Natural Language Understanding, Generation, and Machine Translation

Lecture 4: Perceptrons

Frank Keller

Week of 24 January 2022 (week 2)

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

Overview

Reading: Jurafsky and Martin (2021), Chapter 7.

Agenda for Today

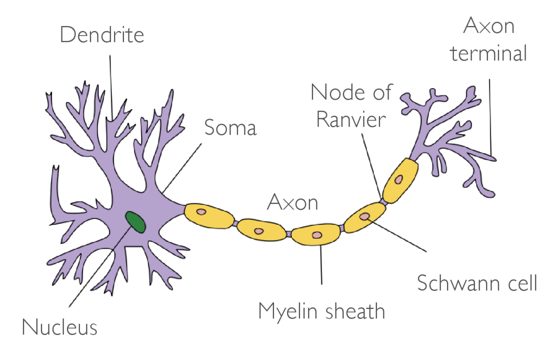
Last lecture: we reviewed some simple probabilistic models.

This lecture: we will review some basic nonlinear models: neural networks.

Next lecture: we'll combine these ideas by parameterizing a simple probabilistic model with a neural network!

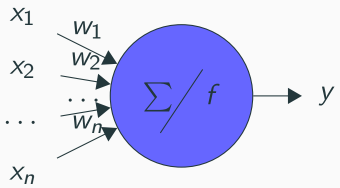
Simple Perceptrons

A (very) loose inspiration: biological neural networks



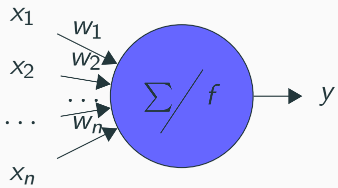
- Neuron receives inputs and combines these in the cell body.
- If the combined input reaches a threshold, then the neuron may fire (produce an output).
- Some inputs are excitatory, while others are inhibitory.

The perceptron: a very simple artificial neuron

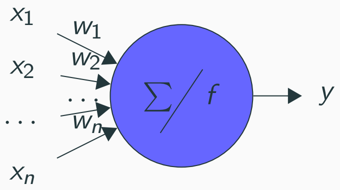


- Input is a vector $\mathbf{x}$, whose ith element is x_i. To model a logical function, let each x_i be in the range [0, 1], where 0 is "off" and 1 is "on"—though this isn't necessary in general.
- Weight vector $\mathbf{w}$ is of same length as $\mathbf{x}$. Each w_i can be any real number (positive or negative).

The perceptron: a very simple artificial neuron



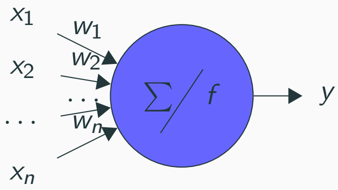
The perceptron: a very simple artificial neuron



Input function:

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

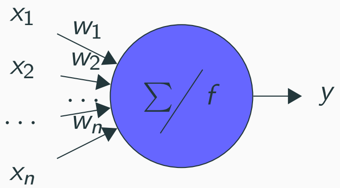
The perceptron: a very simple artificial neuron



Input function:

$u(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$

The perceptron: a very simple artificial neuron

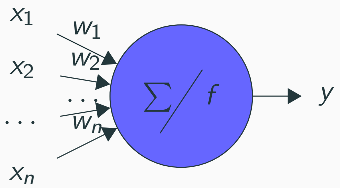


Activation function: threshold

Input function:
$u(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$

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

The perceptron: a very simple artificial neuron



Input function:
$u(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$

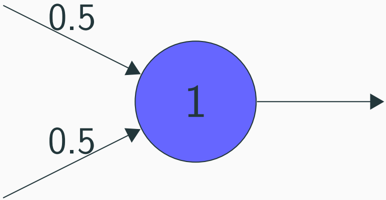
Activation function: threshold

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

Activation state:
0 or 1

Perceptrons can represent logic functions

Perceptron for AND

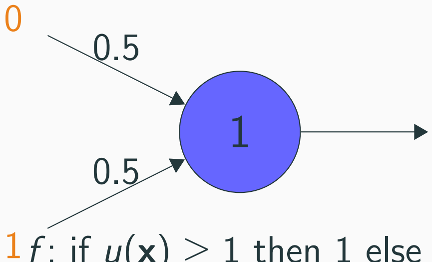


f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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

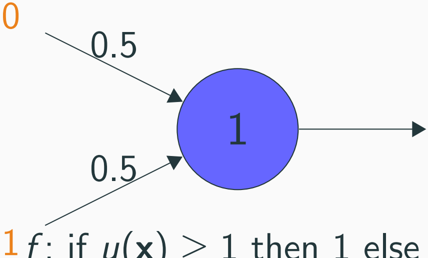


f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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



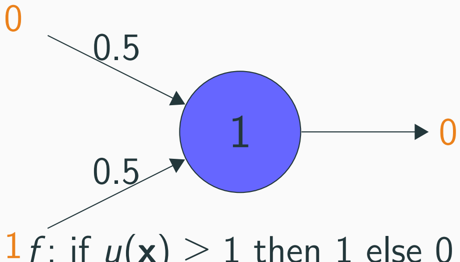
f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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

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



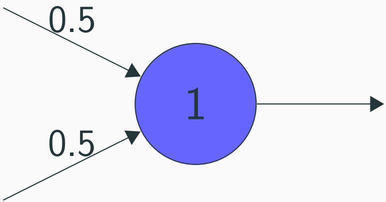
f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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

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

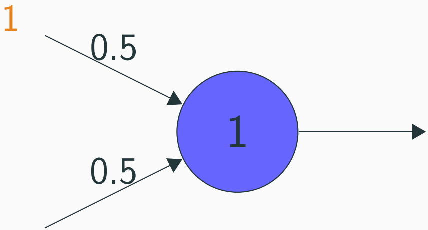


f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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

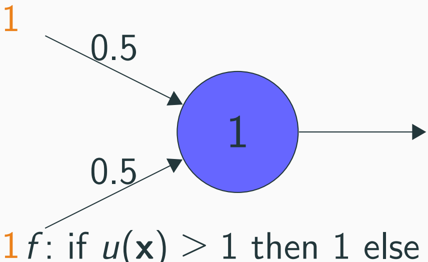


f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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



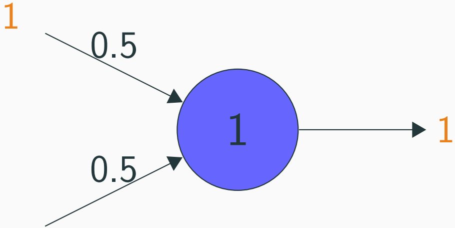
f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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

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



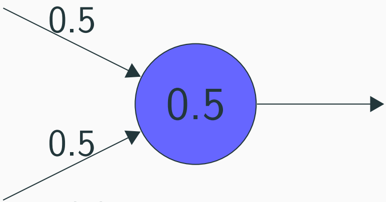
1 f: if $u(\mathbf{x}) \geq 1$ then 1 else 0

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

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

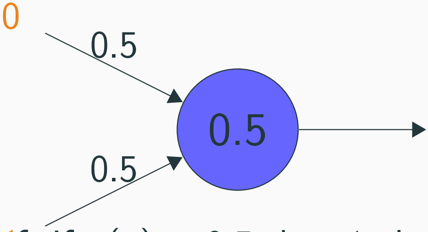


f: if $u(\mathbf{x}) \geq 0.5$ then 1 else 0

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

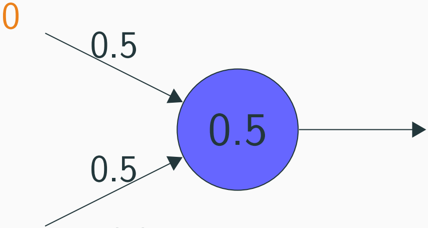


f : if $u(\mathbf{x}) \geq 0.5$ then 1 else 0

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



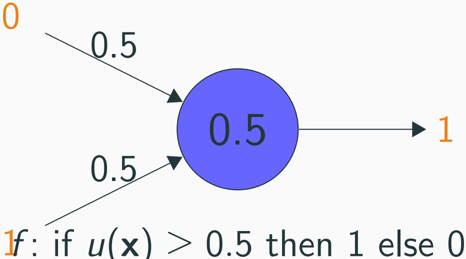
f: if $u(\mathbf{x}) \geq 0.5$ then 1 else 0

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

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



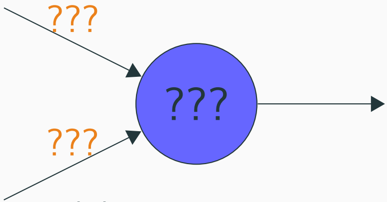
f: if $u(\mathbf{x}) \geq 0.5$ then 1 else 0

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

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)?

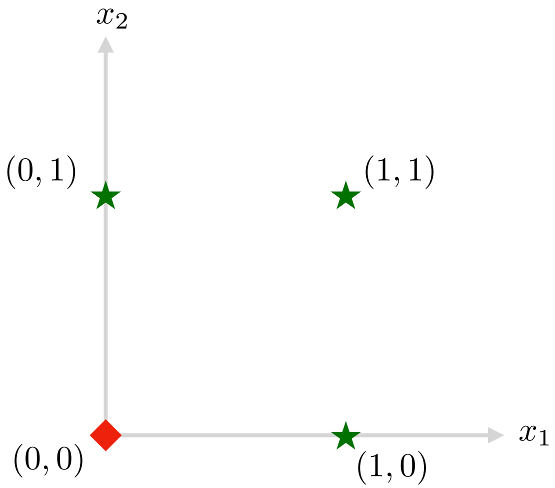
Perceptron for NOT(OR)



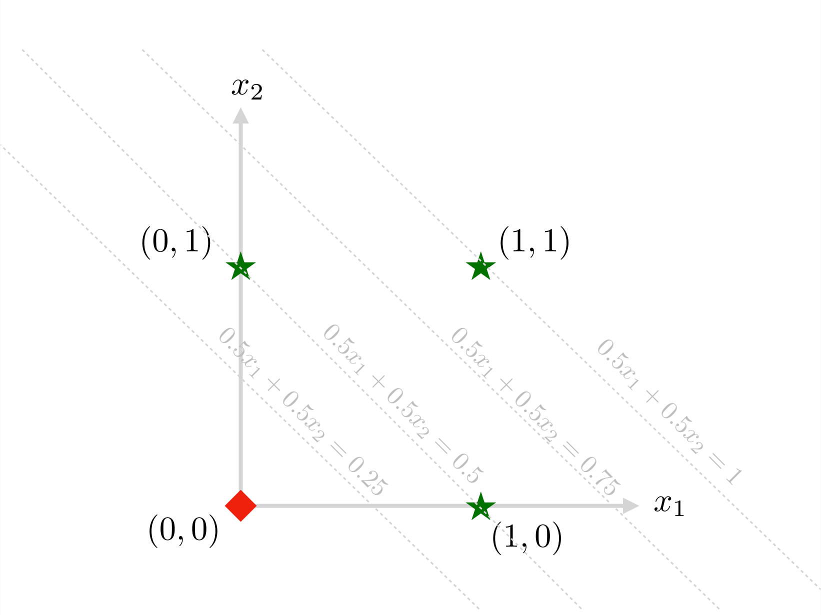
f: if $u(\mathbf{x}) \geq$??? then 1 else 0

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

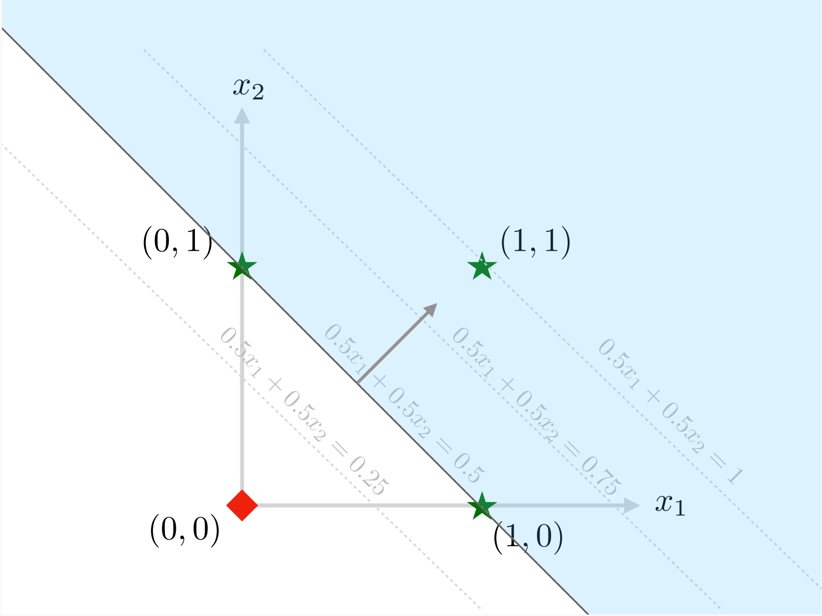
Perceptrons are linear classifiers (OR example)



Perceptrons are linear classifiers (OR example)



Perceptrons are linear classifiers (OR example)



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	yes
2	(1,0,0,0)	0	0	
3	(0,1,1,1)	0	1	yes
4	(1,0,1,0)	0	1	yes
5	(1,1,1,1)	1	0	yes
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	yes
2	(1,0,0,0)	0	0	
3	(0,1,1,1)	0	1	yes
4	(1,0,1,0)	0	1	yes
5	(1,1,1,1)	1	0	yes
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

Original: $\mathbf{w} \cdot \mathbf{x} > \theta$

Simplify by converting θ into a weight

Original:	$\mathbf{w} \cdot \mathbf{x} > \theta$
Rewritten:	$\mathbf{w} \cdot \mathbf{x} - \theta > 0$

Simplify by converting θ into a weight

Original: $\mathbf{w} \cdot \mathbf{x} > \theta$

Rewritten: $\mathbf{w} \cdot \mathbf{x} - \theta > 0$

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

Simplify by converting θ into a weight

Original: $\mathbf{w} \cdot \mathbf{x} > \theta$

Rewritten: $\mathbf{w} \cdot \mathbf{x} - \theta > 0$

Rename $-\theta$ to b: $\mathbf{w} \cdot \mathbf{x} + b > 0$

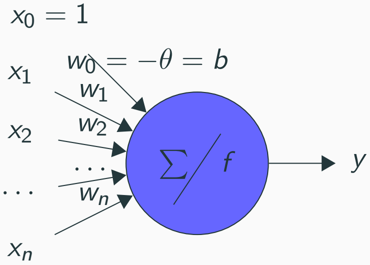
The quantity $-\theta$ is called the bias, and is usually denoted with b.

Simplify by converting θ into a weight

Original: $\mathbf{w} \cdot \mathbf{x} > \theta$

Rewritten: $\mathbf{w} \cdot \mathbf{x} - \theta > 0$

Rename $-\theta$ to b: $\mathbf{w} \cdot \mathbf{x} + b > 0$



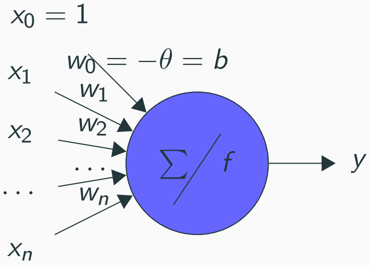
The quantity $-\theta$ is called the bias, and is usually denoted with b.

Simplify by converting θ into a weight

Original: $\mathbf{w} \cdot \mathbf{x} > \theta$

Rewritten: $\mathbf{w} \cdot \mathbf{x} - \theta > 0$

Rename $-\theta$ to b: $\mathbf{w} \cdot \mathbf{x} + b > 0$



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

Activation function: threshold

New input function:
$u(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$
Linear!

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

Activation state:
0 or 1

Nonlinear!

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 (that is, towards the classification boundary).

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 (that is, towards the classification boundary).

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 (that is, towards the classification boundary).

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 (that is, towards the classification boundary).

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 (that is, towards the classification boundary).

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 (that is, towards the classification boundary).

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

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

(t-o) gives the direction
x_i gives how much to update

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

- 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
```

update within batches could affect the weight changes

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

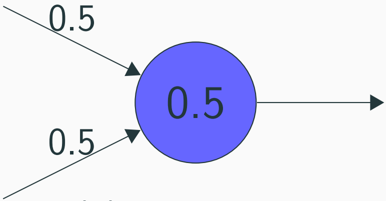
```
Initialize all weights randomly.
repeat
    for each training example do
        Apply the learning rule.
    end for
until the error is acceptable or a certain number of iterations is reached
```

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



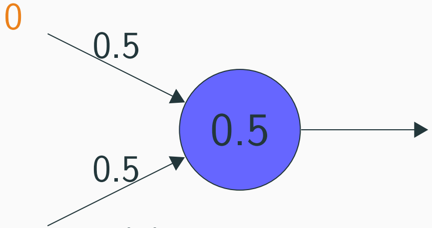
f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

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



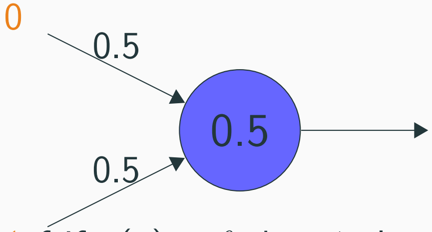
1 f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

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 $u(\mathbf{x}) \geq \theta$ then 1 else 0

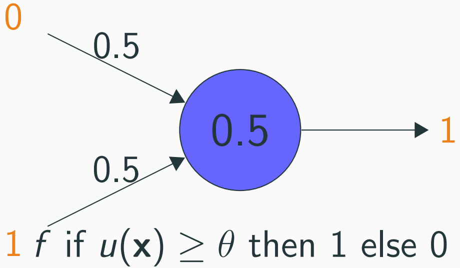
$$0 \cdot 0.5 + 1 \cdot 0.5 = 0.5$$

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



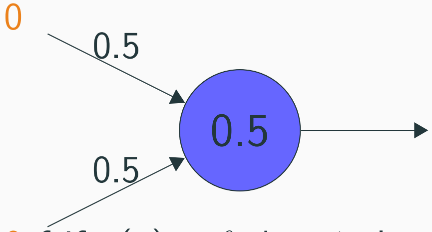
f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

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

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.

Perceptron for XOR



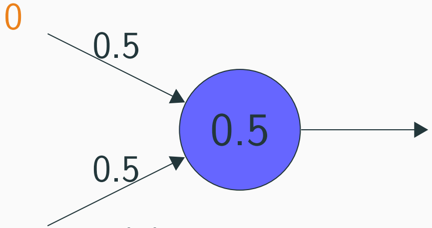
f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

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



0 f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

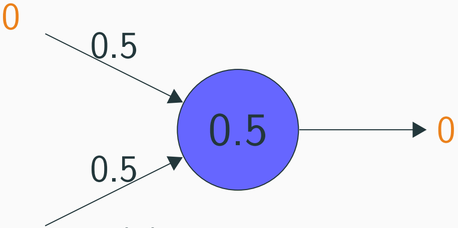
$$0 \cdot 0.5 + 0 \cdot 0.5 = 0$$

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



0 f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

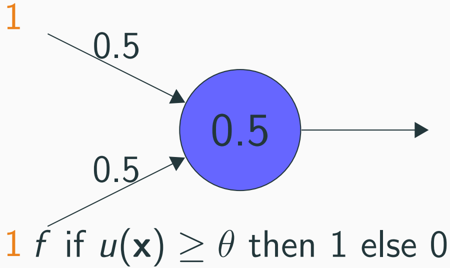
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

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

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

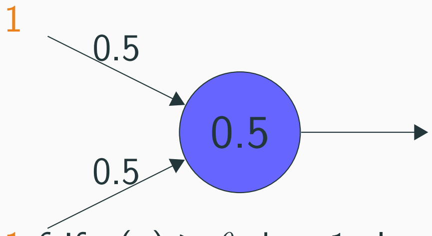


x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

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



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

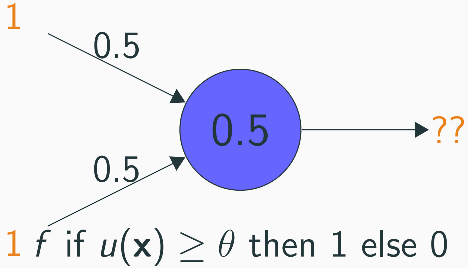
f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

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

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



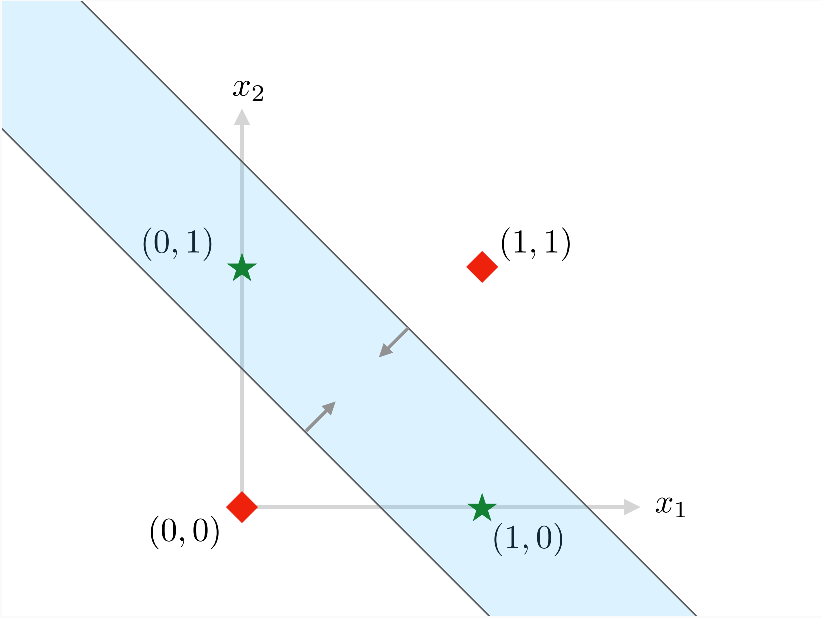
f if $u(\mathbf{x}) \geq \theta$ then 1 else 0

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

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

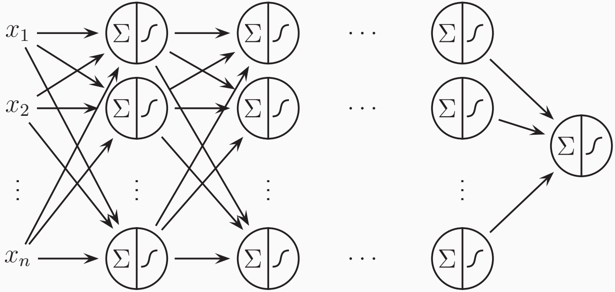
XOR is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.

Problem: XOR is not linearly separable



Multilayer perceptrons

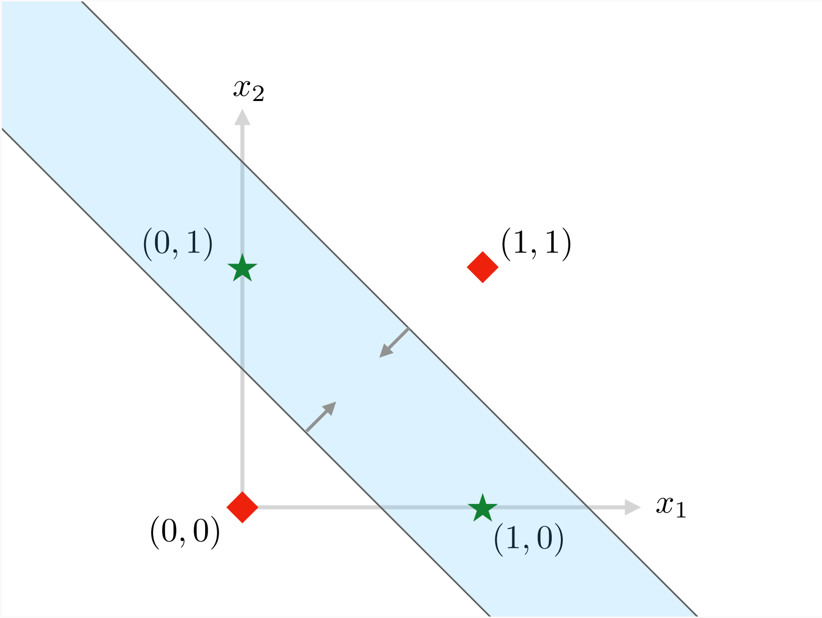
Multilayer Perceptrons (MLPs) are more expressive



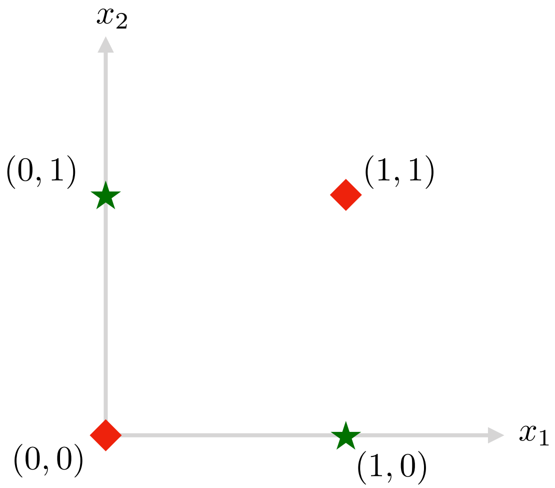
- MLPs are feed-forward neural networks, organized in layers.
- One input layer, one or more hidden layers, one output layer.
- Each node in a layer connected to all other nodes in next layer.
- Each connection has a weight (can be zero).

Q: How would you represent XOR?

MLP combines multiple separators



Hidden layer of MLP computes new representation of data

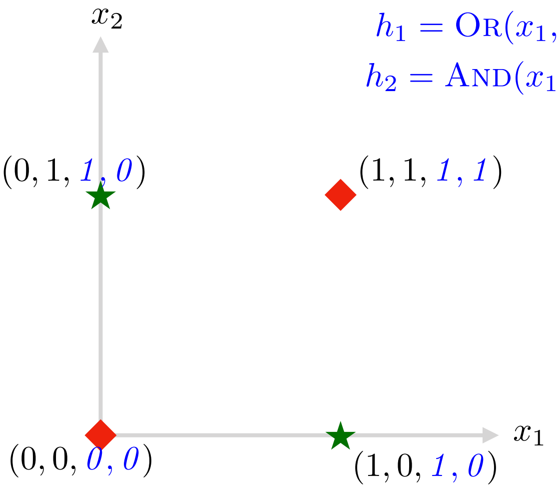


Hidden layer of MLP computes new representation of data

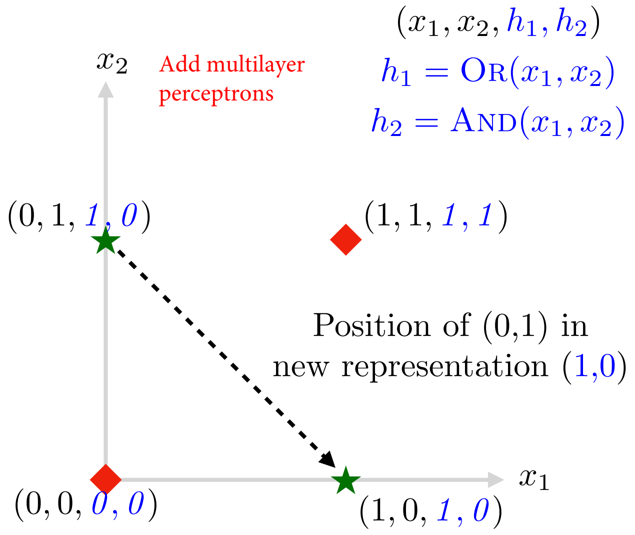
(x_1, x_2, h_1, h_2)

$h_1 = \textsc{Or}(x_1, x_2)$

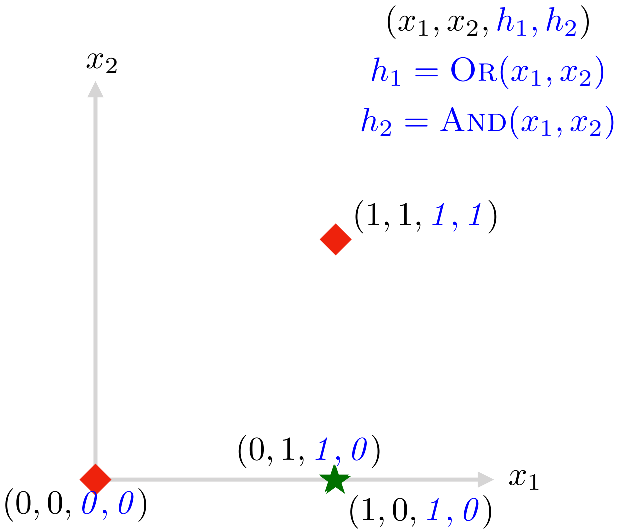
$h_2 = \textsc{And}(x_1, x_2)$



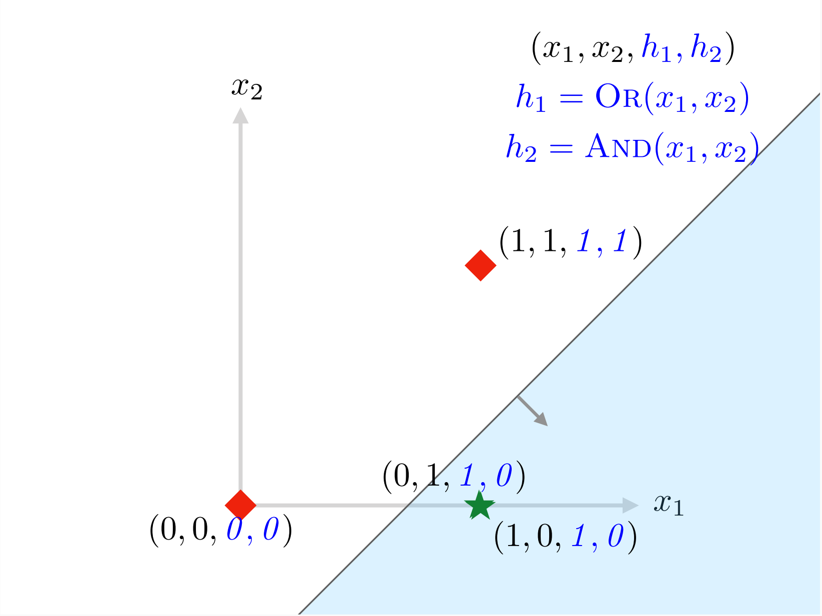
Hidden layer of MLP computes new representation of data



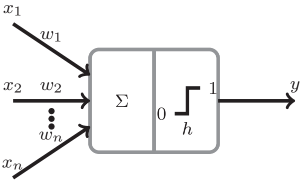
Hidden layer of MLP computes new representation of data



Hidden layer of MLP computes new representation of data

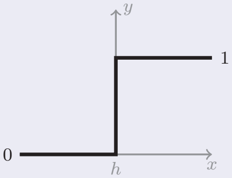


We can use activation functions other than thresholds



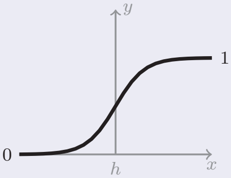
Q: Why many hidden layers rather than a big layer?
A: This is hard to train, overfitting, seperate the parameter into each samll hidden layers.

Step function



Outputs 0 or 1.

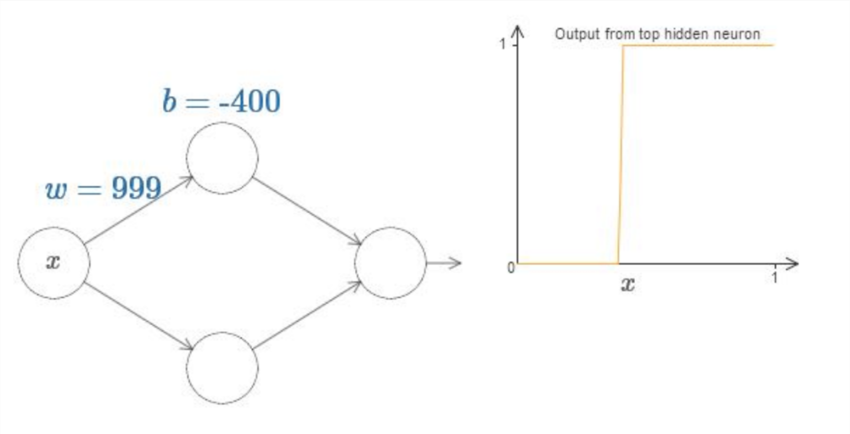
Sigmoid function



Outputs a real value between 0 and 1.

With sigmoid function we get **smooth transitions** instead of hard decision boundaries.

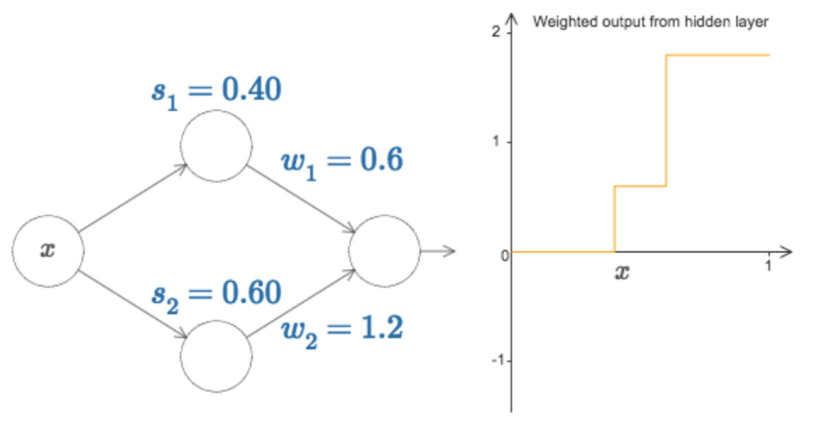
MLPs are universal function approximators



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

Source: http://neuralnetworksanddeeplearning.com

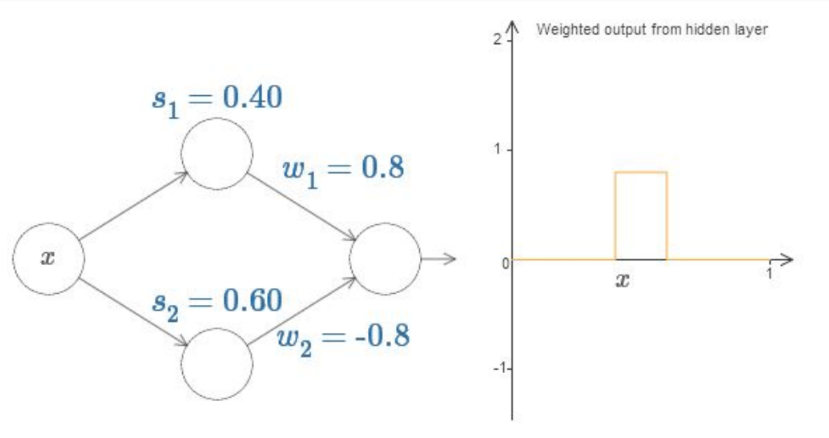
MLPs are universal function approximators



With multiple hidden units, you can get multiple steps.

Source: http://neuralnetworksanddeeplearning.com

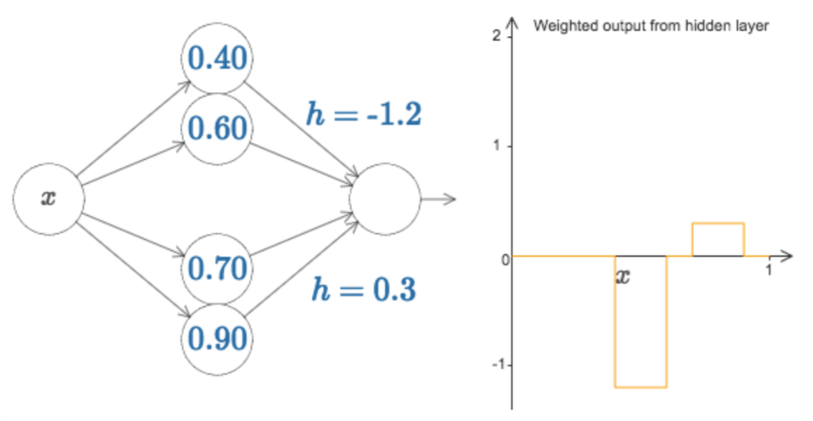
MLPs are universal function approximators



The steps can be positive or negative.

Source: http://neuralnetworksanddeeplearning.com

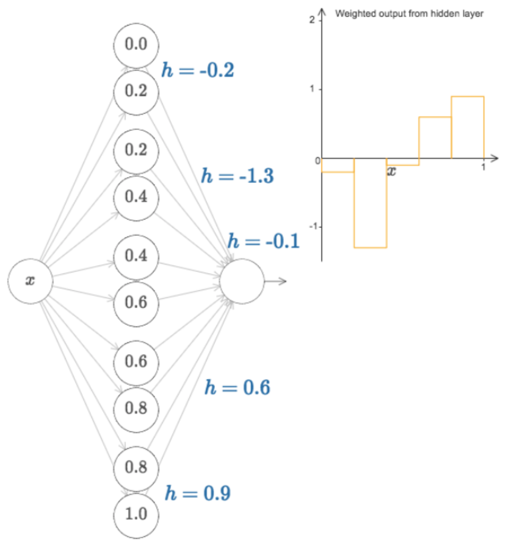
MLPs are universal function approximators



With more hidden units, you can represent more steps.

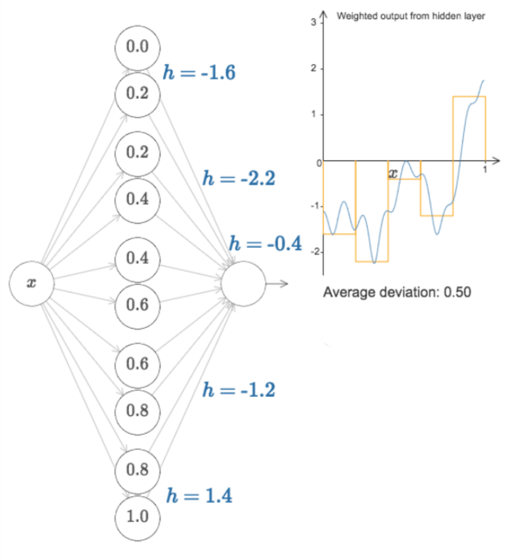
Source: http://neuralnetworksanddeeplearning.com

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

- A perceptron is a simple learnable function.
- A simple perceptron can express linearly separable functions.
- The perceptron itself is a non-linear function.
- There is a simple learning rule for the perceptron.
- A multilayer perceptron is more expressive than a perceptron, and in fact is a universal function approximator.
- The hidden layers of a multilayer perceptron compute a new representation of the input data.

Next lecture: We will use multilayer perceptrons to represent n-gram probabilities.

References

Jurafsky, Daniel, and James H. Martin. 2021. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Upper Saddle River, NJ: Pearson Education, draft of 3rd edn. https://web.stanford.edu/~jurafsky/slp3/.