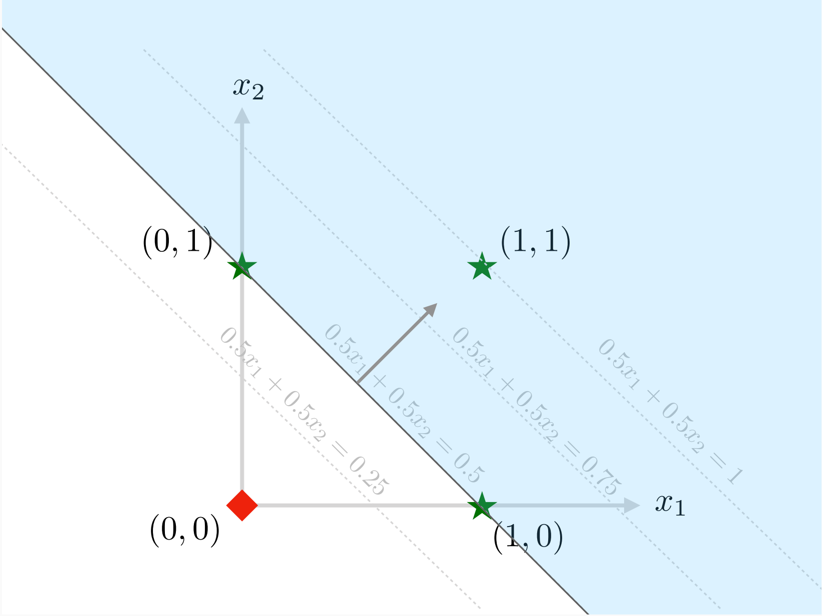
Perceptrons are linear classifiers (OR example)



Perceptrons are linear classifiers (OR example)

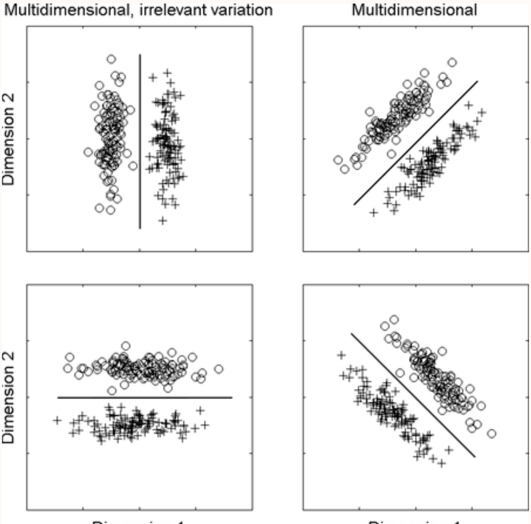


Perceptrons are linear classifiers (OR example)

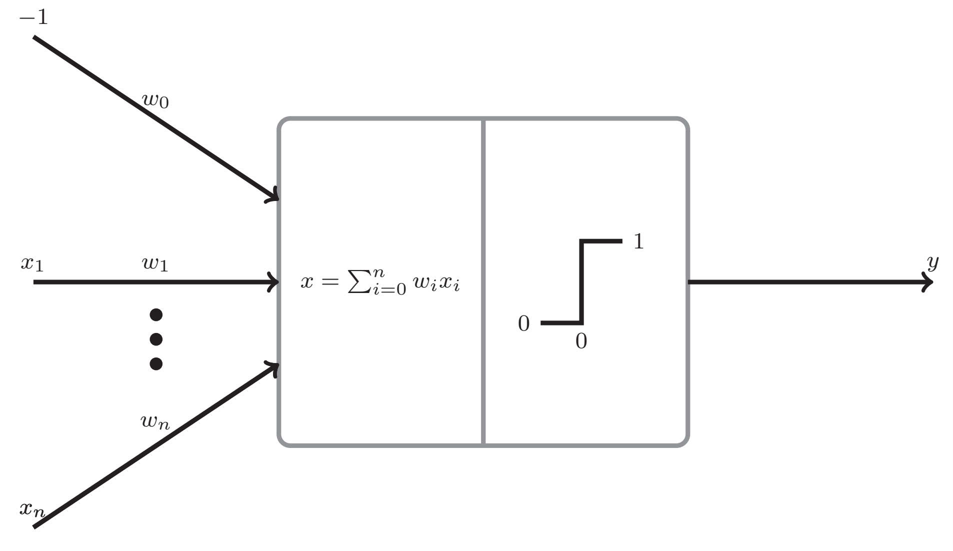


Perceptrons are linear classifiers

More generally: can separate arbitrary real-valued points, under certain conditions.



Schematic representation of a perceptron



Perceptron can learn (some) logic functions from examples

Give some examples to the Perceptron:

N	**input** x	**target** t
1	(0,1,0,0)	1
2	(1,0,0,0)	0
3	(0,1,1,1)	0
4	(1,0,1,0)	0
5	(1,1,1,1)	1
6	(0,1,0,0)	1
...	...	...

- Input: a vector of 1's and 0's—-a feature vector.
- Output: a 1 or 0, given as the target.

Perceptron can learn (some) logic functions from examples

Give some examples to the Perceptron:

N	**input** x	**target** t	**output** o
1	(0,1,0,0)	1	0
2	(1,0,0,0)	0	0
3	(0,1,1,1)	0	1
4	(1,0,1,0)	0	1
5	(1,1,1,1)	1	0
6	(0,1,0,0)	1	1
...	...	...	...

- Input: a vector of 1's and 0's—-a feature vector.
- Output: a 1 or 0, given as the target.

Perceptron can learn (some) logic functions from examples

Give some examples to the Perceptron:

N	**input** x	**target** t	**output** o	**update?**
1	(0,1,0,0)	1	0	y
2	(1,0,0,0)	0	0	
3	(0,1,1,1)	0	1	y
4	(1,0,1,0)	0	1	y
5	(1,1,1,1)	1	0	y
6	(0,1,0,0)	1	1	
...	...	...		

- Input: a vector of 1's and 0's—-a feature vector.
- Output: a 1 or 0, given as the target.

Perceptron can learn (some) logic functions from examples

Give some examples to the Perceptron:

N	**input** x	**target** t	**output** o	**update?**
1	(0,1,0,0)	1	0	y
2	(1,0,0,0)	0	0	
3	(0,1,1,1)	0	1	y
4	(1,0,1,0)	0	1	y
5	(1,1,1,1)	1	0	y
6	(0,1,0,0)	1	1	
...	...	...		

- Input: a vector of 1's and 0's—-a feature vector.
- Output: a 1 or 0, given as the target.
- How do we efficiently find the weights and threshold?

Learning

Q_1: Choosing weights and threshold θ for the perceptron is not easy! What's an effective way to learn the weights and threshold from examples?

A_1: We use a learning algorithm that adjusts the weights and threshold based on examples.

Simplify by converting θ into a weight

$$\sum_{i=1}^{n} w_i x_i > \theta$$

Simplify by converting θ into a weight

$$\sum_{i=1}^{n} w_i x_i > \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta > 0$$

Simplify by converting θ into a weight

$$\sum_{i=1}^{n} w_i x_i > \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta > 0$$

$$w_1 x_1 + w_2 x_2 + \ldots w_n x_n - \theta > 0$$

Simplify by converting θ into a weight

$$\sum_{i=1}^{n} w_i x_i > \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta > 0$$

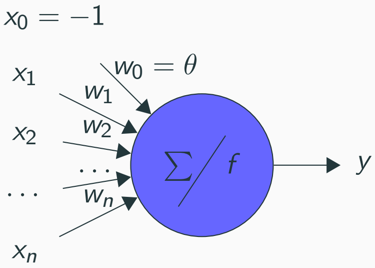
$$w_1 x_1 + w_2 x_2 + \ldots w_n x_n - \theta > 0$$

$$w_1 x_1 + w_2 x_2 + \ldots w_n x_n + \theta(-1) > 0$$

Simplify by converting θ into a weight

$$\sum_{i=1}^{n} w_i x_i > \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta > 0$$

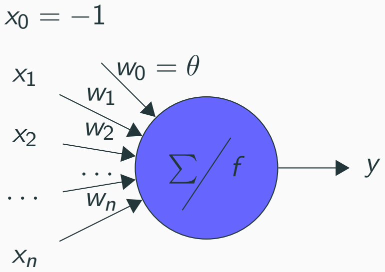


$$w_1 x_1 + w_2 x_2 + \dots w_n x_n - \theta > 0$$

$$w_1 x_1 + w_2 x_2 + \dots w_n x_n + \theta(-1) > 0$$

The quantity $-\theta$ is called the bias, often denoted with b.

Simplify by converting θ into a weight



Let $x_0 = -1$ be the weight of θ. Now our activation function is:

$$y = f(u(\mathbf{x})) = \begin{cases} 1, & \text{if } u(\mathbf{x}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

Learn by adjusting weights whenever output $\neq$ target

Intuition: classification depends on the sign (+ or -) of the output. If output has a different sign than the target, adjust weights to move output in the direction of 0.

Learn by adjusting weights whenever output $\neq$ target

Intuition: classification depends on the sign (+ or -) of the output. If output has a different sign than the target, adjust weights to move output in the direction of 0.

output$=0$ and target$=0$	Don't adjust weights

Learn by adjusting weights whenever output $\neq$ target

Intuition: classification depends on the sign (+ or -) of the output. If output has a different sign than the target, adjust weights to move output in the direction of 0.

output$=0$ and target$=0$	Don't adjust weights
output$=0$ and target$=1$	$u(\mathbf{x})$ was too low. Make it bigger!

Learn by adjusting weights whenever output $\neq$ target

Intuition: classification depends on the sign (+ or -) of the output. If output has a different sign than the target, adjust weights to move output in the direction of 0.

output$=0$ and target$=0$	Don't adjust weights
output$=0$ and target$=1$	$u(\mathbf{x})$ was too low. Make it bigger!
output$=1$ and target$=0$	$u(\mathbf{x})$ was too high. Make it smaller!

Learn by adjusting weights whenever output $\neq$ target

Intuition: classification depends on the sign (+ or -) of the output. If output has a different sign than the target, adjust weights to move output in the direction of 0.

output$=0$ and target$=0$	Don't adjust weights
output$=0$ and target$=1$	$u(\mathbf{x})$ was too low. Make it bigger!
output$=1$ and target$=0$	$u(\mathbf{x})$ was too high. Make it smaller!
output$=1$ and target$=1$	Don't adjust weights

Learn by adjusting weights whenever output $\neq$ target

Intuition: classification depends on the sign (+ or -) of the output. If output has a different sign than the target, adjust weights to move output in the direction of 0.

output$=0$ and target$=0$	Don't adjust weights
output$=0$ and target$=1$	$u(\mathbf{x})$ was too low. Make it bigger!
output$=1$ and target$=0$	$u(\mathbf{x})$ was too high. Make it smaller!
output$=1$ and target$=1$	Don't adjust weights

Notice: the sign of $t - o$ is the direction we want to move in.

Learn by adjusting weights whenever output $\neq$ target

Perceptron Learning Rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

- η, $0 < \eta \leq 1$ is a constant called the learning rate.
- t is the target output of the current example.
- o is the output of the Perceptron with the current weights.

Learning Rule

Perceptron Learning Rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i \ = \eta(t - o)x_i$$

$o = 1$ and $t = 1$
$o = 0$ and $t = 1$

- Learning rate η is positive; controls how big changes Δw_i are.
- If $x_i > 0$, $\Delta w_i > 0$. Then w_i increases in an so that $w_i x_i$ becomes larger, increasing $u(\mathbf{x})$.
- If $x_i < 0$, $\Delta w_i < 0$. Then w_i reduces so that the absolute value of $w_i x_i$ becomes smaller, increasing $u(\mathbf{x})$.

Learning Rule

Perceptron Learning Rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

$$o = 1 \text{ and } t = 1 \quad \Delta w_i = \eta(t - o)x_i = \eta(1 - 1)x_i = 0$$
$$o = 0 \text{ and } t = 1 \quad \Delta w_i = \eta(t - o)x_i = \eta(1 - 0)x_i = \eta x_i$$

- Learning rate η is positive; controls how big changes Δw_i are.
- If $x_i > 0$, $\Delta w_i > 0$. Then w_i increases in an so that $w_i x_i$ becomes larger, increasing $u(\mathbf{x})$.
- If $x_i < 0$, $\Delta w_i < 0$. Then w_i reduces so that the absolute value of $w_i x_i$ becomes smaller, increasing $u(\mathbf{x})$.

Learning Algorithm

```
Initialize all weights randomly.
repeat
    for each training example do
        Apply the learning rule.
    end for
until the error is acceptable or a certain number of iterations is reached
```

Learning Algorithm

```
Initialize all weights randomly.
repeat
    for each training example do
        Apply the learning rule.
    end for
until the error is acceptable or a certain number
of iterations is reached
```

This algorithm is guaranteed to find a solution with zero error in a limited number of iterations **if** the examples are linearly separable.

Learning Algorithm

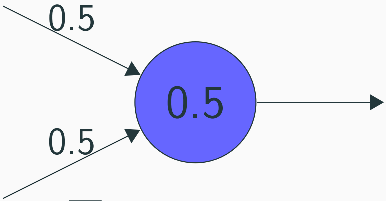
Initialize all weights randomly.
repeat
    for each training example **do**
        Apply the learning rule.
    end for
until the error is acceptable or a certain number of iterations is reached

This algorithm is guaranteed to find a solution with zero error in a limited number of iterations **if** the examples are linearly separable.

http://www.youtube.com/watch?v=vGwemZhPlsA&feature=youtu.be

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



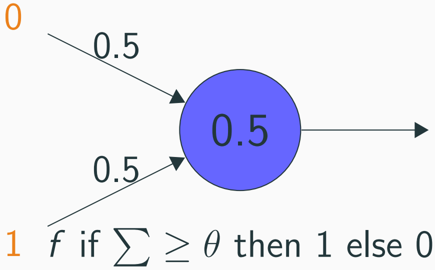
f if $\sum \geq \theta$ then 1 else 0

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR

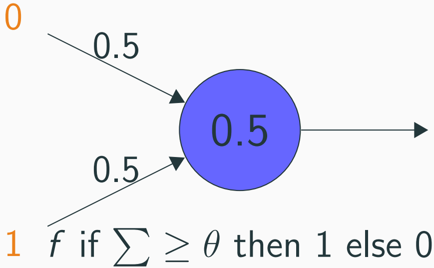


x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



f if $\sum \geq \theta$ then 1 else 0

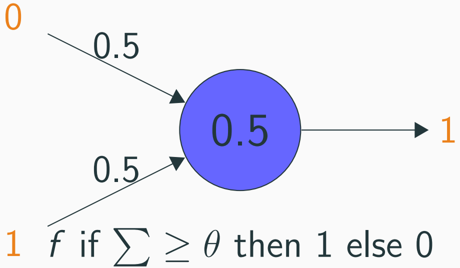
$$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5$$

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



f if $\sum \geq \theta$ then 1 else 0

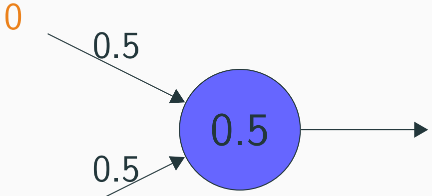
$$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5$$

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



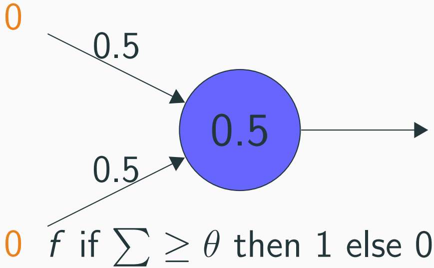
f if $\sum \geq \theta$ then 1 else 0

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



f if $\sum \geq \theta$ then 1 else 0

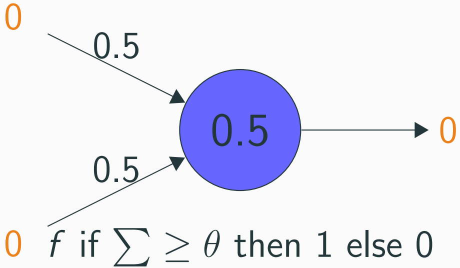
$$0 \cdot 0.5 + 0 \cdot 0.5 = 0$$

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



f if $\sum \geq \theta$ then 1 else 0

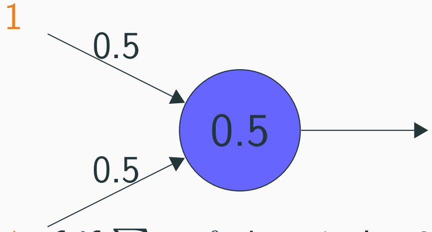
$$0 \cdot 0.5 + 0 \cdot 0.5 = 0$$

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



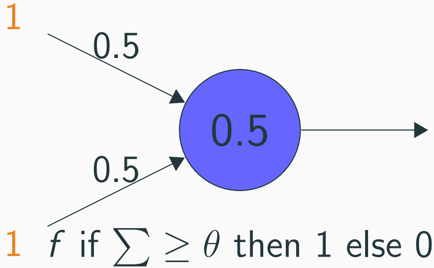
f if $\sum \geq \theta$ then 1 else 0

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR



f if $\sum \geq \theta$ then 1 else 0

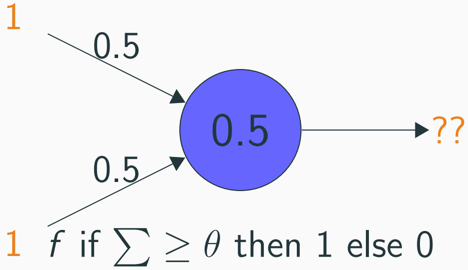
$$1 \cdot 0.5 + 1 \cdot 0.5 = 1$$

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Perceptrons can represent some logic functions... but not all!

Perceptron for XOR

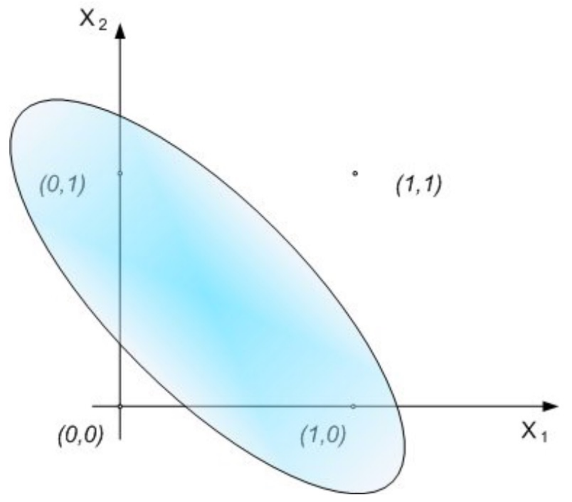


$$1 \cdot 0.5 + 1 \cdot 0.5 = 1$$

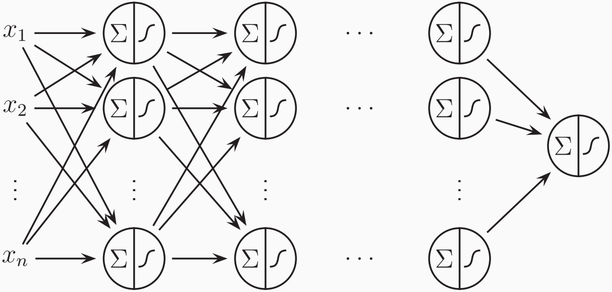
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Problem: XOR is not linearly separable



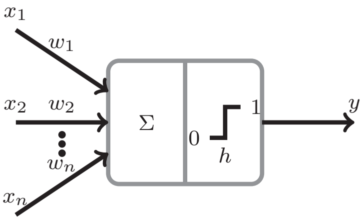
Multilayer Perceptrons (MLPs) are more expressive



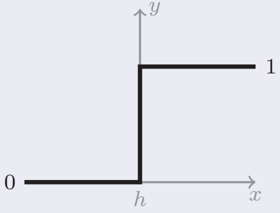
- MLPs are feed-forward neural networks, organized in layers.
- One input layer, one or more hidden layers, one output layer.
- Each node in a layer connected to all other nodes in next layer.
- Each connection has a weight (can be zero).

Q: How would you represent XOR?

We can use activation functions other than thresholds

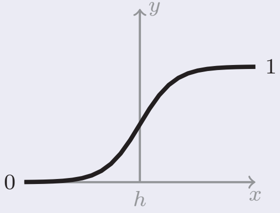


Step function



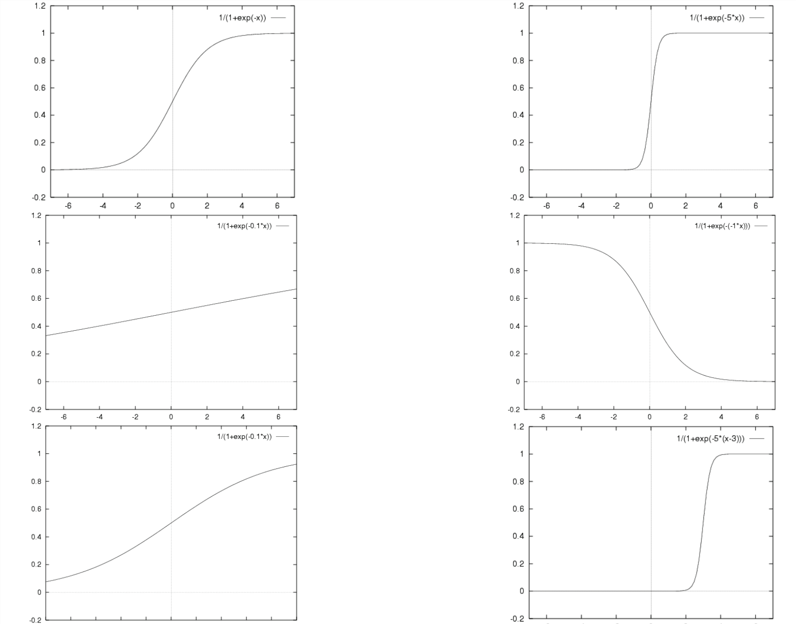
Outputs 0 or 1.

Sigmoid function

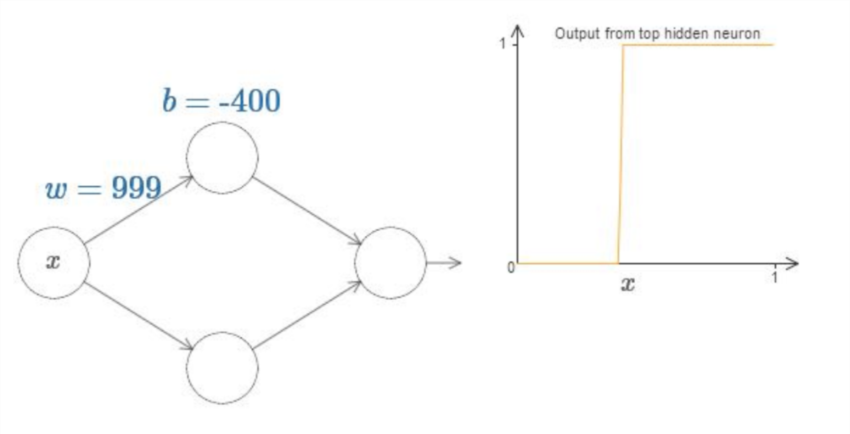


Outputs a real value between 0 and 1.

Sigmoid can be made sharper or smoother



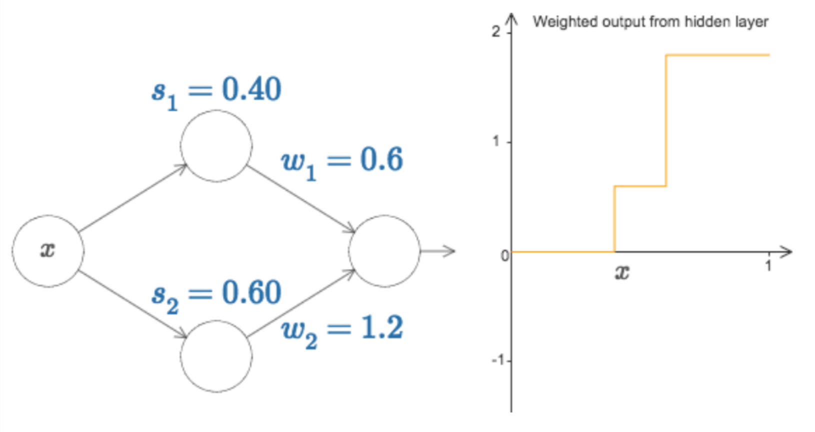
MLPs are universal function approximators



Each hidden unit can approximate a step function of the input.

Source: http://neuralnetworksanddeeplearning.com

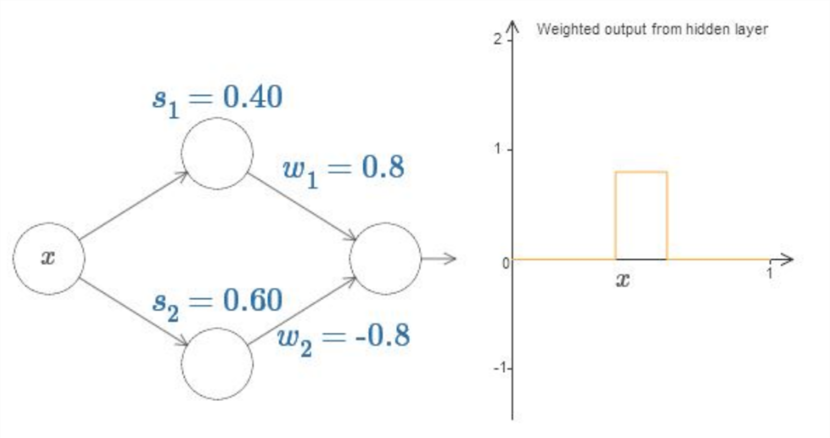
MLPs are universal function approximators



With multiple hidden units, you can get multiple steps.

Source: http://neuralnetworksanddeeplearning.com

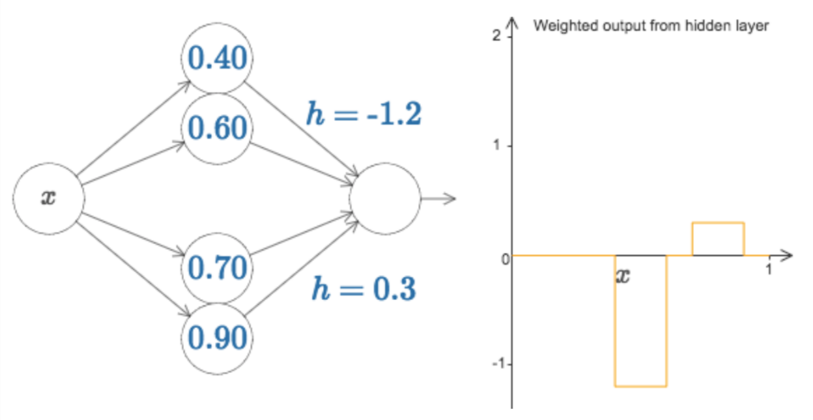
MLPs are universal function approximators



The steps can be positive or negative.

Source: http://neuralnetworksanddeeplearning.com

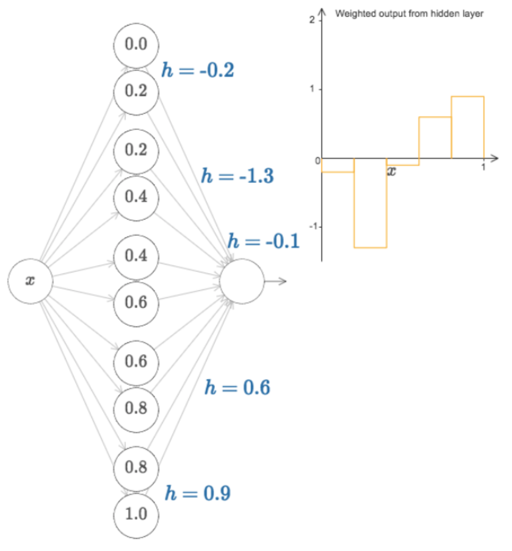
MLPs are universal function approximators



With more hidden units, you can represent more steps.

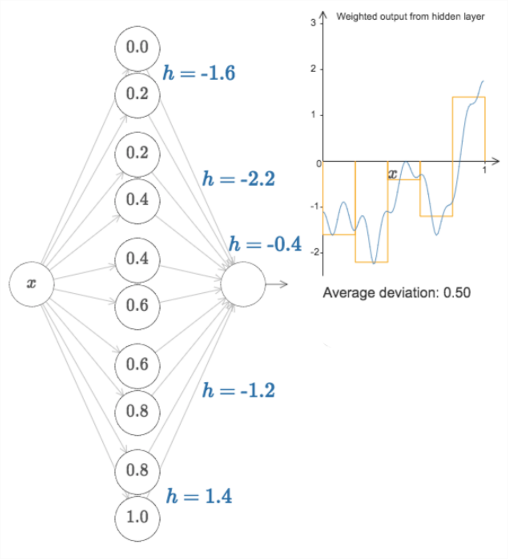
Source: http://neuralnetworksanddeeplearning.com

MLPs are universal function approximators



Many hidden units produce more complex functions.

MLPs are universal function approximators



Add hidden units to approximate very complex functions.

Summary of key points (i.e. examinable content)

- We learnt what a perceptron is.
- We have seen that perceptrons can learn linearly separable functions.
- We know a learning rule for the perceptron.
- We have seen that a multilayer perceptron (MLP) is more powerful than a perceptron, and in fact is a universal function approximator.

Next lecture: Learning with multilayer perceptrons and using them to represent *n*-gram probabilities.