Day 1 Lecture 3

Multi-Layer Perceptron

Acknowledgements: To my colleagues of this seminar and previous ones, specially Xavier Giró

https://telecombcn-dl.github.io/idl-2021/

Elisa Sayrol

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Department of Signal Theory and Communications
Image Processing Group

Outline

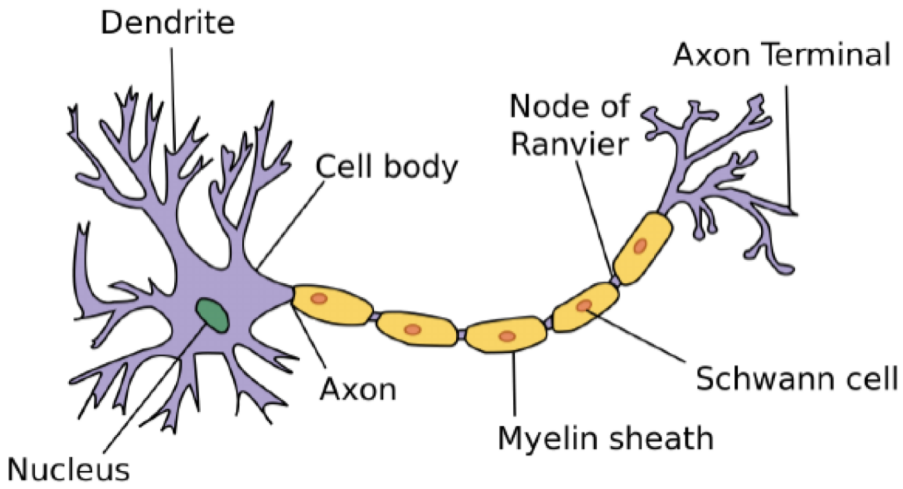
1. Single neuron model (Perceptron)
2. Multilayer Perceptron (MLP)
3. MNIST Example
4. Training

Single neuron model (perceptron)

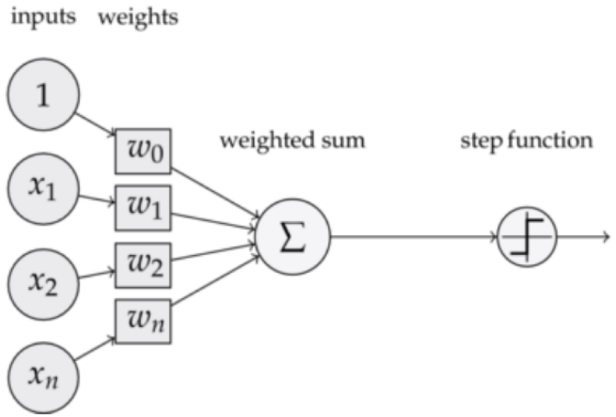
The Perceptron is seen as an **analogy** to a biological neuron.

Biological neurons fire an impulse once the sum of all inputs is over a threshold.

The perceptron acts like a switch (learn how in the next slides...).

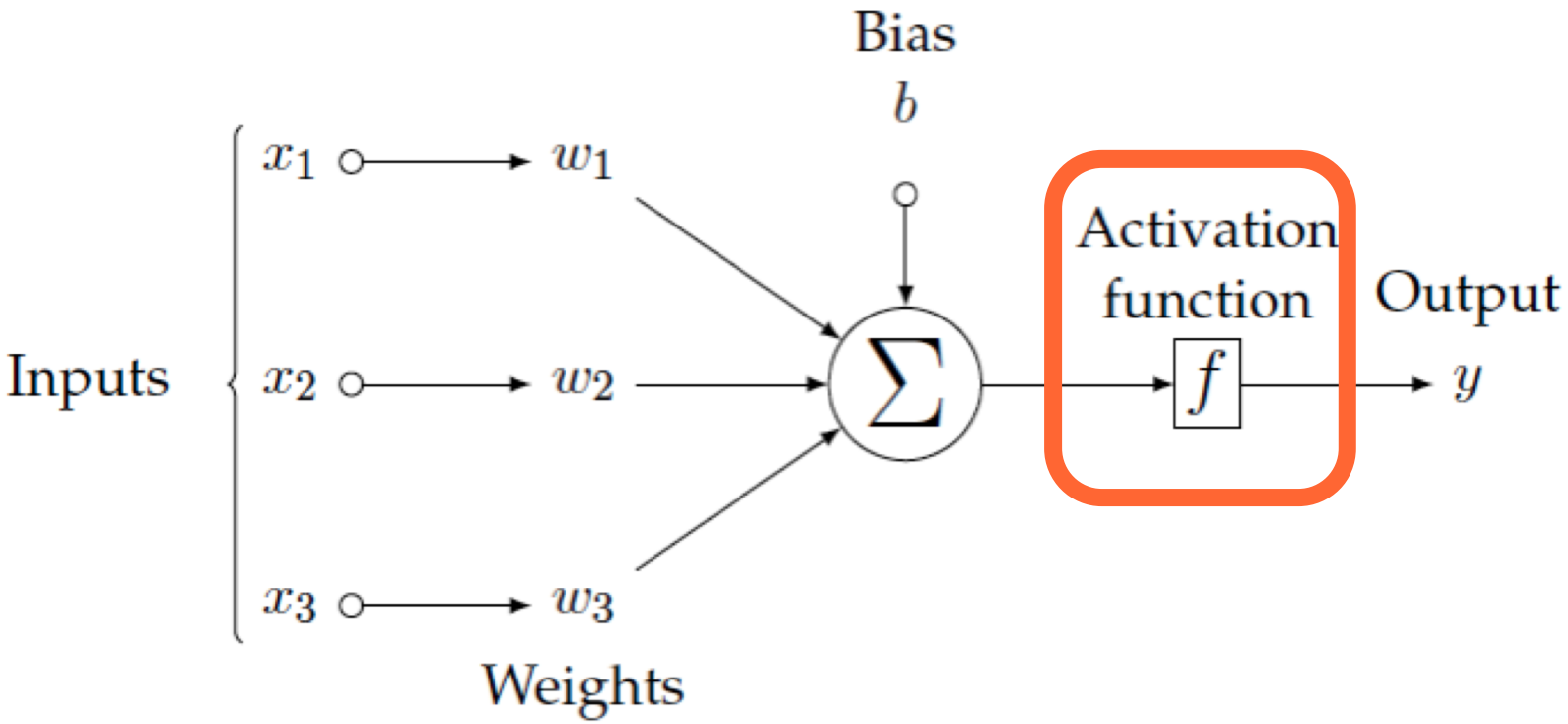


Rosenblatt's Perceptron (1958)



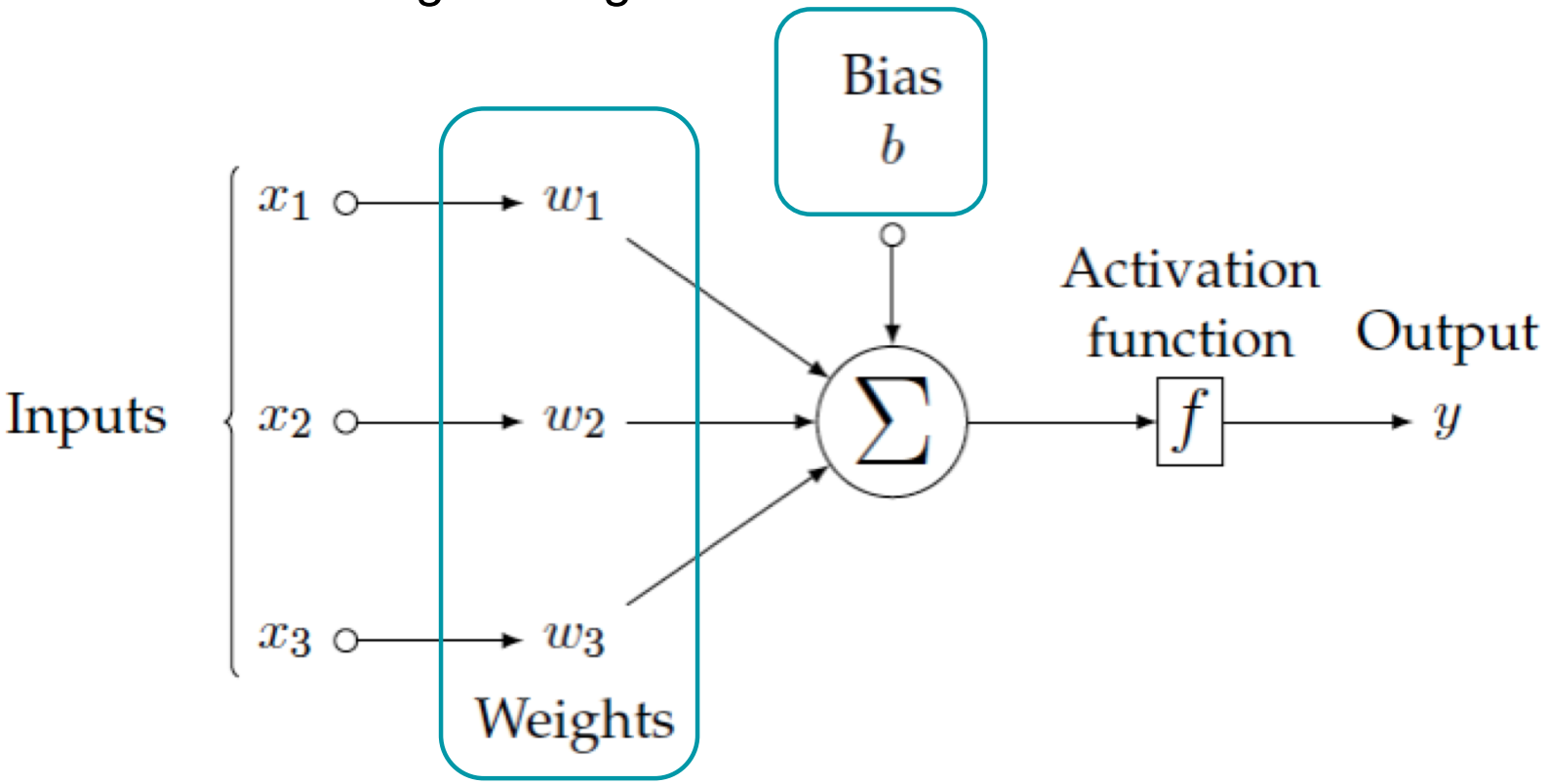
Single neuron model (perceptron)

The perceptron can address both regression or classification problems, depending on the chosen **activation function**.



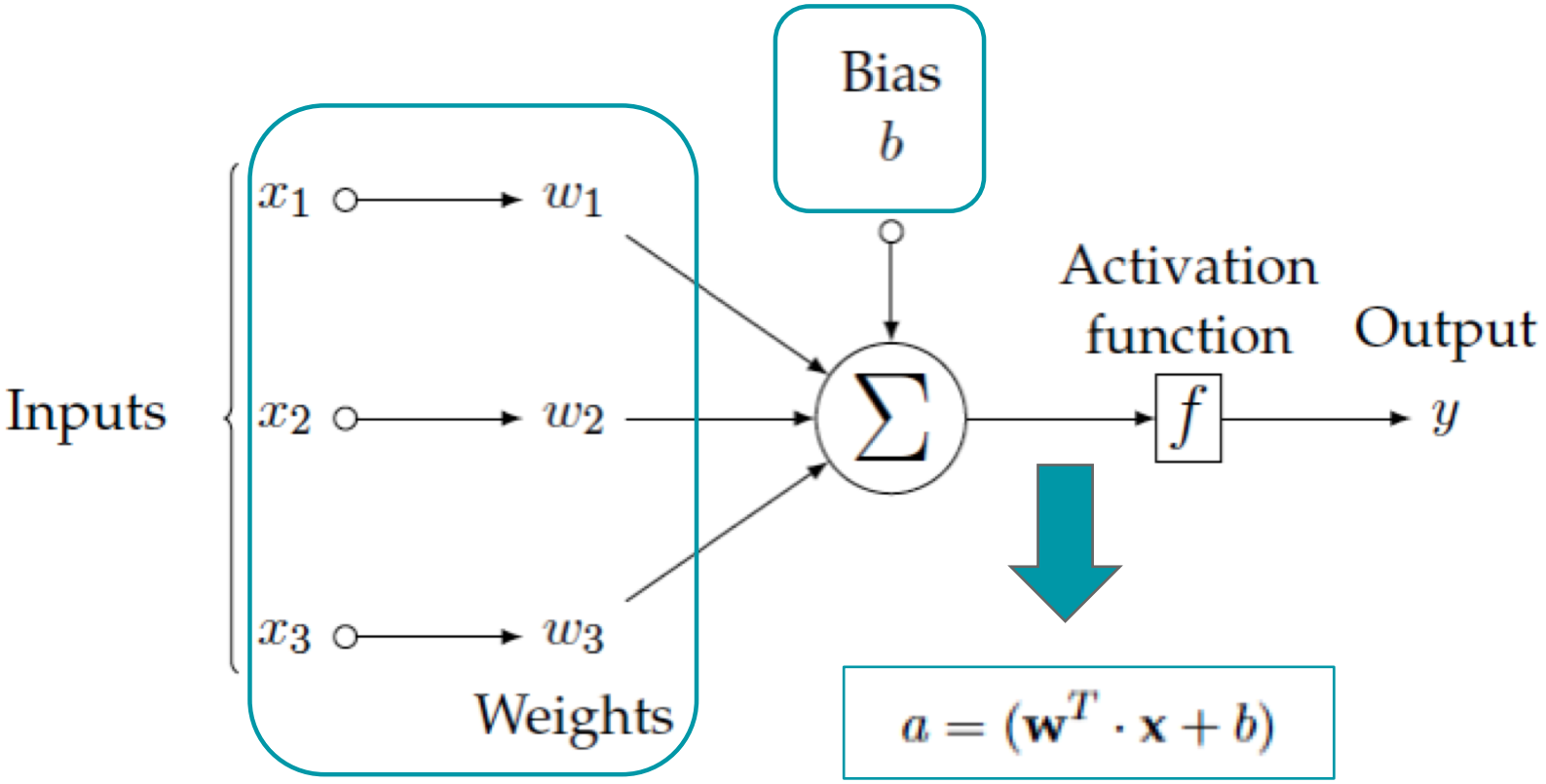
Single neuron model (perceptron)

Weights and bias are the parameters that define the behavior. They must be estimated during training.



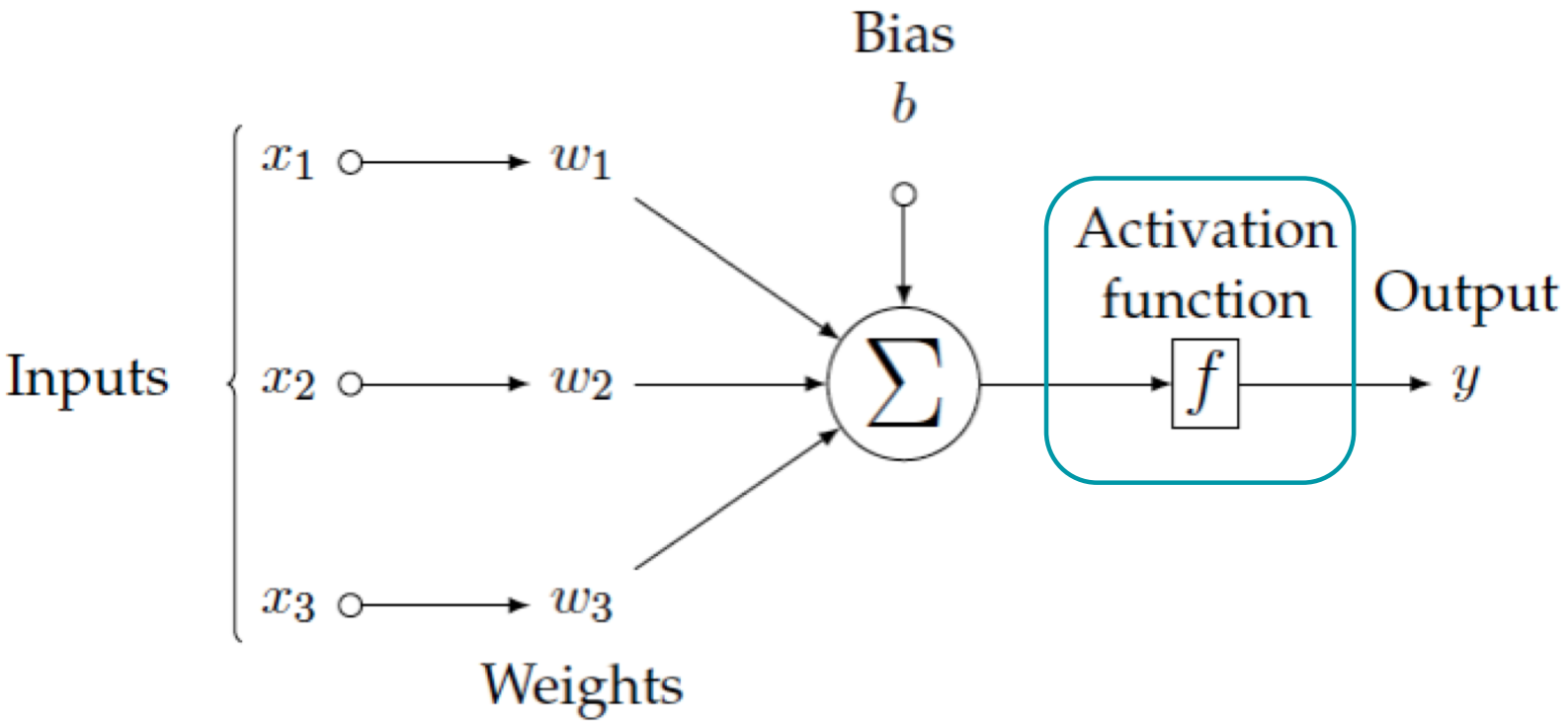
Single neuron model (perceptron)

The output y is derived from a sum of the **weighted** inputs plus a **bias** term**.**



Single neuron model (perceptron)

The **activation function** introduces non-linearities.



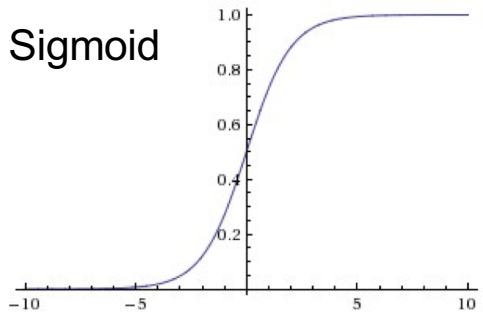
Single neuron model (perceptron)

Desirable properties

- Mostly smooth, continuous, differentiable
- Fairly linear

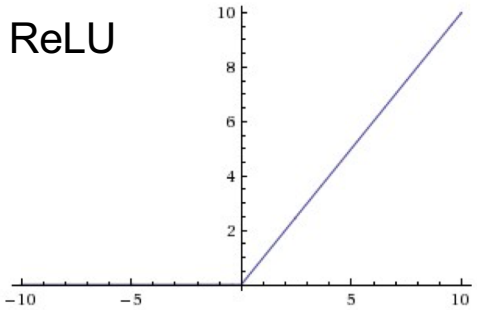
Common nonlinearities

- Sigmoid
- Tanh
- ReLU = max(0, x)



Why do we need non-linearities ?

If we only use linear layers we are only able to learn linear transformations of our input.



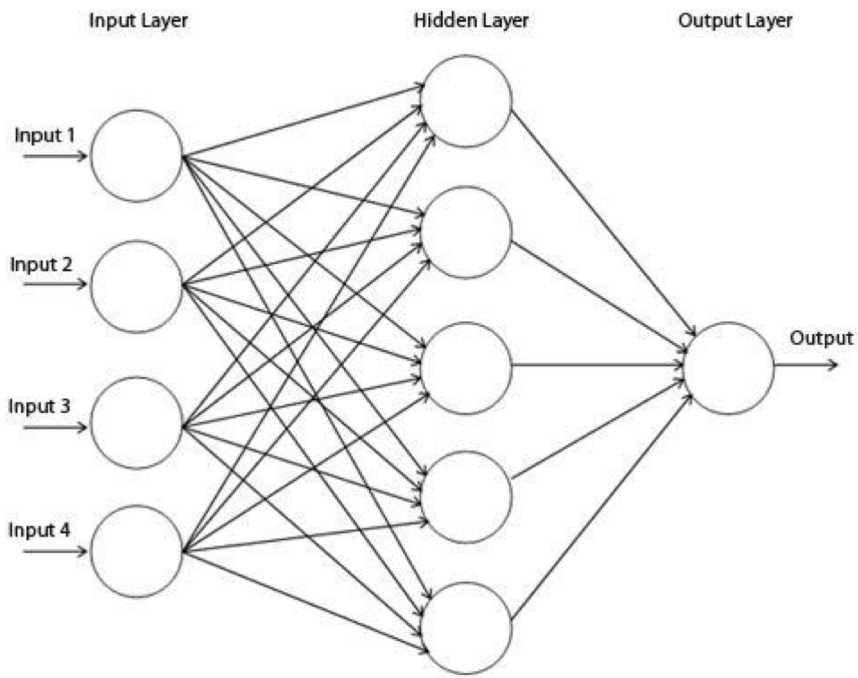
Neural Network

A neural network is simply a **composition** of simple neurons/ perceptrons into several layers

As we have seen each neuron simply computes a **linear combination** of its inputs, adds a bias, and passes the result through an **activation function**

The network can contain one or more **hidden layers**. The outputs of these hidden layers can be thought of as a new **representation** of the data (new features).

The final output is the **target** variable ($y = f_\theta(x)$)



Multi-layer Perceptron (MLP)

When each node in each layer is a linear combination of **all inputs from the previous layer** then the network is called a **multilayer perceptron** (MLP)

Weights can be organized into matrices.

Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

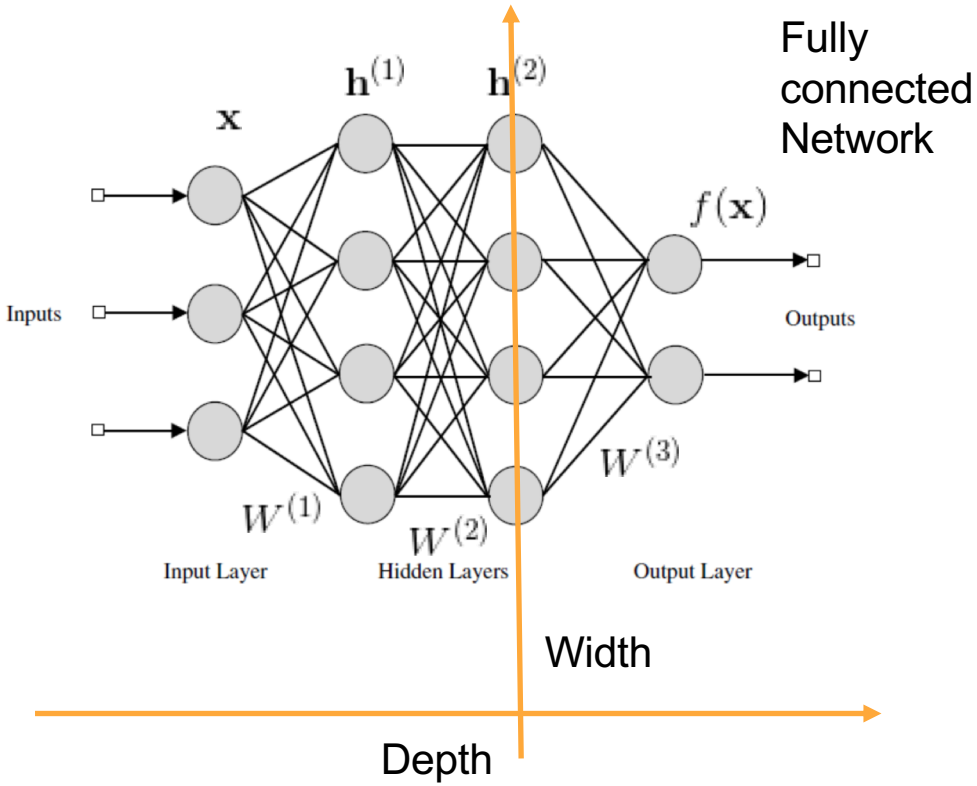
$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

y = f(x)

$\mathbf{h}^{(2)}=g(W^{(2)}\mathbf{h}^{(1)}+\mathbf{b}^{(2)})$

Fully connected Network



g: activation function. i.e. sigmoid f: target function. i.e. softmax

Multi-layer Perceptron (MLP)

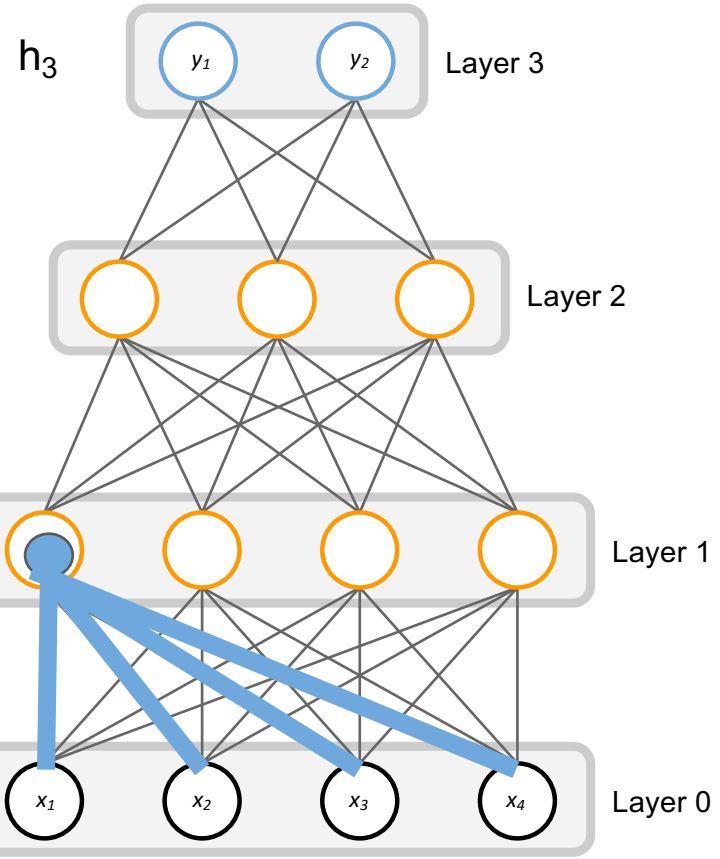
W_1 h_0 b_1

w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}
w_{31}	w_{32}	w_{33}	w_{34}
w_{41}	w_{42}	w_{43}	w_{44}

x_1
x_2
x_3
x_4

b_1
b_2
b_3
b_4

$h_{11} = g(\mathbf{w}\mathbf{x} + b)$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

Multi-layer Perceptron (MLP)

W_1 h_0 b_1

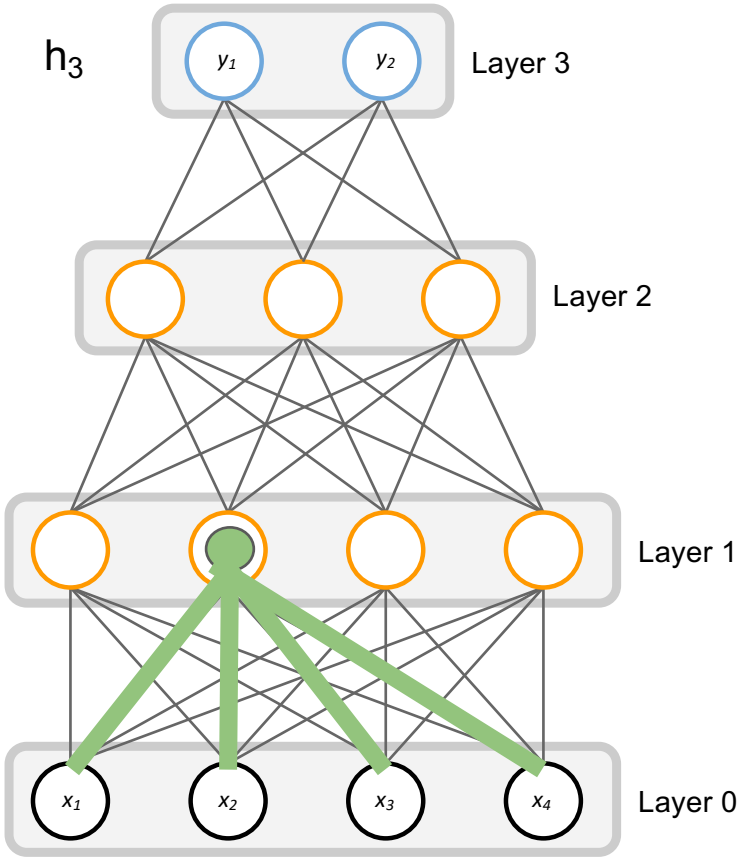
w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}
w_{31}	w_{32}	w_{33}	w_{34}
w_{41}	w_{42}	w_{43}	w_{44}

x_1
x_2
x_3
x_4

b_1
b_2
b_3
b_4

$h_{11} = g(\mathbf{wx} + b)$

$h_{12} = g(\mathbf{wx} + b)$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

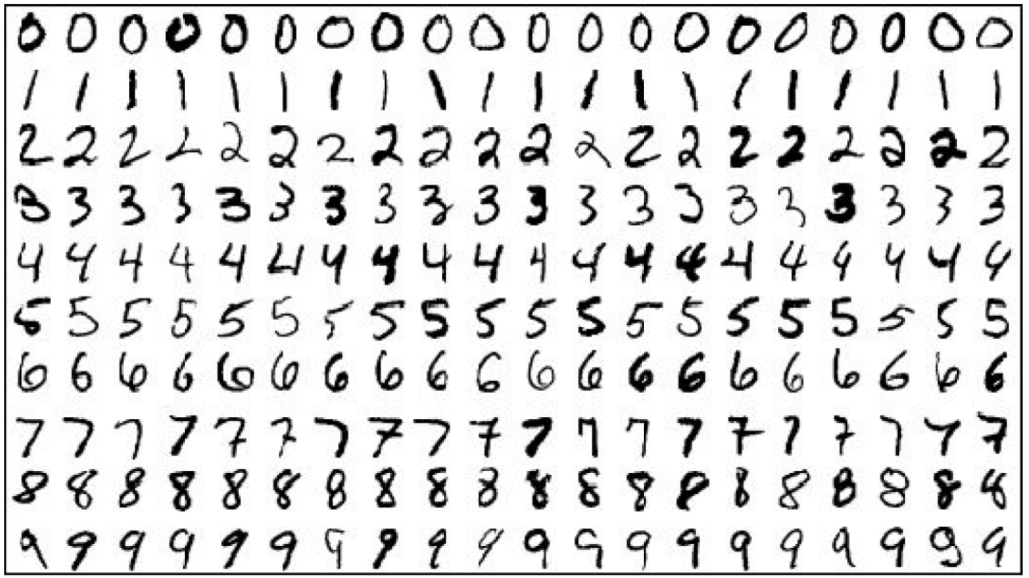
$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

MNIST Example

Handwritten digits

- 60.000 training examples
- 10.000 test examples
- 10 classes (digits 0-9)
- 28x28 grayscale images(784 pixels)
- http://yann.lecun.com/exdb/mnist/



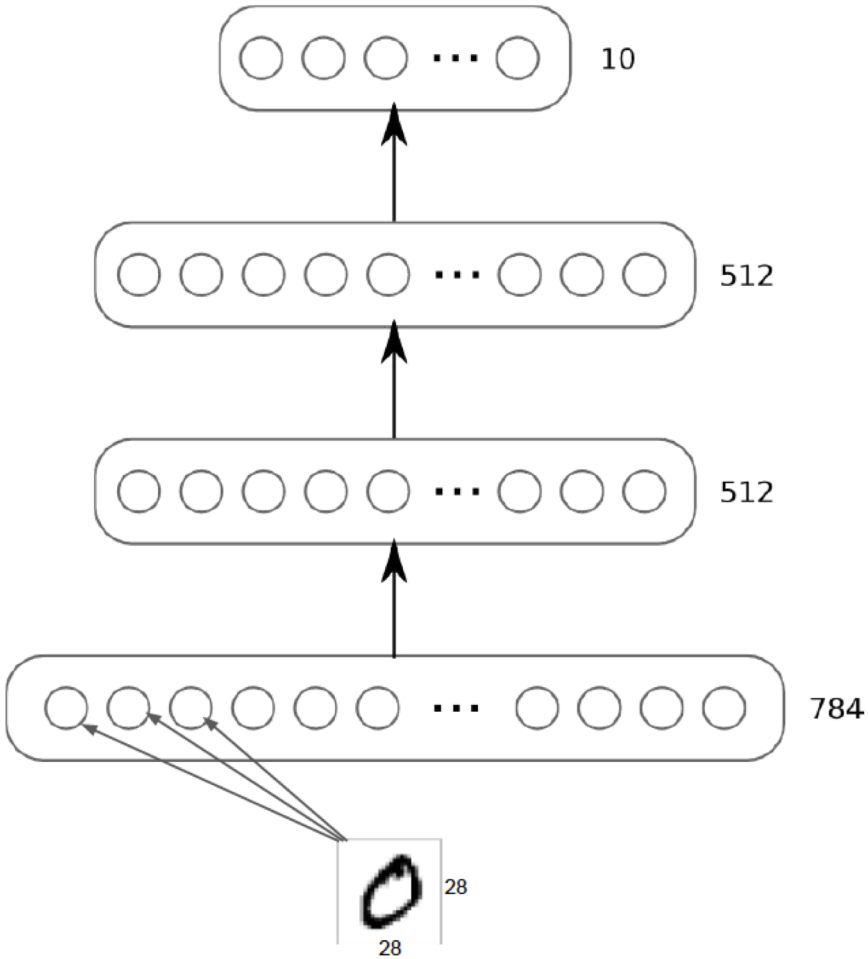
The objective is to learn a function that predicts the digit from the image

MNIST Example

Model

- 3 layer neural-network (2 hidden layers)
- Tanh units (activation function)
- 512-512-10
- Softmax on top layer
- *Cross Entropy Loss*

Layer	#Weights	#Biases	Total
1	784 x 512	512	401,920
2	512 x 512	512	262,656
3	512 x 10	10	5,130
			669,706



MNIST Example

Training

- 40 epochs using min-batch SGD
- Batch Size: 128
- Leaning Rate: 0.1 (fixed)
- Takes 5 minutes to train on GPU

Metrics

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

there are other metrics….

Accuracy Results

- 98.12% (188 errors in 10.000 test examples)

there are ways to improve accuracy…

Training

Estimate parameters $\theta(W^{(t)}, b^{(t)})$ from training examples given a Loss Function

$$\Theta^{*} = argmin_{\theta} \mathcal{L}(f_{\theta}(x), y)$$

- Iteratively adapt each parameter

  Basic idea: **gradient descent.**

- Dependencies are very complex.

  Global minimum: challenging. Local minima: can be good enough.

- Initialization influences the solutions.

Training

Gradient Descent: Move the parameter $\theta_j$in small steps in the direction opposite sign of the derivative of the loss with respect j.

$$\theta^{(n)} = \theta^{(n-1)} - \alpha^{(n-1)} \cdot \nabla_\theta \mathcal{L}\big(y, f(x)\big) - \lambda\theta^{(n-1)}$$

- Stochastic gradient descent (SGD): estimate the gradient with one sample, or better, with a **minibatch** of examples.
- **Momentum:** the movement direction of parameters averages the gradient estimation with previous ones.
- Several strategies have been proposed to update the weights: Adam, RMSProp, Adamax, etc. known as: **optimizers**

Training MLPs

With **Multiple Layer Perceptrons** we need to find the gradient of the loss function with respect to all the parameters of the model ($W^{(k)}$, $b^{(k)}$)

These can be found using the **chain rule** of differentiation.

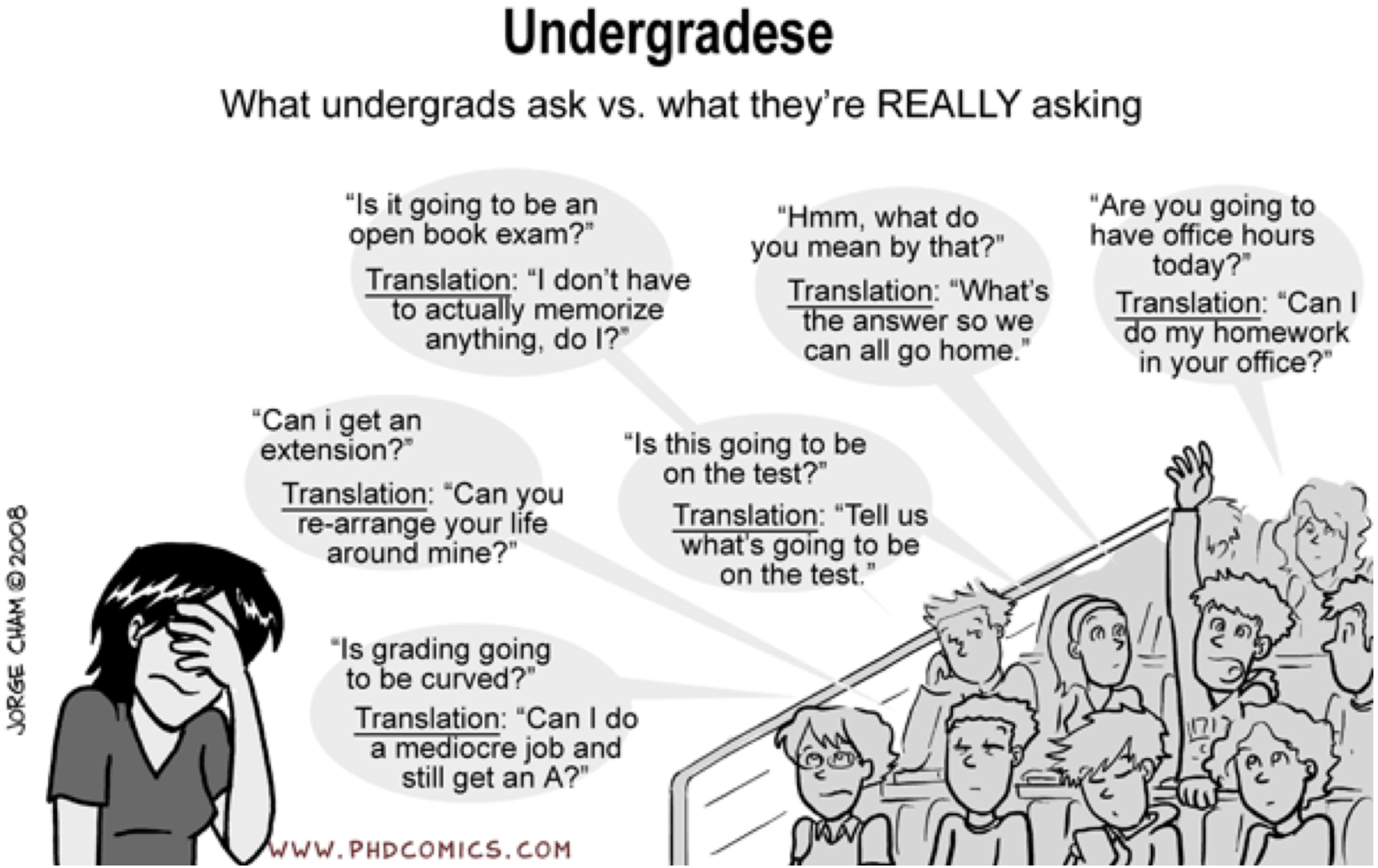
The calculations reveal that the gradient wrt the parameters in layer k only depends on the error from the above layer and the output from the layer below.

This means that the gradients for each layer can be computed iteratively, starting at the last layer and propagating the error back through the network. This is known as the **backpropagation** algorithm.

Summary

- Multi-layer Perceptron Networks allow us to build **non-linear decision boundaries**
- Multi-layer Perceptron Networks are composed of the input layer, hidden layers and the output layer. **All neurons** in one layer **are connected to all** neurons from the previous layer and the layer that follows
- Multi-layer Perceptron Networks have a **large number of parameters** that have to be estimated trough training with the goal of minimizing a given **loss function**
- With Multi-layer Perceptron Networks we need to find the gradient of the loss function with respect to all the parameters of the model ($W^{(t)}$, $b^{(t)}$)

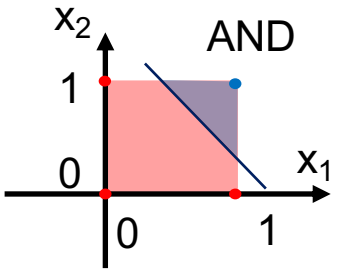
Questions ?

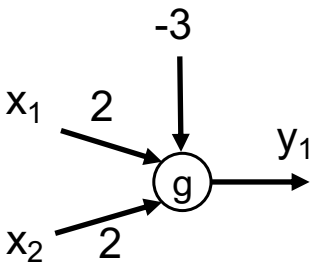

Undergradese
What undergrads ask vs. what they're REALLY asking
"Is it going to be an open book exam?"
Translation: "I don't have to actually memorize anything, do I?"
"Hmm, what do you mean by that?"
Translation: "What's the answer so we can all go home."
"Are you going to have office hours today?"
Translation: "Can I do my homework in your office?"
"Can i get an extension?"
Translation: "Can you re-arrange your life around mine?"
"Is this going to be on the test?"
Translation: "Tell us what's going to be on the test."
"Is grading going to be curved?"
Translation: "Can I do a mediocre job and still get an A?"
JORGE CHAM © 2008
WWW.PHDCOMICS.COM

Additional material

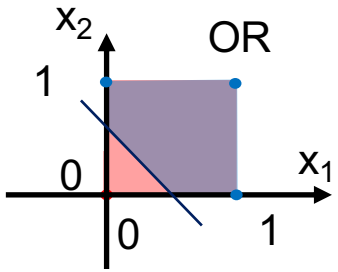
Example: X-OR.

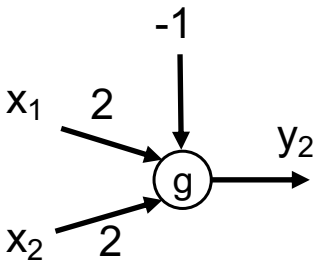
AND and OR can be generated with a single perceptron





Input vector (x1,x2)	Class AND
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1





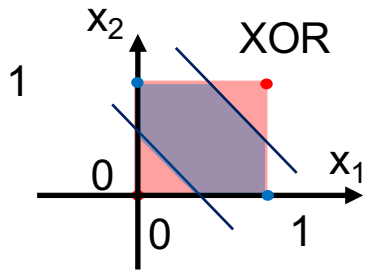
Input vector (x1,x2)	Class OR
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	1

$$y_1 = g\left(\mathbf{w}^T\mathbf{x} + b\right) = u\left((2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 3\right)$$

$$y_2 = g\left(\mathbf{w}^T\mathbf{x} + b\right) = u\left((2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 1\right)$$

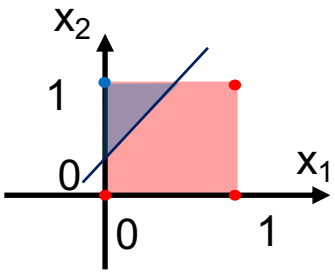
Example: X-OR

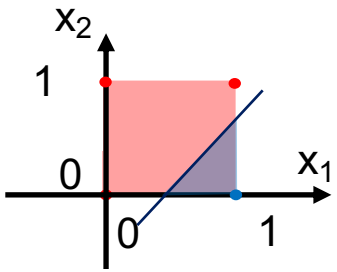
X-OR a Non-linear separable problem can not be generated with a single perceptron

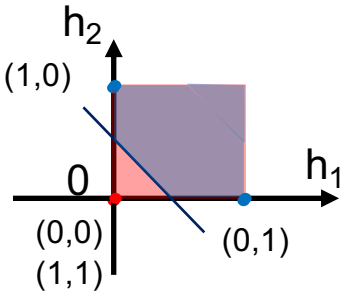


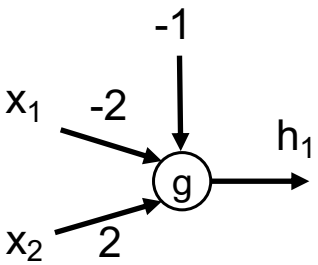
Input vector (x1,x2)	Class XOR
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

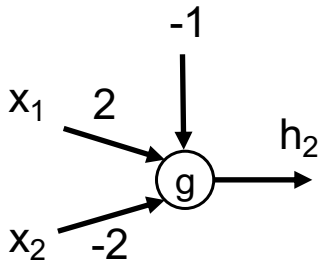
Example: X-OR. However…..

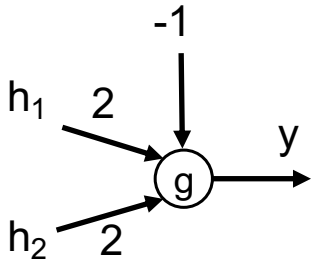










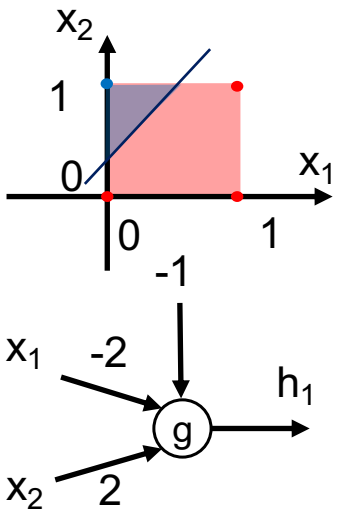


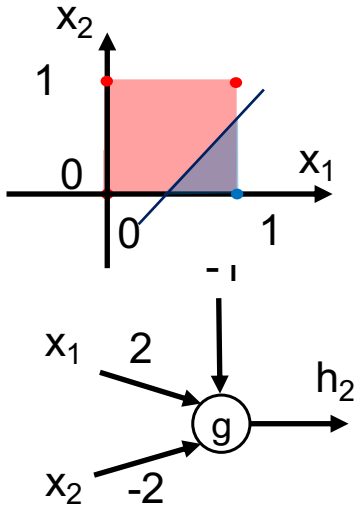
$$h_1 = g\left(\mathbf{w}_{11}^T\,\mathbf{x} + b_{11}\right) = u\left((-2 \quad 2)\cdot\begin{pmatrix}x_1\\x_2\end{pmatrix} - 1\right)$$

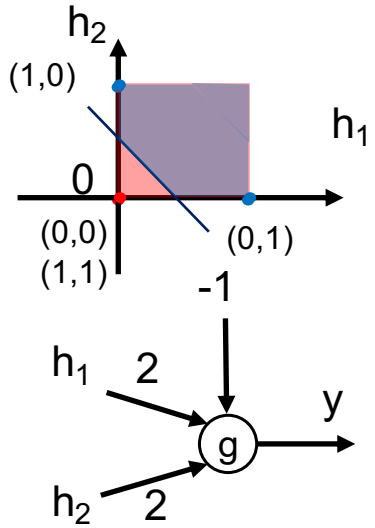
$$h_2 = g\left(\mathbf{w}_{12}^T \quad \mathbf{x} + b_{12}\right) = u\left((2 \quad -2)\cdot\begin{pmatrix}x_1\\x_2\end{pmatrix} - 1\right)$$

$$y = g\left(\mathbf{w}_2^T \quad \mathbf{h} + b_2\right) = u\left((2 \quad 2)\cdot\begin{pmatrix}h_1\\h_2\end{pmatrix} - 1\right)$$

Example: X-OR. However…..







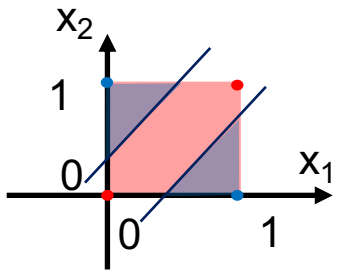
Input vector (x1,x2)	h_1
(0,0)	0
(0,1)	1
(1,0)	0
(1,1)	0

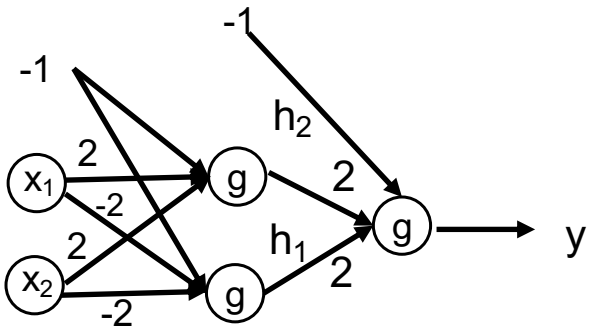
Input vector (x1,x2)	h_1
(0,0)	0
(0,1)	0
(1,0)	1
(1,1)	0

Input vector (h1,h2)	y_1
(0,0)	0
(0,1)	1
(1,0)	1

Input vector (x1,x2)	y_1
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

Example: X-OR. Finally





Input layer | Hidden layer | Output Layer

Three layer Network:

-Input Layer
-Hidden Layer
-Output Layer

2-2-1 Fully connected topology
(all neurons in a layer connected Connected to all neurons in the following layer)

$$h_1 = g\left(\mathbf{w_{11}^T}\,\mathbf{x} + b_{11}\right) = u\left((-2 \quad 2)\cdot\begin{pmatrix}x_1\\x_2\end{pmatrix} - 1\right)$$

$$h_2 = g\left(\mathbf{w_{12}^T} \quad \mathbf{x} + b_{12}\right) = u\left((2 \quad -2)\cdot\begin{pmatrix}x_1\\x_2\end{pmatrix} - 1\right)$$

$$y = g\left(\mathbf{w_2^T} \quad \mathbf{h} + b_2\right) = u\left((2 \quad 2)\cdot\begin{pmatrix}h_1\\h_2\end{pmatrix} - 1\right)$$