Deep Learning: Session 3

Outline

1. Recap
2. Artificial Neural Networks
3. Activation Functions
4. The XOR Problem

In previous chapters...

Quick Remind! Bayes Rule

$$\underbrace{P(y \mid x)}_{\text{Posterior}} = \frac{P(x,y)}{P(x)} = \frac{P(x \mid y)P(y)}{P(x)} \propto \underbrace{P(x \mid y)}_{\text{likelihood}} \underbrace{P(y)}_{\text{Prior}}$$

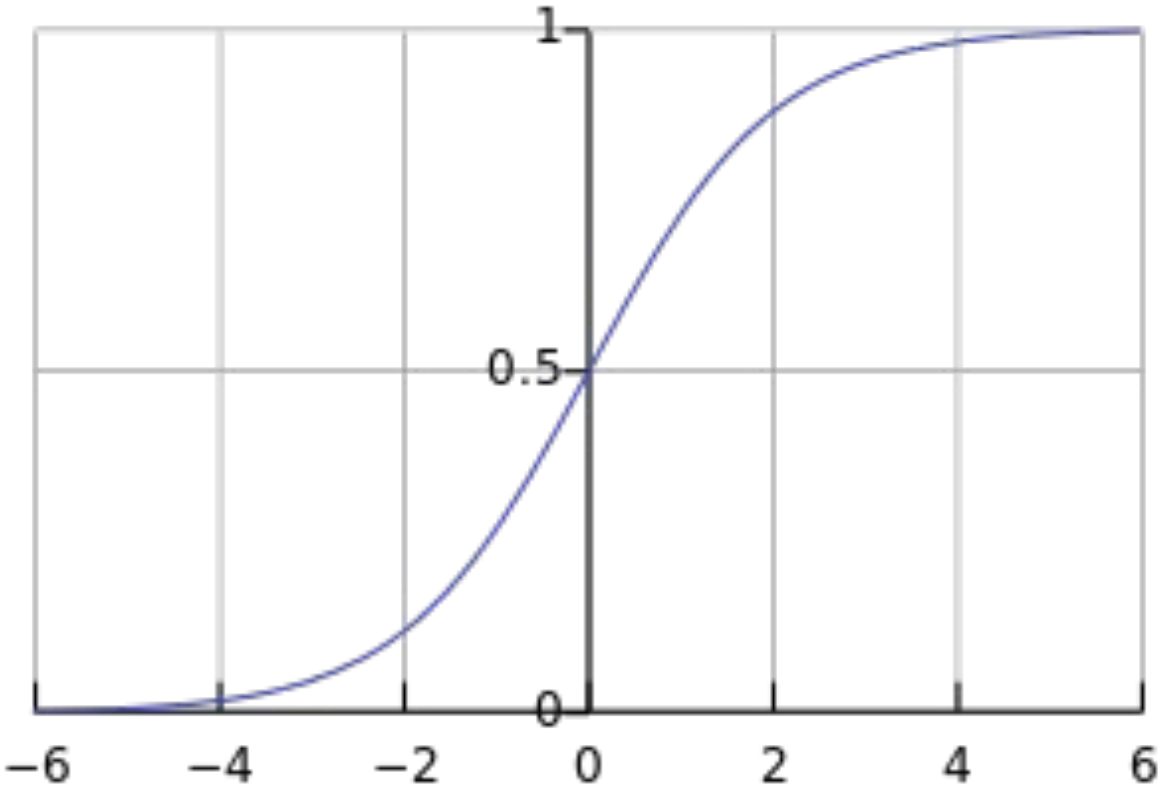
In previous chapters...

… before we really start with:

- Watch out with the name! Logistic Regression is a **CLASSIFICATION** technique
- **Supervised** learning. Thus, we have a training labeled dataset.
- Follow a **discriminative** approach. Thus, it attempts to model the conditional probability p(y | x)
- As a machine learning technique, we have a: mapping function, a cost function and a learning algorithm.

In previous chapters...

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



- This meaning …

$$0 \le h_w(x) = \sigma(w^t x + b) \le 1$$

Using for binary classification

Decide y = 1 if $h_w(x) \ge 0.5$

Decide y = 0 if $h_w(x) < 0.5$

In previous chapters...

- Inputs & outputs
  - Email containing (counting on different words): spam or not spam (binary)
  - Speech signals: Language spoken (multiclass)
  - Faces pictures: is a pretty kitty or not? (binary)
- Hypothesis or mapping function

$$h_w(x) = \sigma(w^t x + b)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

In previous chapters...

- Better cost function: log loss function

$$J(W) = \frac{1}{N}\sum_{i=1}^{N} c(h_w(x_i), y_i)$$

$$c(h_w(x_i), y_i) \begin{cases} -\log h_w(x) \, if \; y = 1 \\ -\log(1 - h_w(x)) \, if \; y = 0 \end{cases}$$

In previous chapters...

- Multiclass Logistic regression mapping function: **Softmax function**

$$P(y=j \mid x)=\frac{e^{w_j^t x+b}}{\sum_{i=k}^{K} e^{w_j^t x+b}}$$

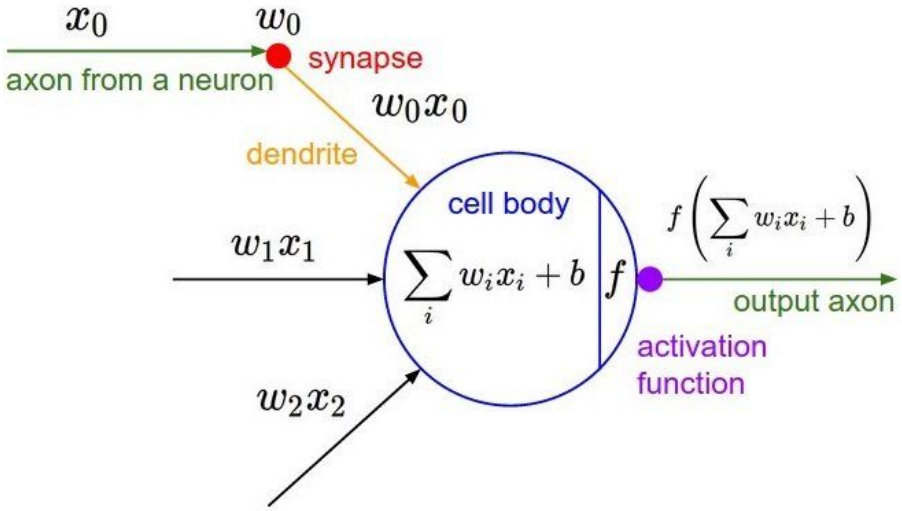
- The outputs of the new mapping function can be directly interpreted as probabilities for classes as:

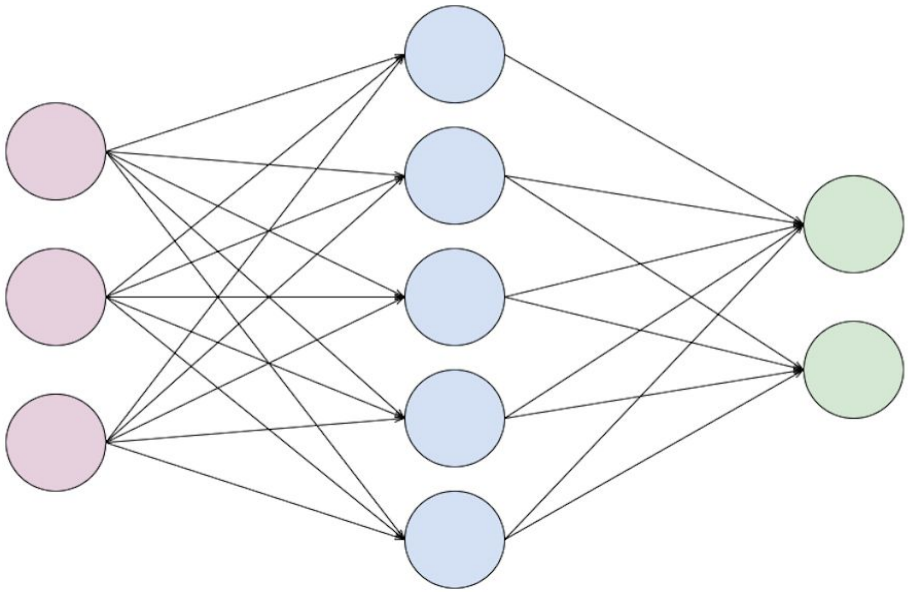
$$\sum_{i=1}^{C} P(y=j \mid x)=1$$

- Multiclass Logistic regression cost function: **Cross-Entropy loss func.**

$$-\sum_{i=1}^{N} y \log(\hat{y})=-\sum_{i=1}^{N} y \log(h_w(x_i))=-\sum_{i=1}^{N} y \log(\sigma(w^t x_i+b))$$

In previous chapters...





Logistic regression can be seen as a ‘degenerated’ neural net with one single node in a single output layer

Artificial Neural Networks

- When we think about complex learning, the human brain comes to scene.

Artificial Neural Networks

- When we think about complex learning, the human brain comes to scene.
- The brain is a highly complex, nonlinear and parallel information processing system.

Artificial Neural Networks

- When we think about complex learning, the human brain comes to scene.
- The brain is a highly complex, nonlinear and parallel information processing system.
- Brain has the capability to organize its structural constituents, known as neurons, so as to perform certain complex computations (pattern recognition, perception and motor control). Examples: human vision, sonar of a bat.

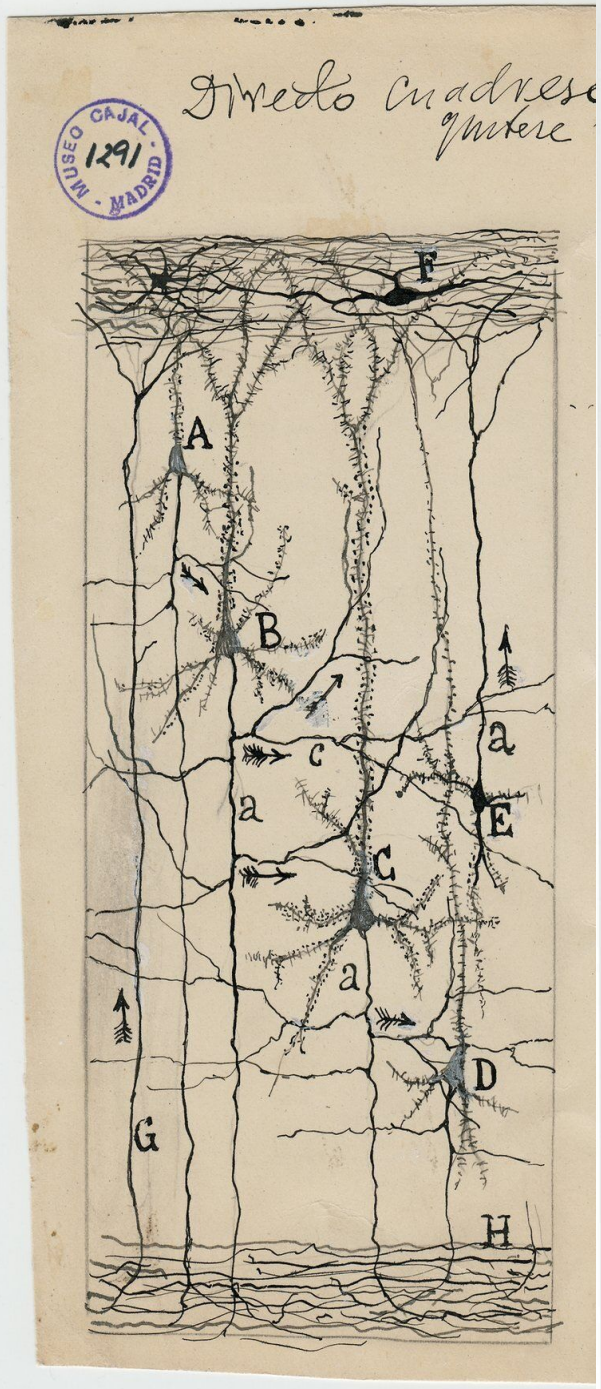
Artificial Neural Networks

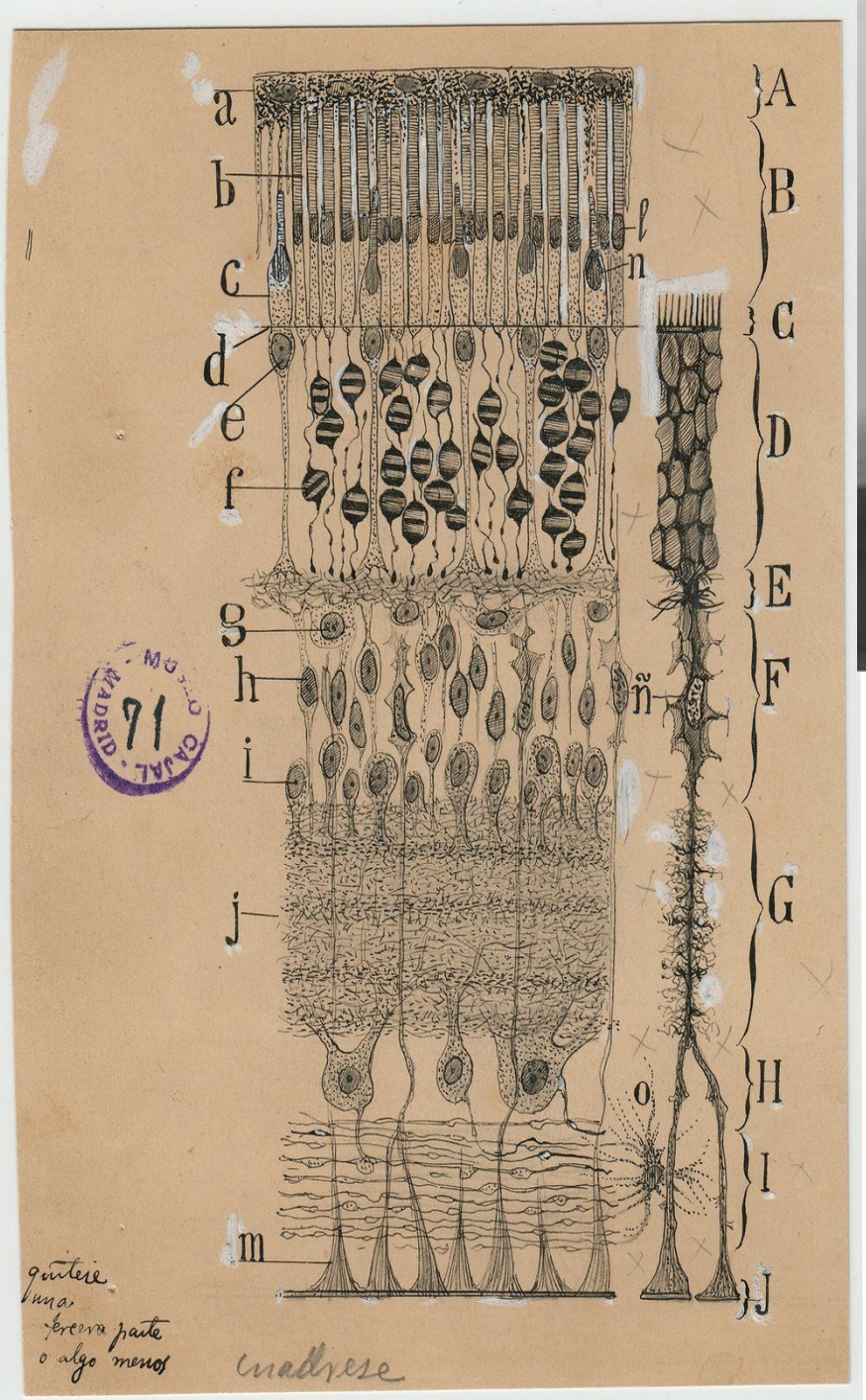
- When we think about complex learning, the human brain comes to scene.
- The brain is a highly complex, nonlinear and parallel information processing system.
- Brain has the capability to organize its structural constituents, known as neurons, so as to perform certain complex computations (pattern recognition, perception and motor control). Examples: human vision, sonar of a bat.
- At birth, a brain has great structure and the ability to build up its own rules through what we usually refer to as “experience”. Experience is built up over time.

Artificial Neural Networks

- When we think about complex learning, the human brain comes to scene.
- The brain is a highly complex, nonlinear and parallel information processing system.
- Brain has the capability to organize its structural constituents, known as neurons, so as to perform certain complex computations (pattern recognition, perception and motor control). Examples: human vision, sonar of a bat.
- At birth, a brain has great structure and the ability to build up its own rules through what we usually refer to as “experience”. Experience is built up over time.

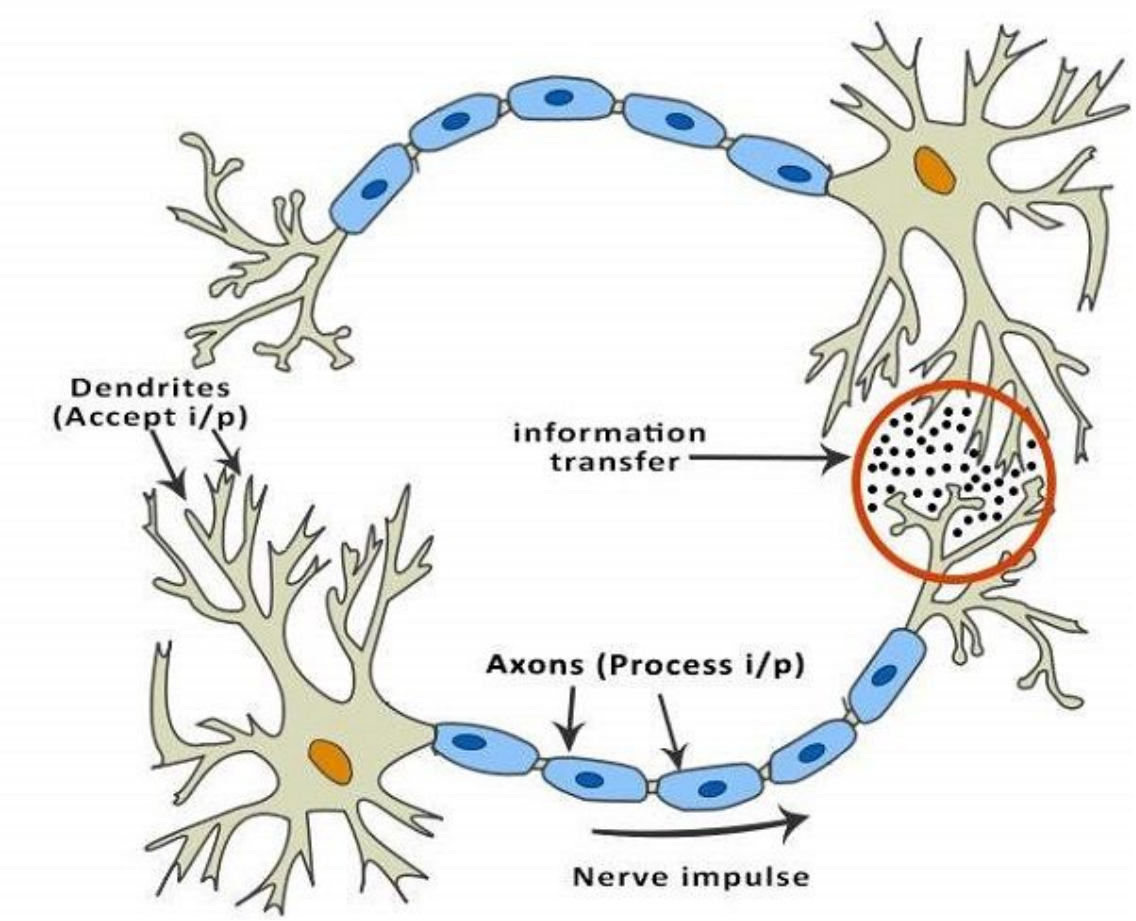
Machine Learning system definition!!!

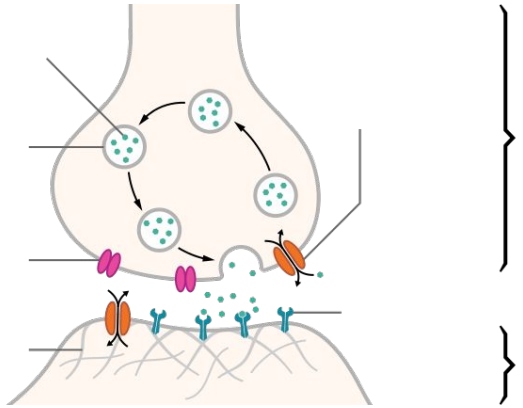




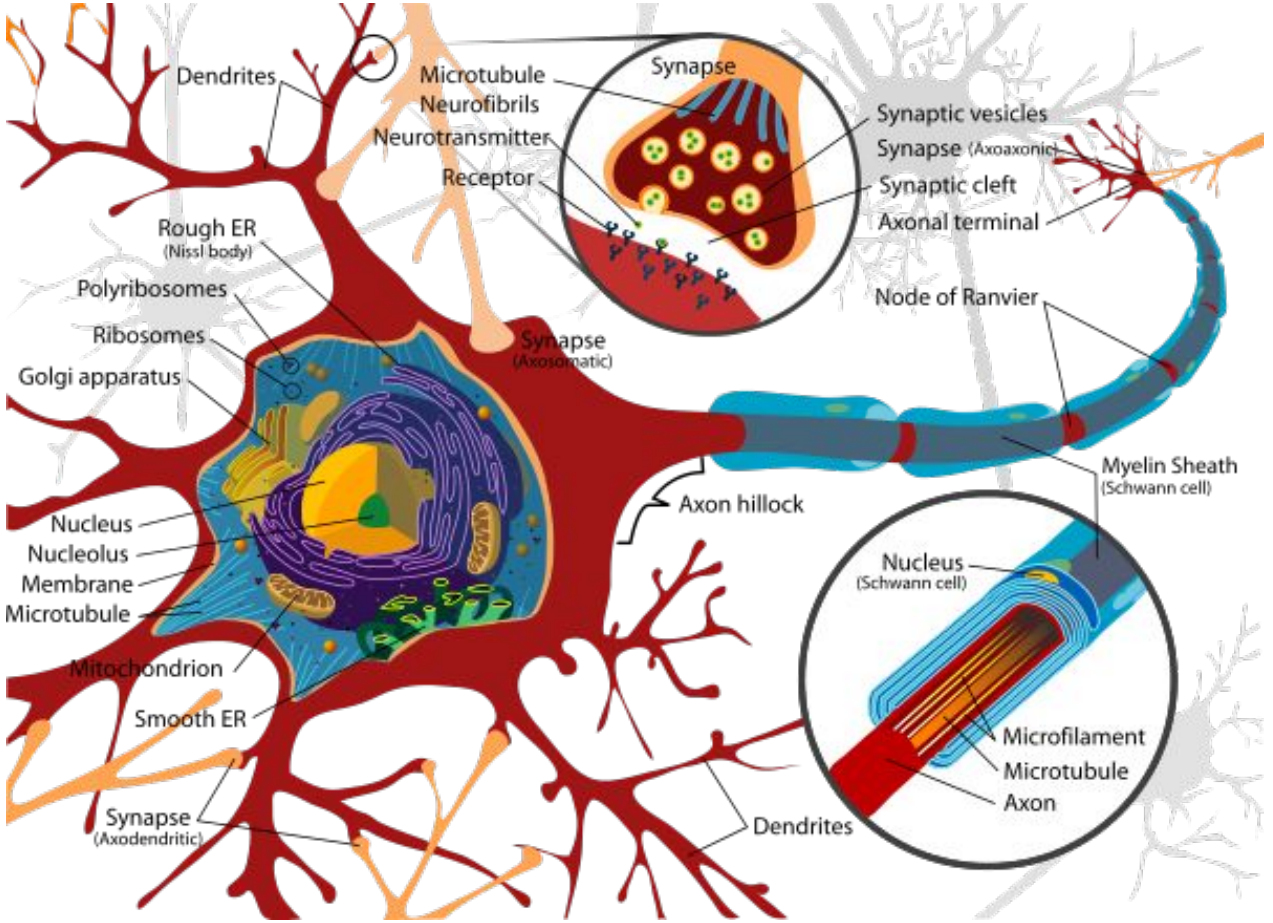
- Ramón y Cajal / Camilo Golgi. Nobel Prize 1906
- The neuron doctrine and the theory of dynamic polarization.
- The neuron as an isolated cell (not physically connected to others)
- Where the information flows from axon to next cell dendrites by an electrical polarization

Neurons: the biological model

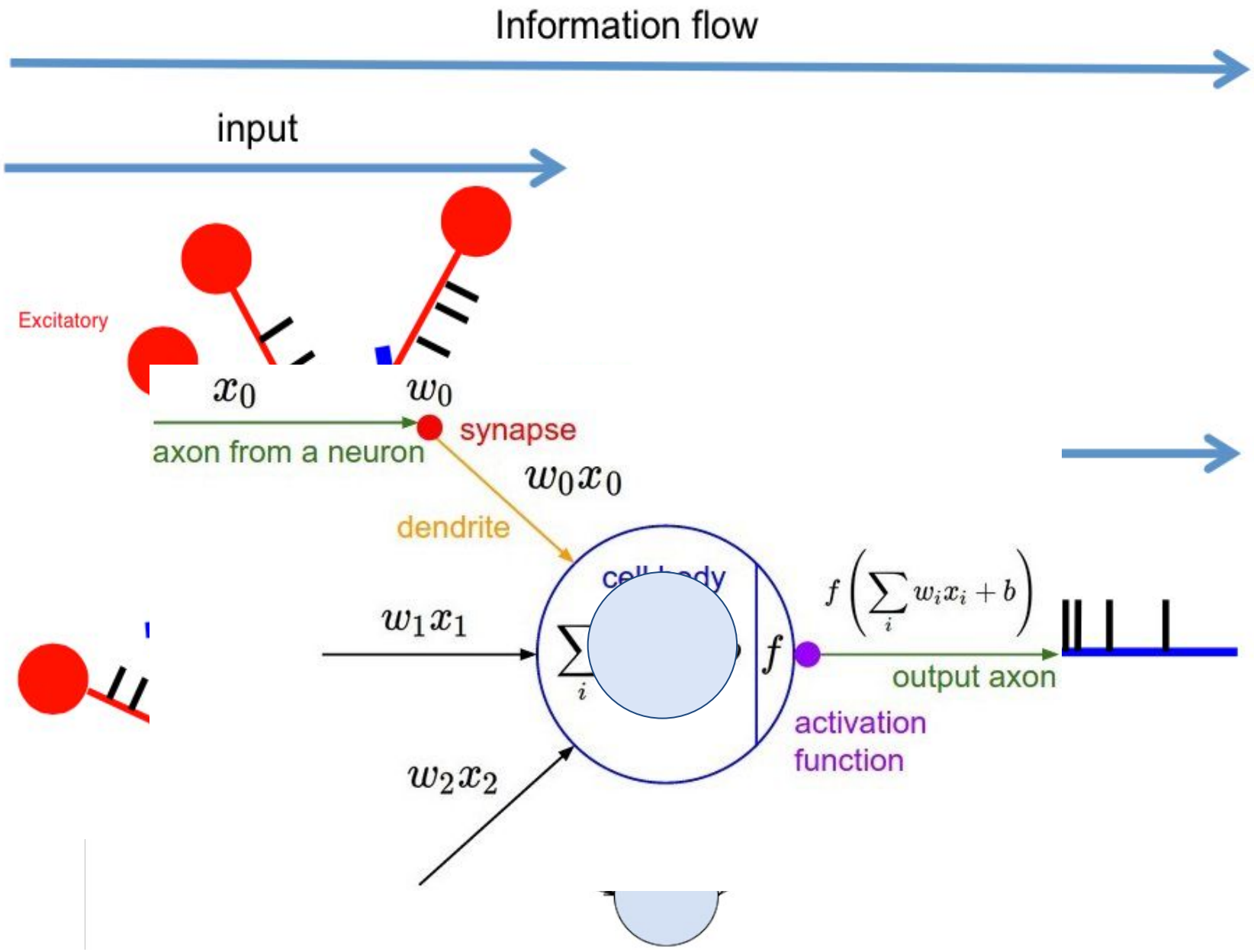




Neurons: the biological model



Extracted from Idan Segev's course: Synapses, neurons and brains.


Information flow
input
Excitatory
x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
$w_1 x_1$
$w_2 x_2$
$\sum_i$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function

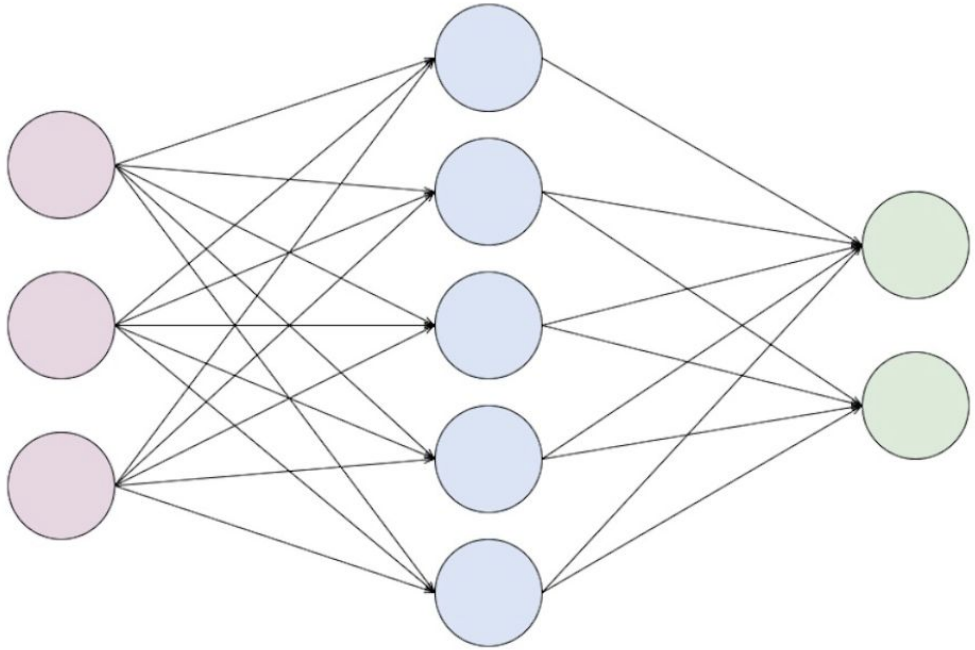
Neurons: the biological model

- How does it works? (simplified version)
  - Each neuron receives inputs from other neurons
  - The effect of each input line on the neuron is controlled by a synaptic weight
    - The weights can be positive (excitatory) or negative (inhibitory).
  - The synaptic weights adapt so that the whole network learns to perform different computations

Neurons: the biological model

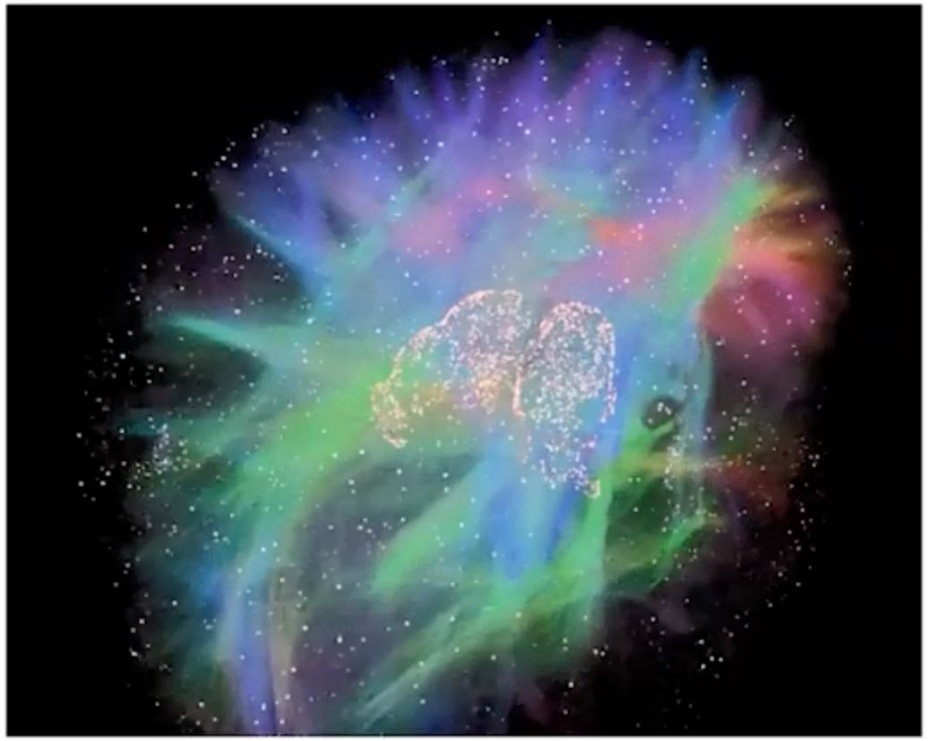
- How does it works (simplified version)?
  - Neural tissue looks pretty much the same, although we know different parts of the brain solve different things
  - However, due to the the synapse weight capacity of adaptation different zones can be re-programmed.
  - Hinton: Cortex is made of general purpose stuff that has the ability to turn into special purpose hardware in response to experience.
    - This gives rapid parallel computation plus flexibility.
  - Human Brain has about 10^{11} neurons each with 10.000 weights

Artificial Neural Networks (ANNs, DNNs)



A net of information

Neurons: the biological model



Human Brain

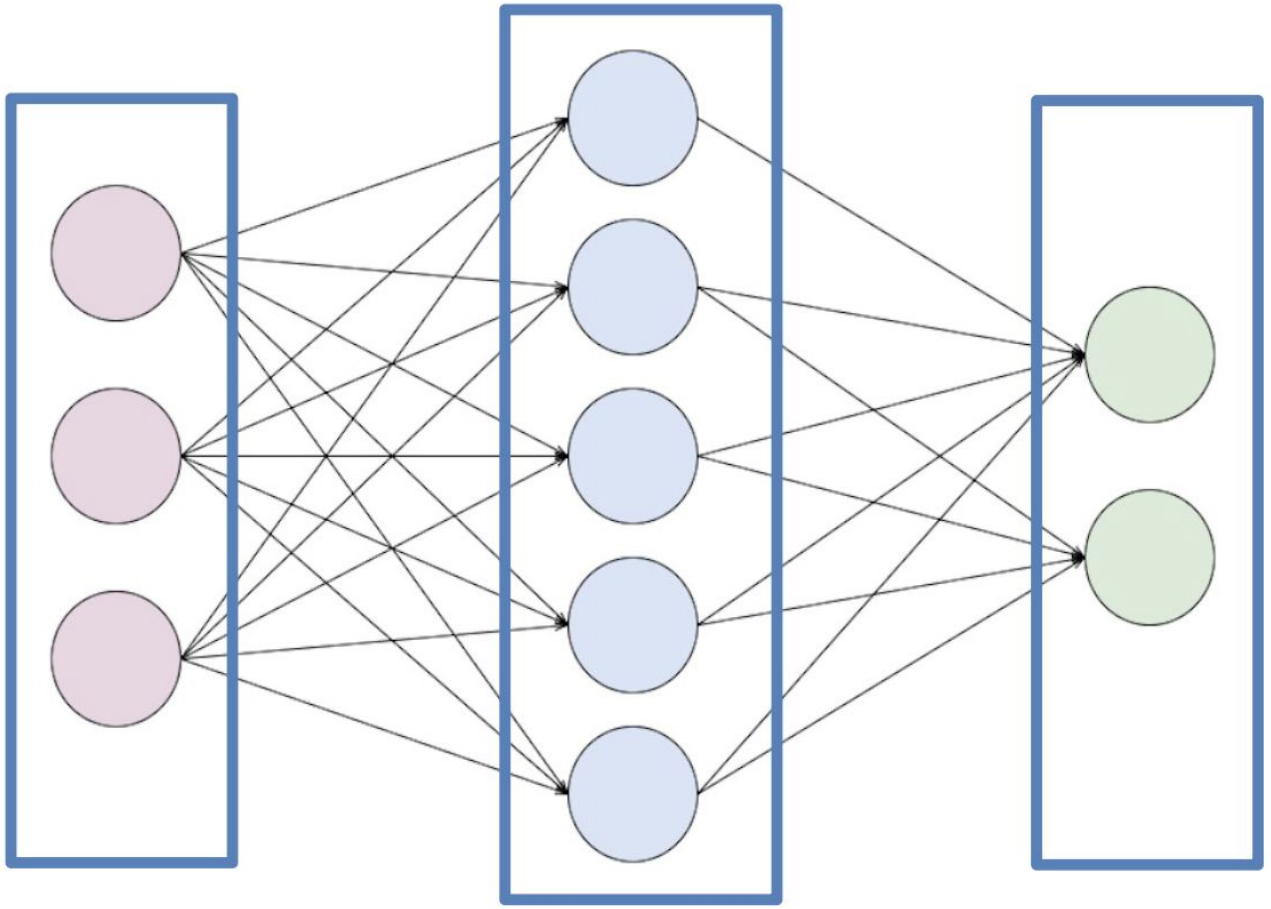
- Thalamocortical system:
  3 million neurons
  476 million synapses
- Full brain:
  100 billion neurons
  1,000 trillion synapses

Artificial Neural Network

- ResNet-152:
  60 million synapses

Human brains have ~10,000,000 times synapses
than artificial neural networks.

ANNs. The Structure



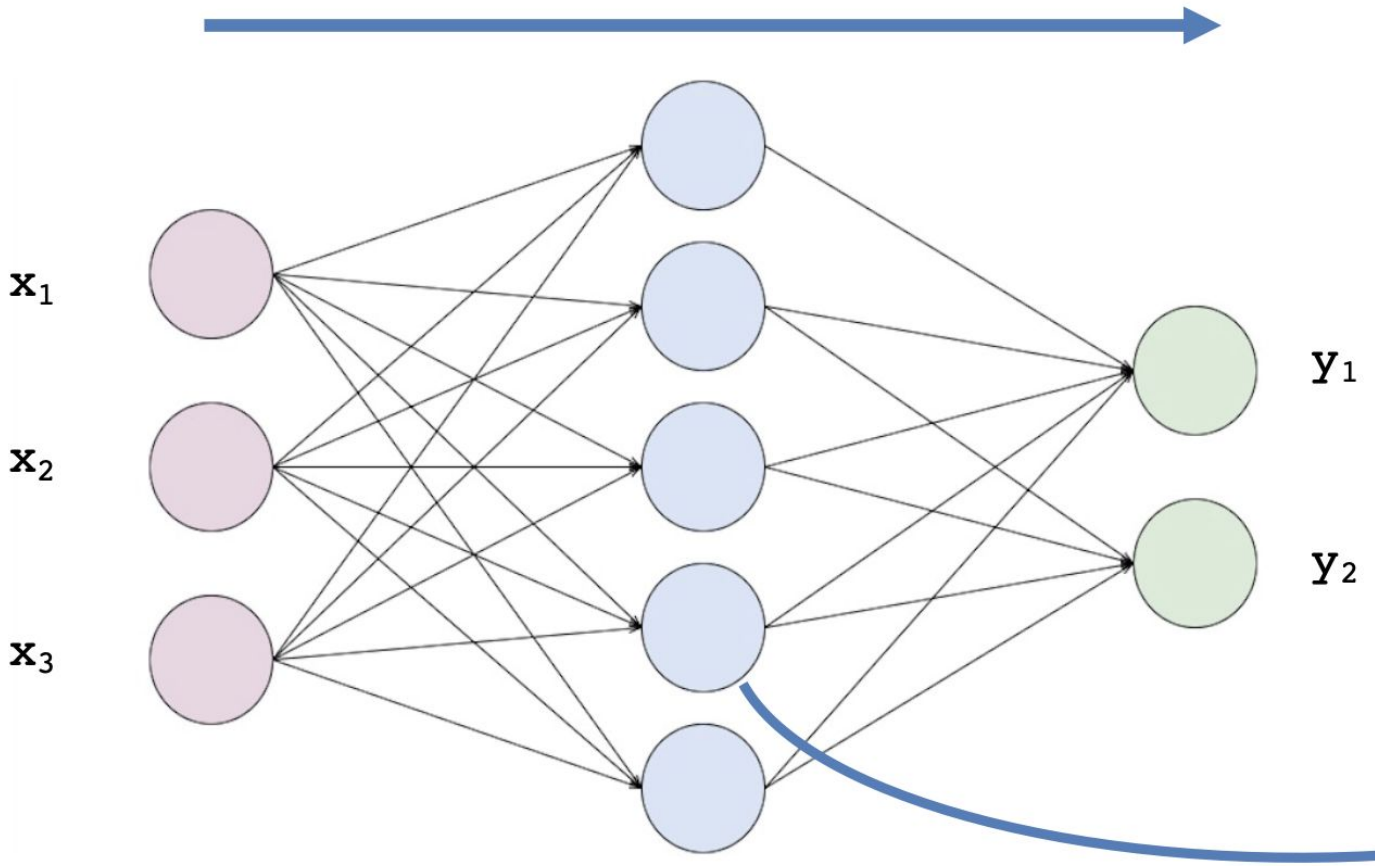
The Input Layer

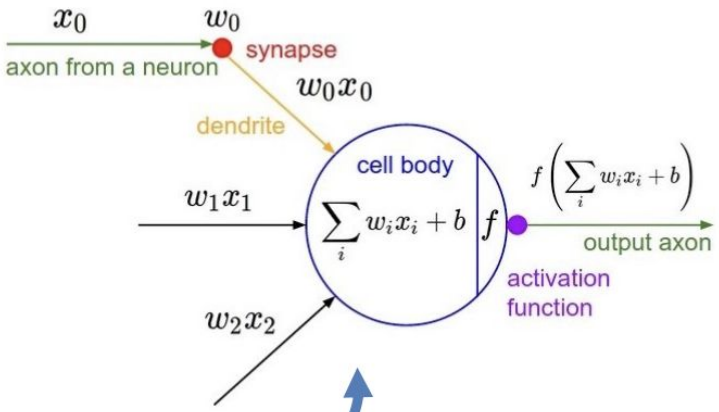
The Hidden Layer

The Output Layer

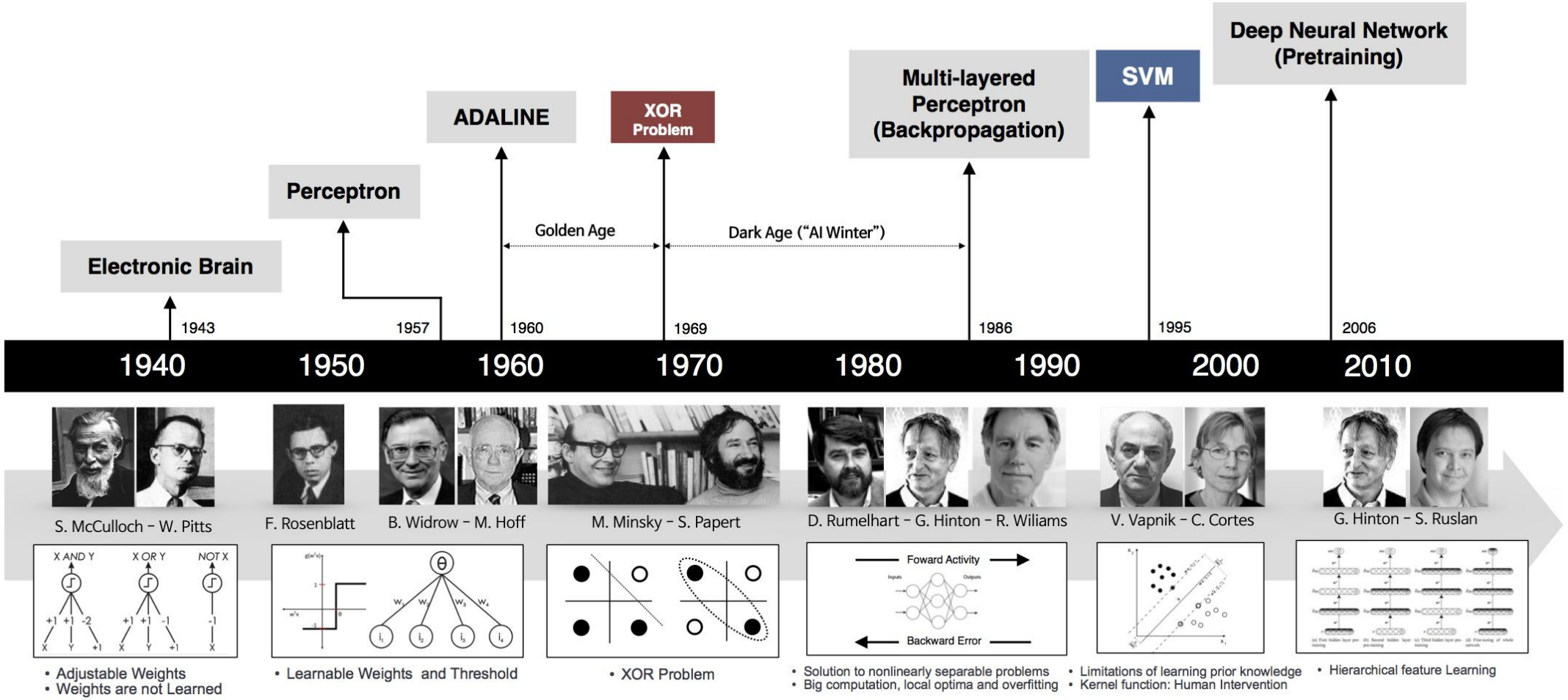
ANNs. Mapping Function

$$Y = F(x)$$



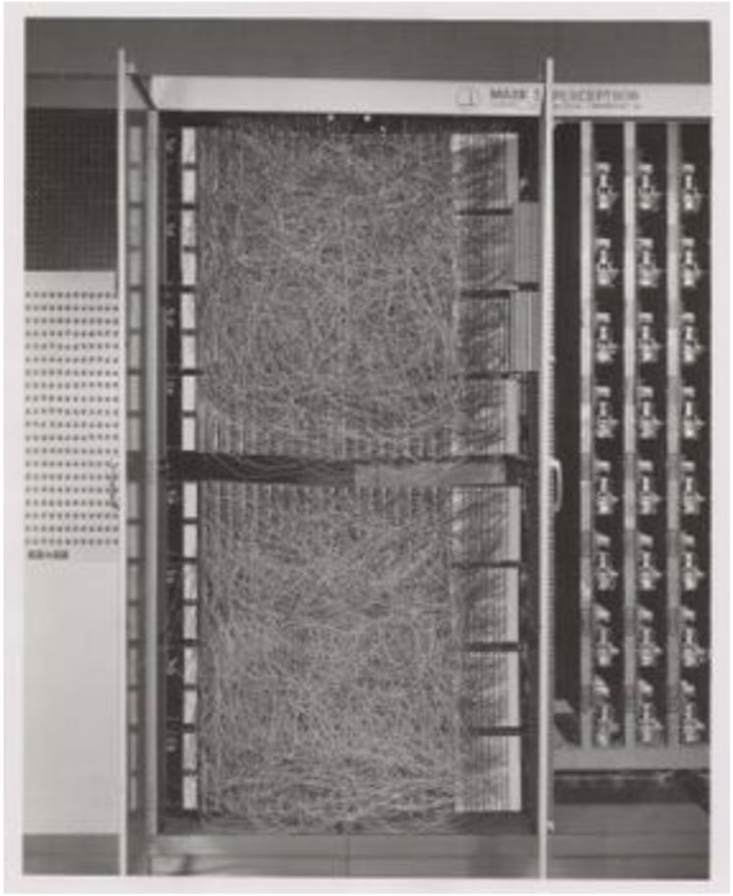


Artificial Neural Networks. Some History



By Andrew. L Beam

The Perceptron



'Mark I Perceptron at the Cornell Aeronautical Laboratory', hardware implementation of the first Perceptron (Source: Wikipedia / Cornell Library)

ANNs. Keras

```
my_first_neural_net = tf.keras.Sequential([
    layers.Input(shape=(3, ), name="input"),
    layers.Dense(units=5, activation="sigmoid", name="hidden"),
    layers.Dense(units=2, activation="softmax", name="output")
])
```

ANNs. Keras

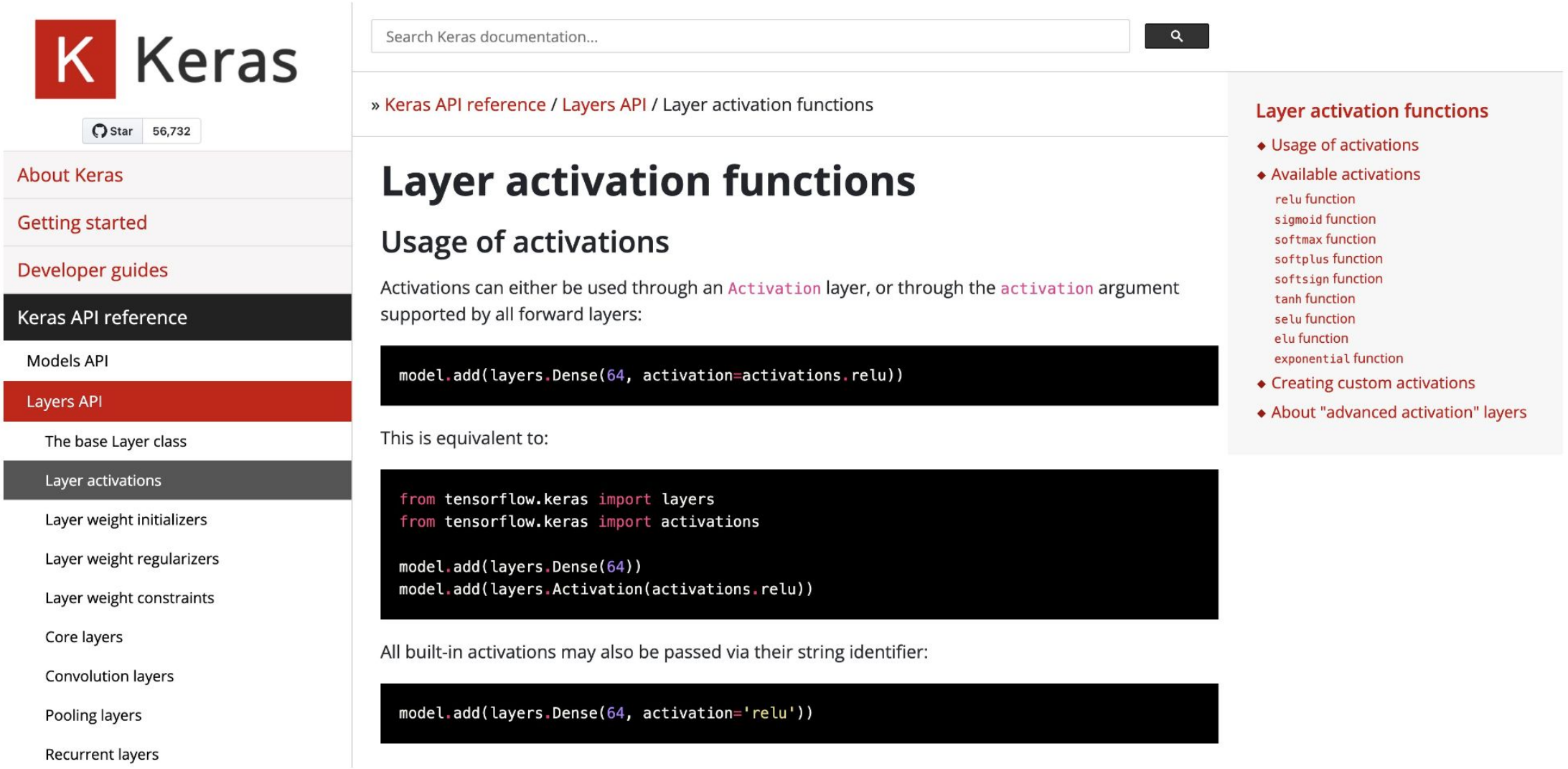
```
my_first_neural_net.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden (Dense)              (None, 5)                 20

 output (Dense)              (None, 2)                 12

=================================================================
Total params: 32
Trainable params: 32
Non-trainable params: 0
_________________________________________________________________
```

ANNs. Activation Functions



Keras

Star 56,732

About Keras
Getting started
Developer guides
Keras API reference
Models API
Layers API
The base Layer class
Layer activations
Layer weight initializers
Layer weight regularizers
Layer weight constraints
Core layers
Convolution layers
Pooling layers
Recurrent layers

Search Keras documentation...

» Keras API reference / Layers API / Layer activation functions

Layer activation functions

Usage of activations

Activations can either be used through an `Activation` layer, or through the `activation` argument supported by all forward layers:

```
model.add(layers.Dense(64, activation=activations.relu))
```

This is equivalent to:

```
from tensorflow.keras import layers
from tensorflow.keras import activations

model.add(layers.Dense(64))
model.add(layers.Activation(activations.relu))
```

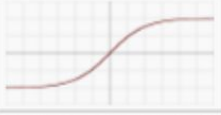
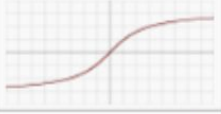
All built-in activations may also be passed via their string identifier:

```
model.add(layers.Dense(64, activation='relu'))
```

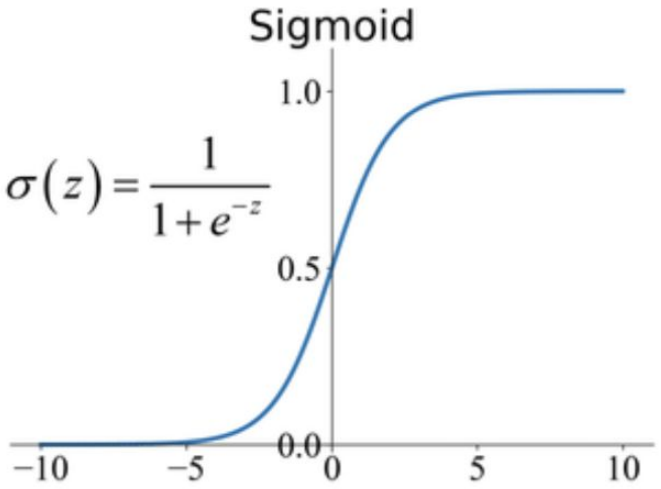
Layer activation functions

- Usage of activations
- Available activations
  - `relu` function
  - `sigmoid` function
  - `softmax` function
  - `softplus` function
  - `softsign` function
  - `tanh` function
  - `selu` function
  - `elu` function
  - `exponential` function
- Creating custom activations
- About "advanced activation" layers

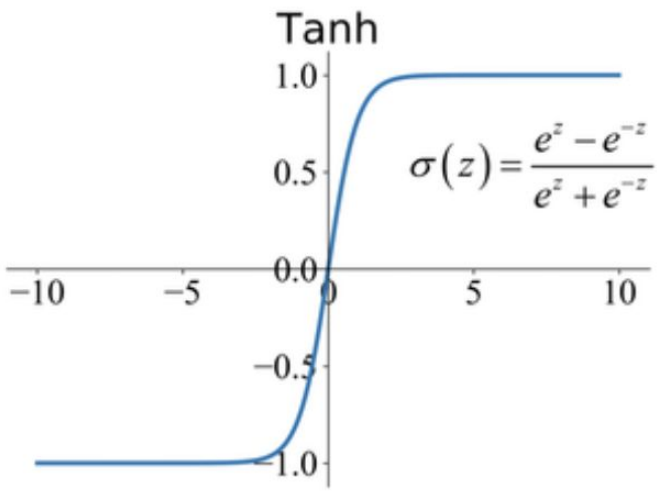
ANNs. Activation Functions

Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$

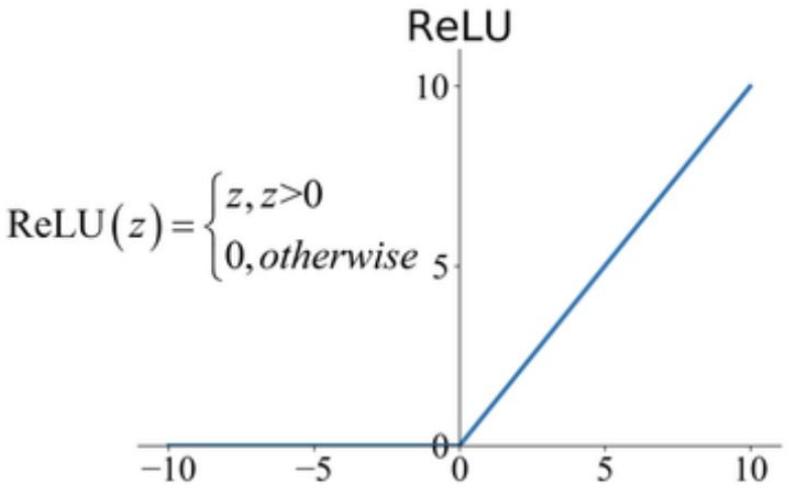
ANNs. Activation Functions



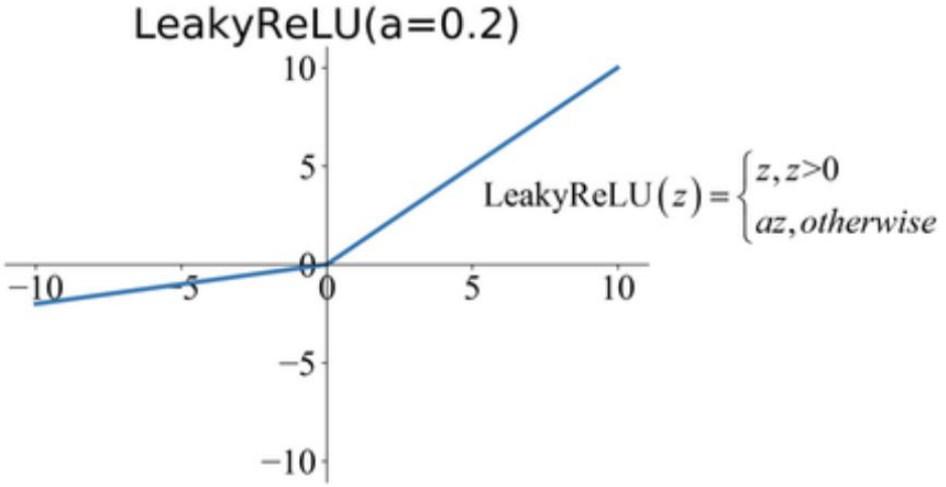
(a)



(b)



(c)



(d)

ANNs. Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

ANNs. Activation Functions

- The combination of linear functions is just another linear function. No matter the number of hidden layers, if we use linear functions as activations, we could just deal with outputs that are a linear combination of our inputs... not interesting.

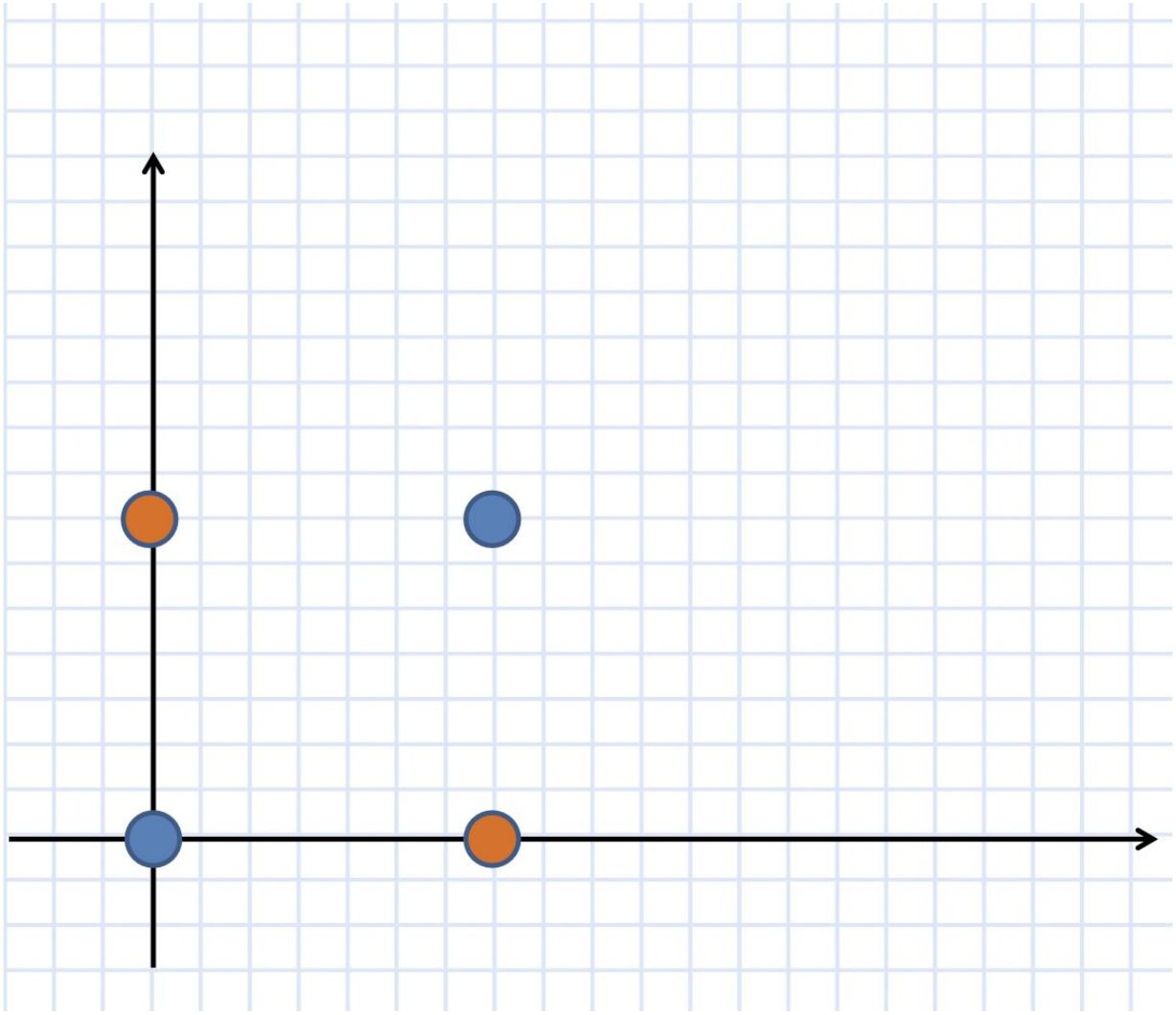
- Non-linear activations functions are key in neural nets to represent non-linear mapping functions.

- **In the output layer**, at general: use sigmoid for binary classification, softmax for multiclass classification and linear for regression.

    - You can use softmax for binary classification using two outputs and in the case of a problem where data that can belong to multiple classes, then use sigmoid. [i.e., a picture of Audi belongs to the classes' car and vehicle]

- **In hidden layers:**
  - Tanh use to work better than sigmoid as it centers data.
  - ReLu is faster and it works very well too. It is the most used.

The XOR (exclusive OR) problem

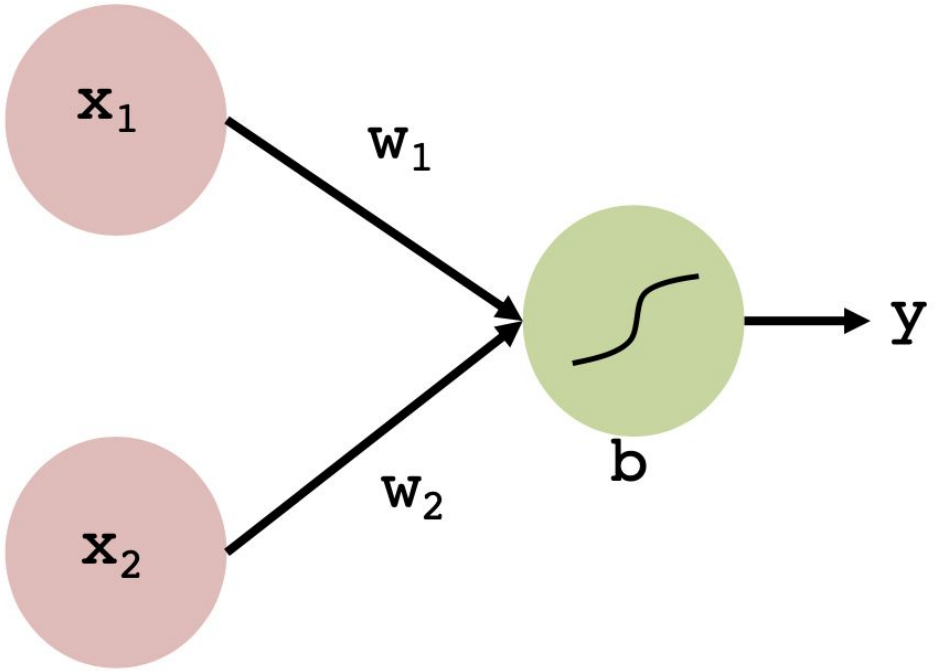


x_1	x_2	$y = x_1 \text{ xor } x_2$
0	0	0
1	0	1
0	1	1
1	1	0

The XOR (exclusive OR) problem

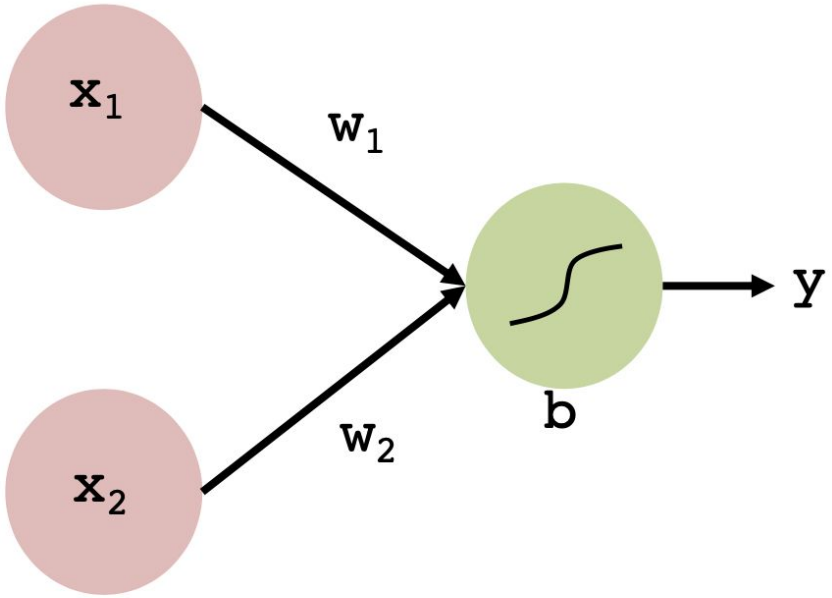
- XOR is a toy problem but very didactic.
- XOR represents a non-linear problem where a linear classifier will fail in discerning classes 0 and 1 – there is not a single boundary able to separate the classes
- **If we use a single hidden unit in the hidden layer, we could not solve the problem**
- **... but why if we use more hidden units using non-linear activations**

The XOR (exclusive OR) problem

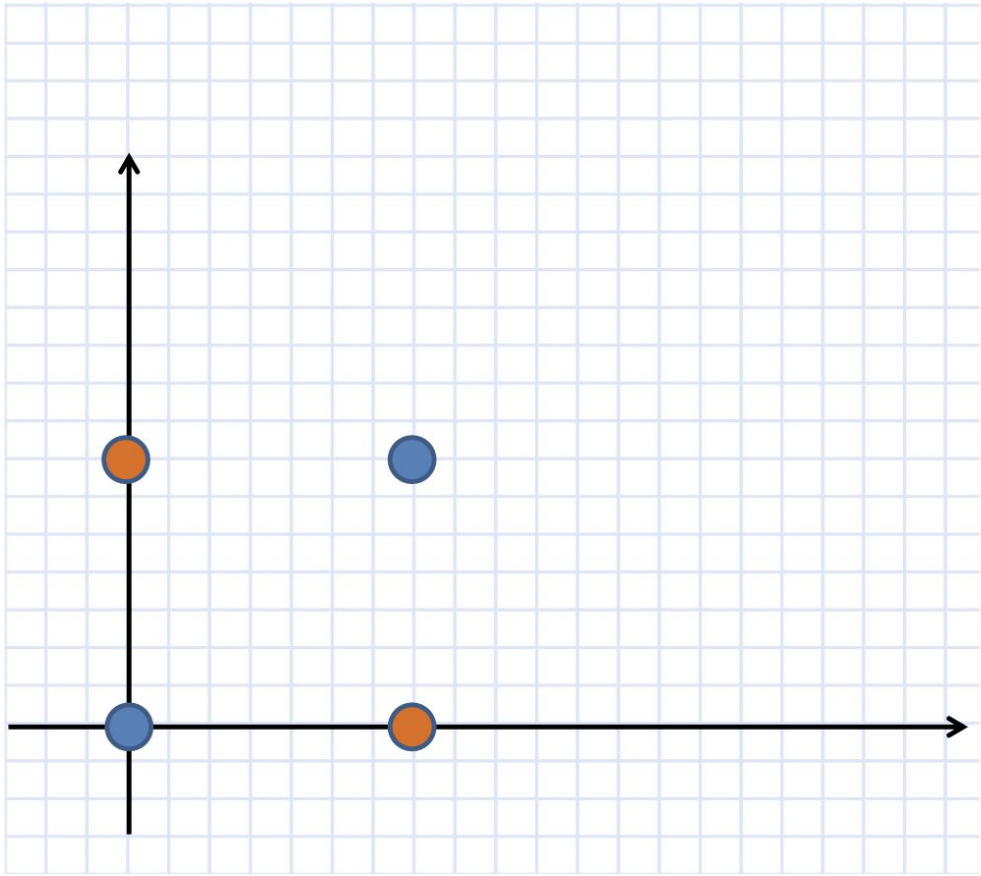


$$y = s(w_1x_1 + w_2x_2 + b)$$

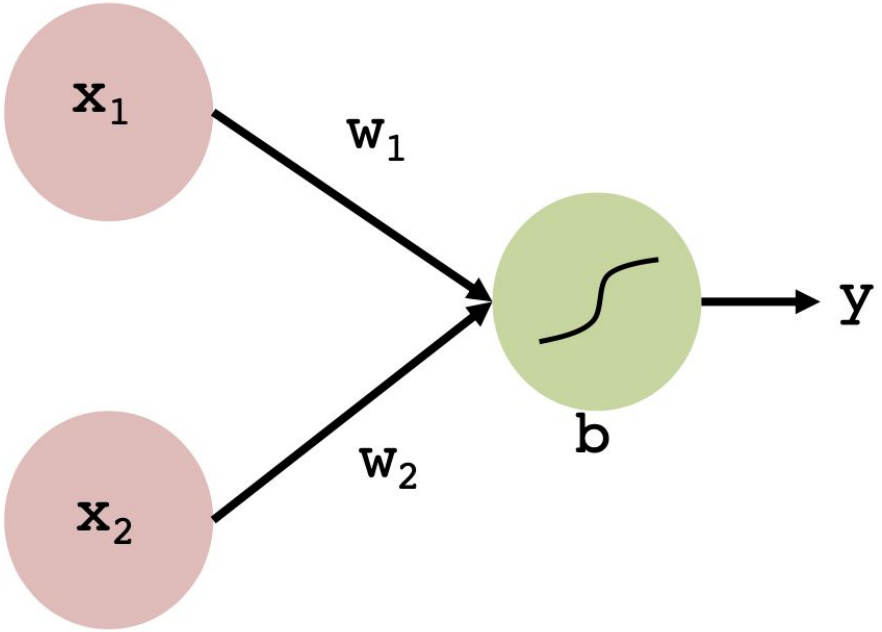
The XOR (exclusive OR) problem



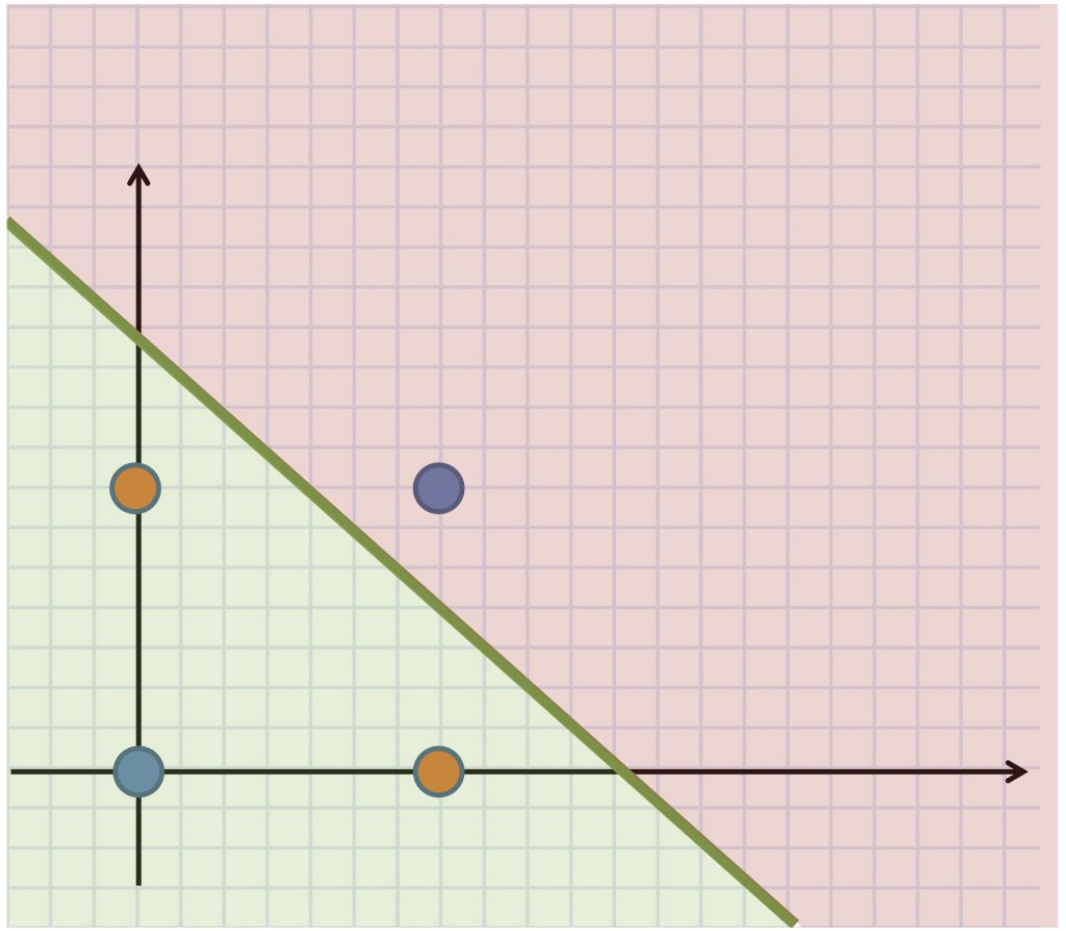
$$y = s(w_1x_1 + w_2x_2 + b)$$



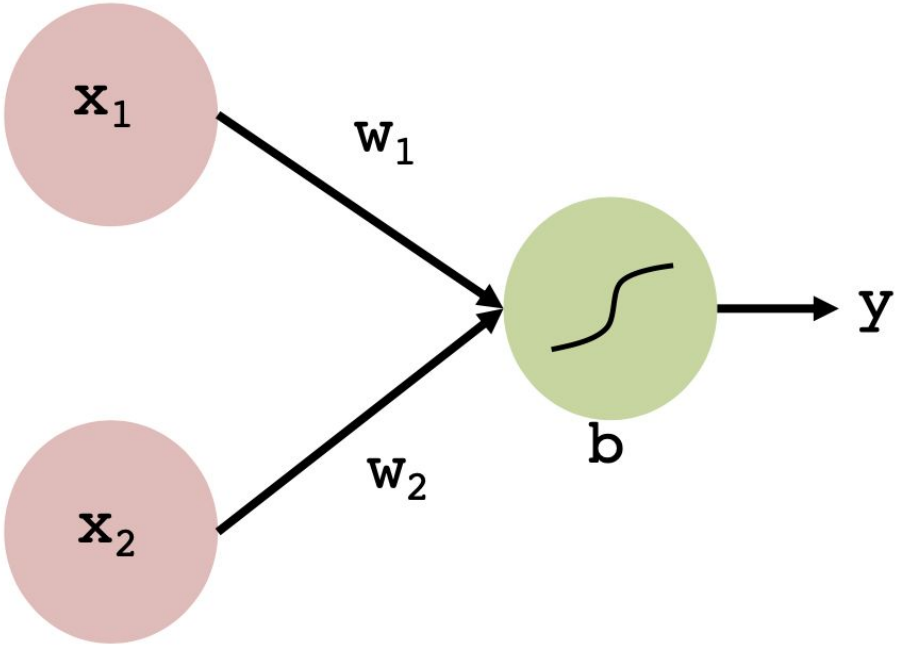
The XOR (exclusive OR) problem



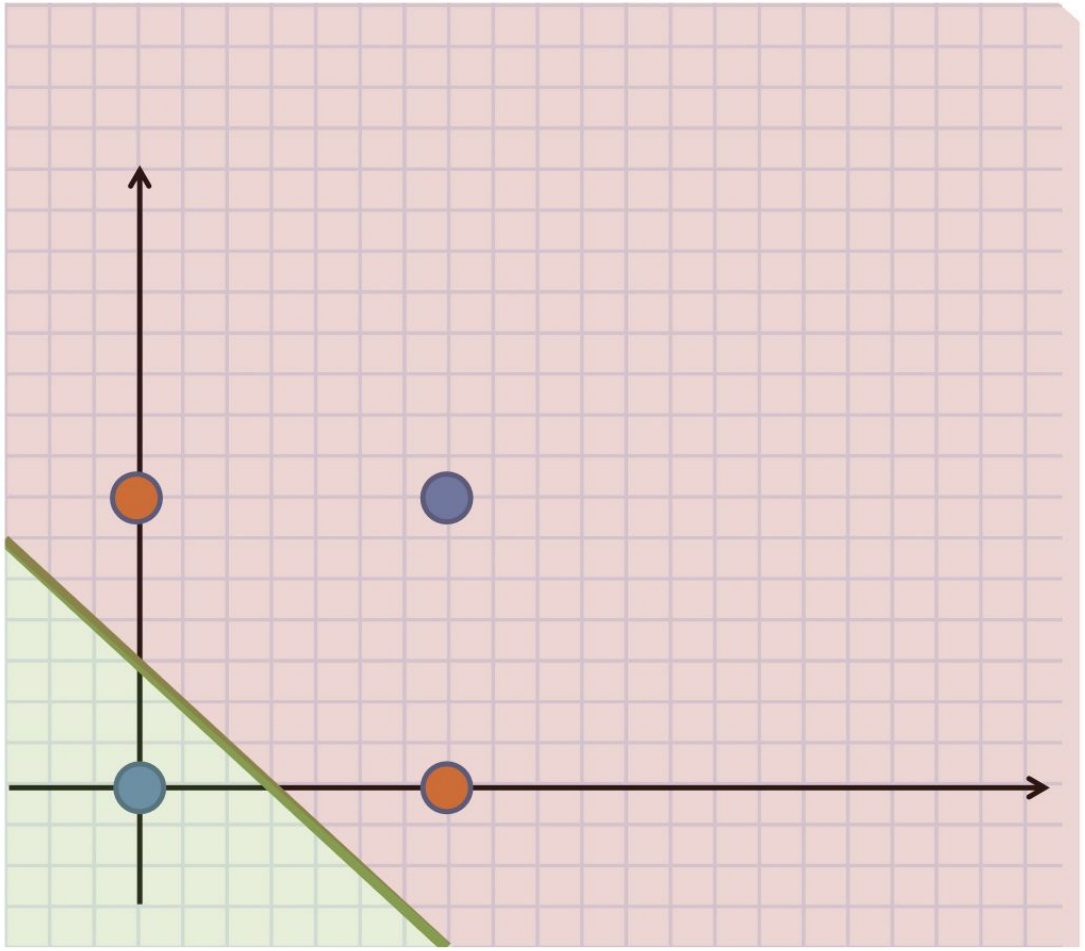
$$y = s(w_1x_1 + w_2x_2 + b)$$



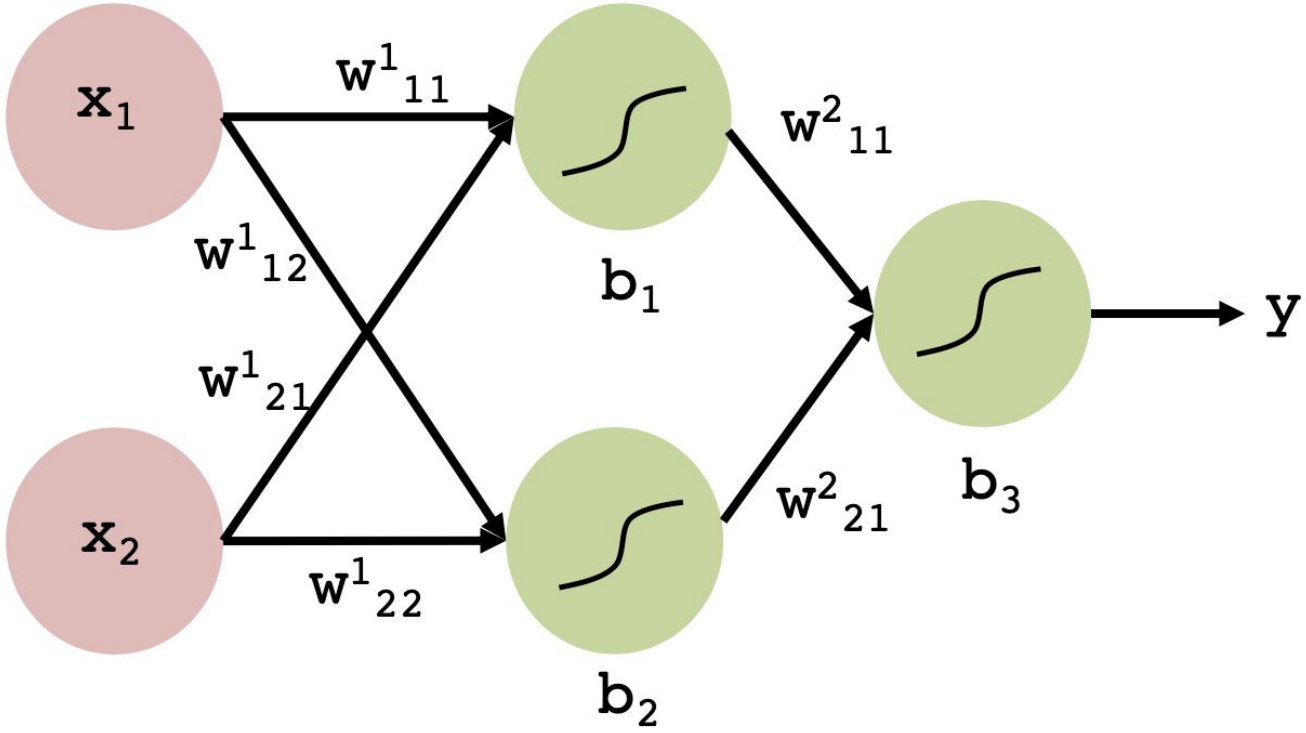
The XOR (exclusive OR) problem



$$y = s(w_1x_1 + w_2x_2 + b)$$

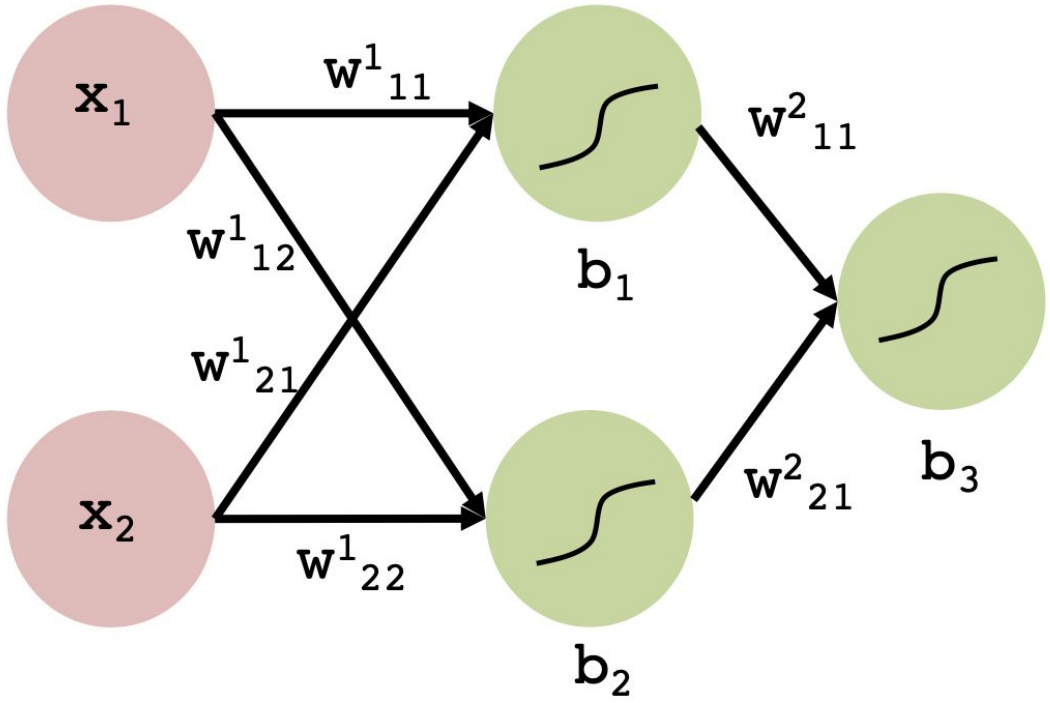


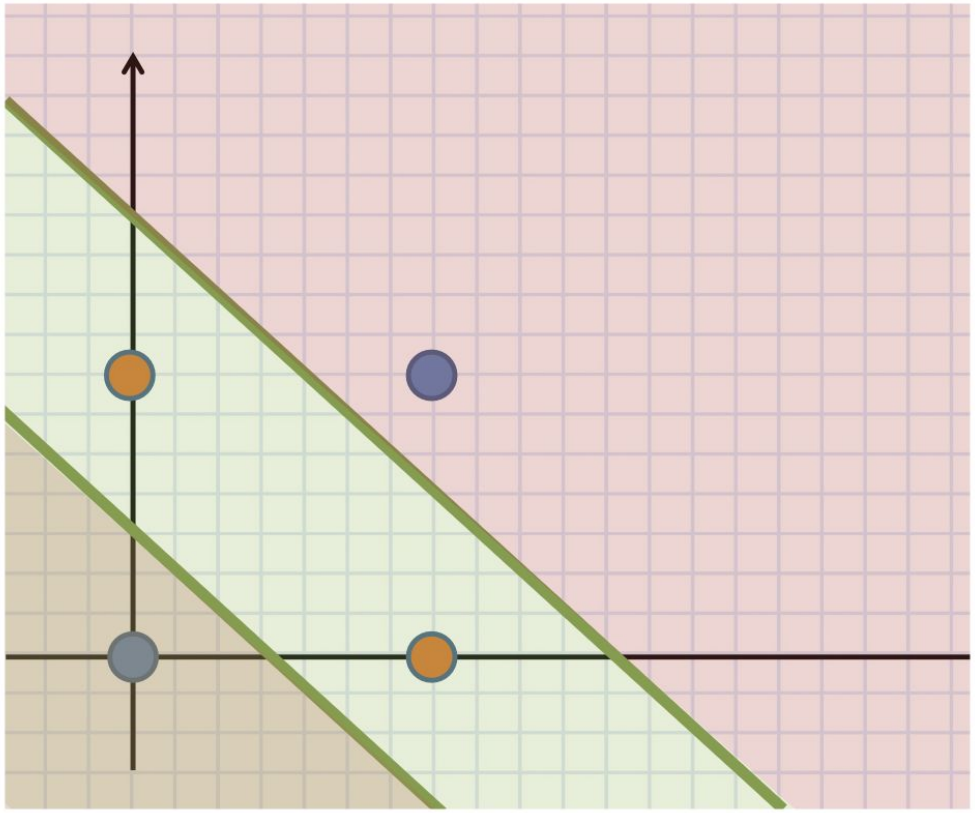
The XOR (exclusive OR) problem



$$y = s(s(w^1_{11}x_1 + w^1_{21}x_2 + b_1) + s(w^1_{21}x_1 + w^1_{22}x_2 + b_2) + b_3)$$

The XOR (exclusive OR) problem





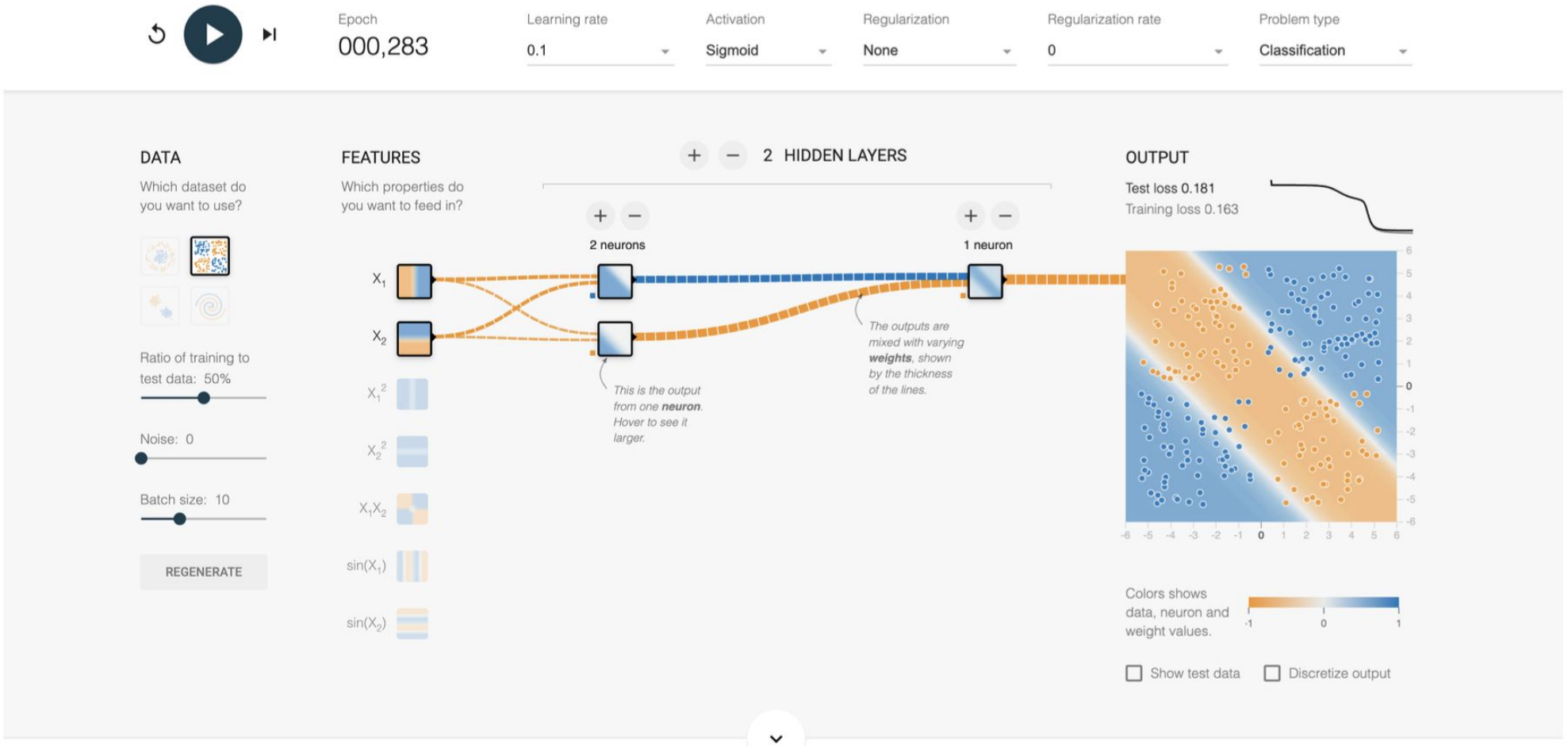
$$y = s(s(w^1_{11}x_1 + w^1_{21}x_2 + b_1) + s(w^1_{21}x_1 + w^1_{22}x_2 + b_2) + b_3)$$

The XOR (exclusive OR) problem

- By using a hidden layer with 2 non-linear units we can create two different boundaries.
- If we increase the number of hidden units, we can get more complicated boundaries... watch out with the overfitting effect!
- The cost function is not longer convex so there is a possibility to reach a “local or asymptotic minima”.

The XOR (exclusive OR) problem

http://playground.tensorflow.org/



Main Concepts

- Artificial neural networks try to mimic the brain
  - The first advantage we see is the parallel use of the information. All neurons connected.

- ANNs. The neuron

- ANNs. The structure

- ANNs. The mapping function
  - Simply by modifying the weights we will be changing the behavior of the net (as in any other machine learning algorithm)

- The XOR Problem