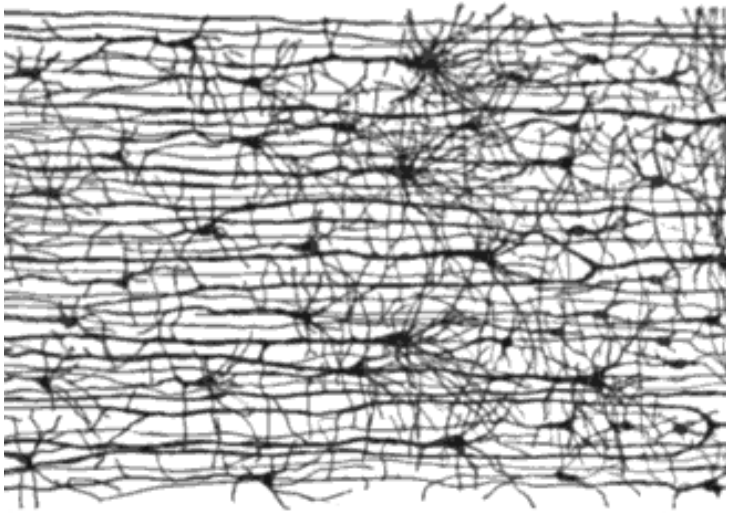
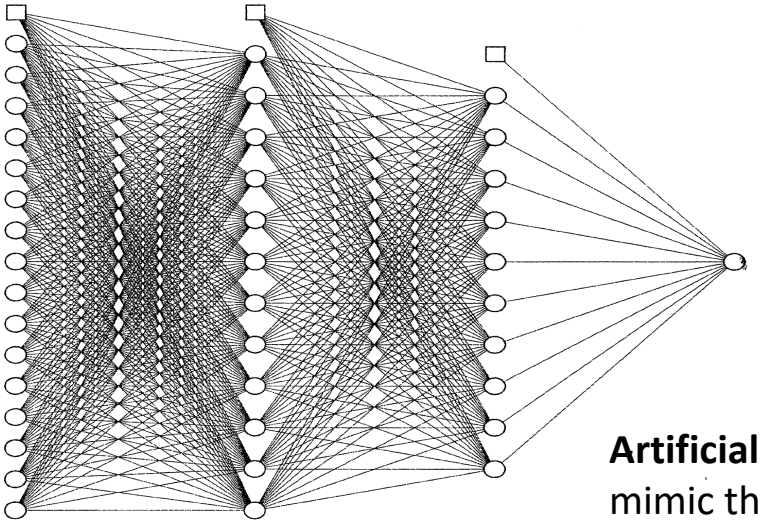
Section 10 Neural Network

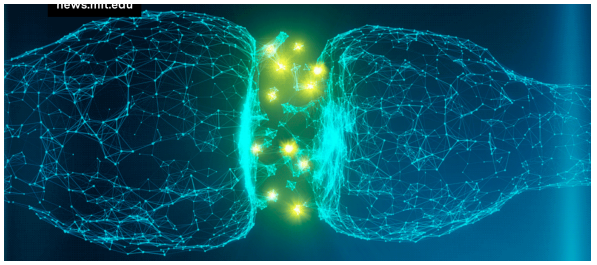
- Perceptron
- Neural Network
- Backpropagation



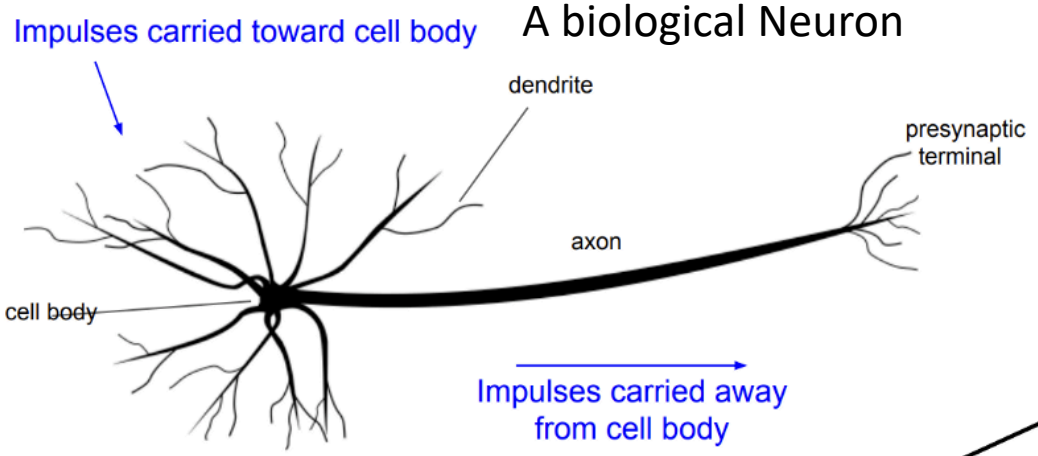


Human Neural Networks was introduced in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts to model neurons in the brain using electrical circuits.

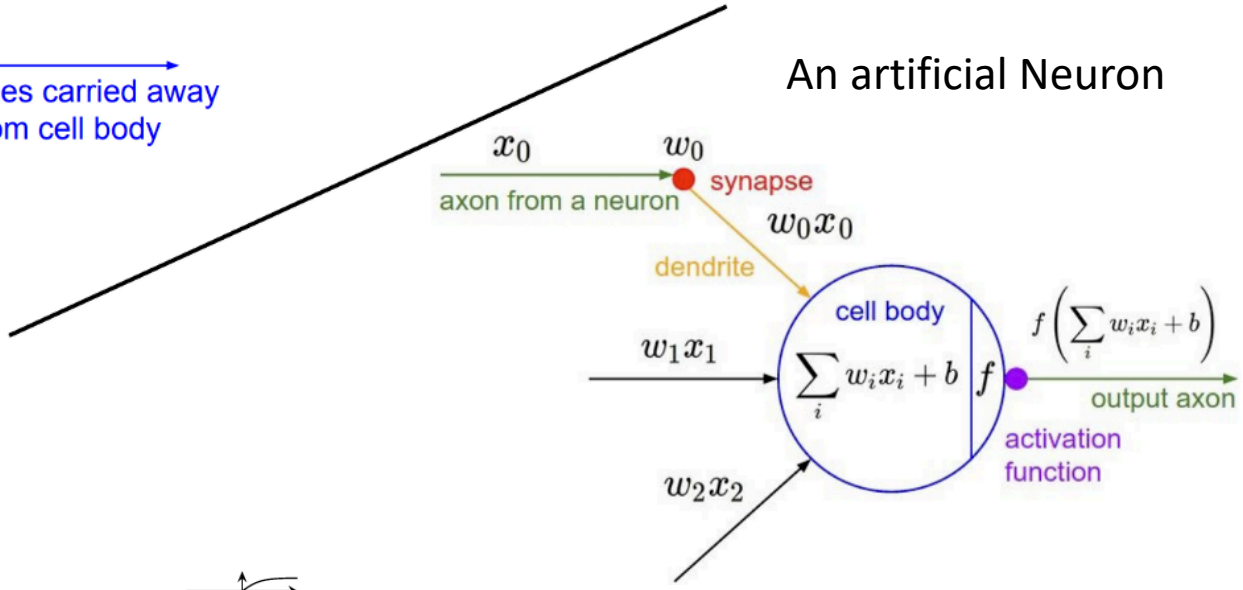
Artificial Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data. It's a very broad term that encompasses any form of Deep Learning model.

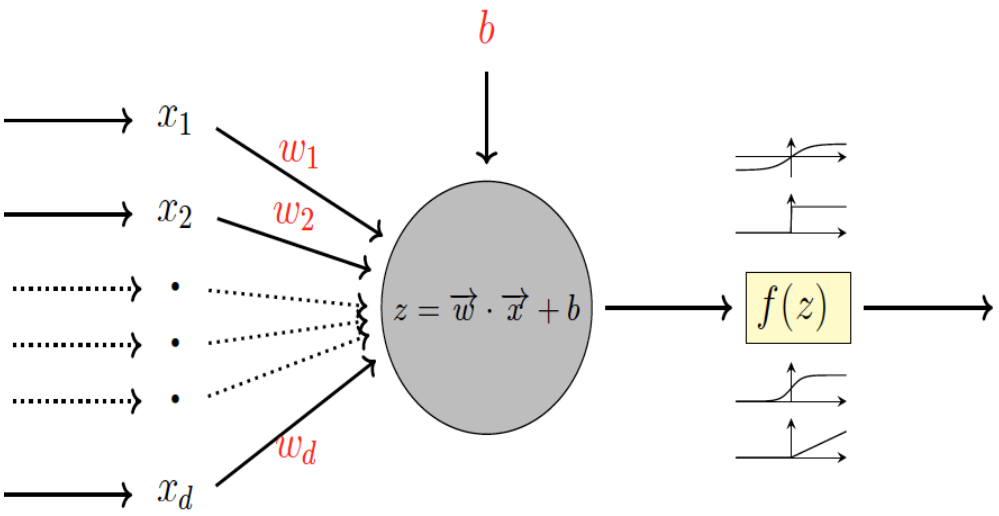


A biological Neuron



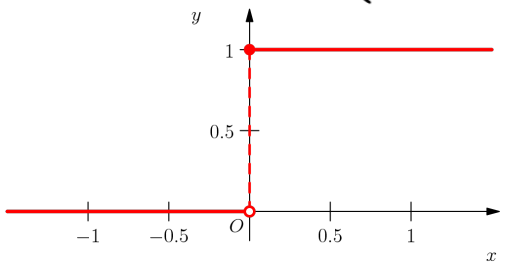
An artificial Neuron



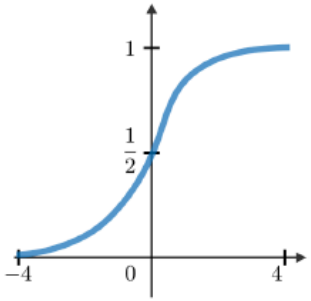
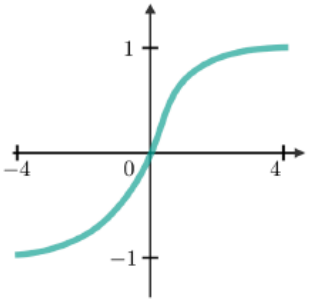
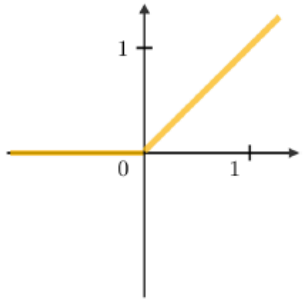
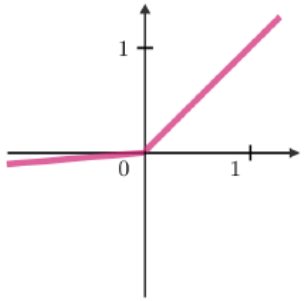


Activation function example: Heaviside step function

$$f(x) = \text{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

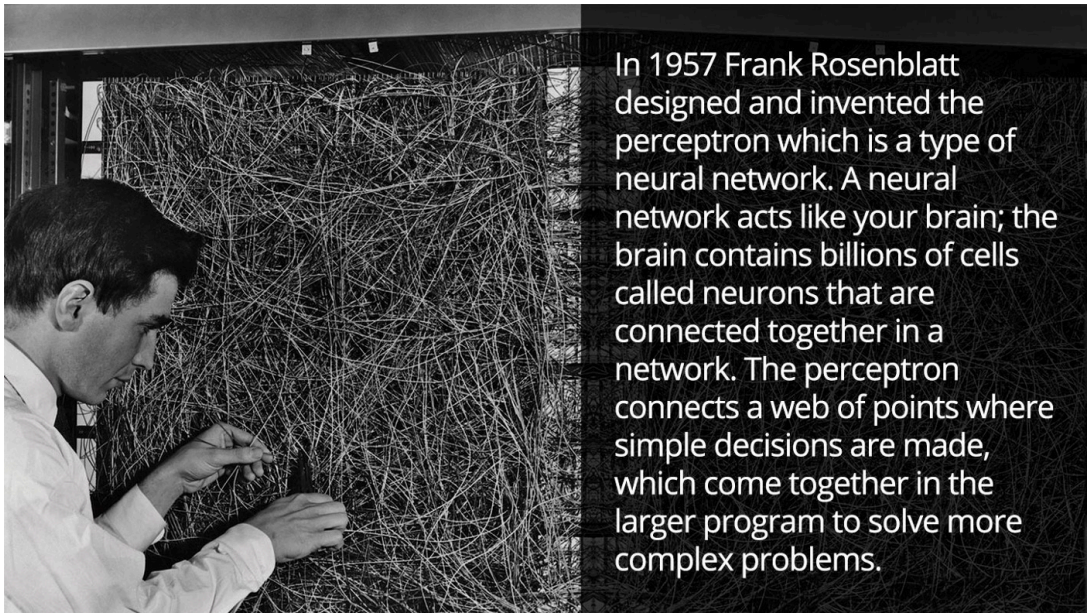


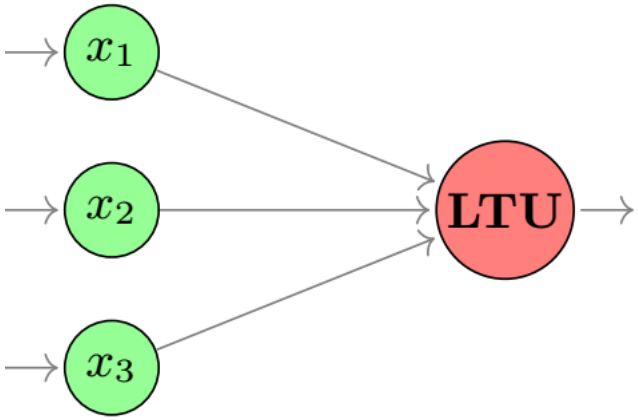
- Other activation functions:

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Perceptron

The perceptron is based around a *linear threshold unit* (LTU).





The New Yorker, December 6, 1958 P. 44

Talk story about the perceptron, a new electronic brain which hasn't been built, but which has been successfully simulated on the I.B.M. 704. Talk with Dr. Frank Rosenblatt, of the Cornell Aeronautical Laboratory, who is one of the two men who developed the prodigy; the other man is Dr. Marshall C. Yovits, of the Office of Naval Research, in Washington. Dr. Rosenblatt defined the perceptron as the first non-biological object which will achieve an organization o its external environment in a meaningful way. It interacts with its environment, forming concepts that have not been made ready for it by a human agent. If a triangle is held up, the perceptron's eye picks up the image & conveys it along a random succession of lines to the response units, where the image is registered. It can tell the difference betw. a cat and a dog, although it wouldn't be able to tell whether the dog was to theleft or right of the cat. Right now it is of no practical use, Dr. Rosenblatt conceded, but he said that one day it might be useful to send one into outer space to take in impressions for us.

https://www.newyorker.com/magazine/1958/12/06/rival-2

https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

➢ Perceptron

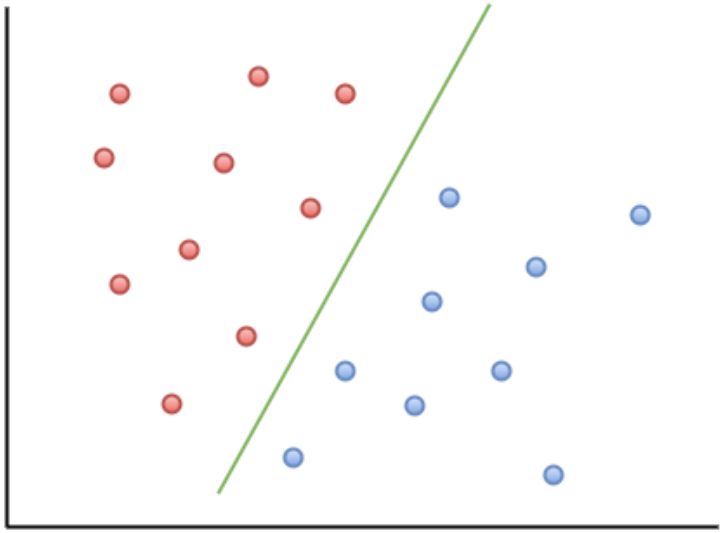
Training Data: $\mathcal{D} = \left(\vec{x}^{(i)}, y^{(i)}\right)$ for $i = 1 \ldots n$.

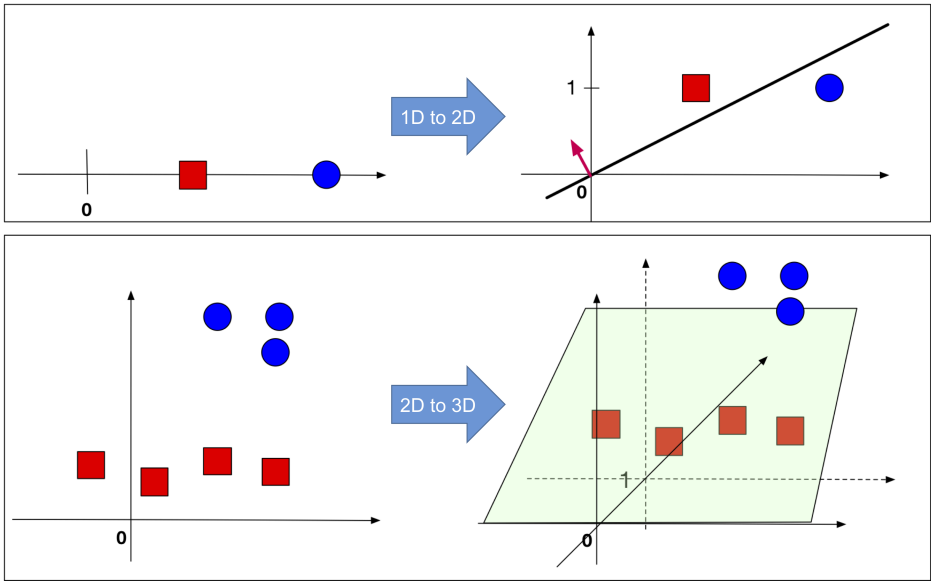
Assumptions:

- *Binary classification* (i.e. $y^{(i)} \in \{-1, +1\}$)
- Data is *linearly separable*, i.e., there exists a hyperplane that separates all the sample points in class A from classes B.

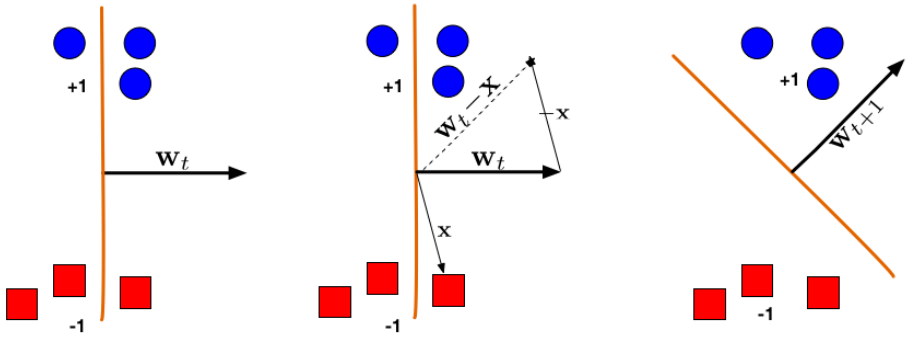
Classifier:

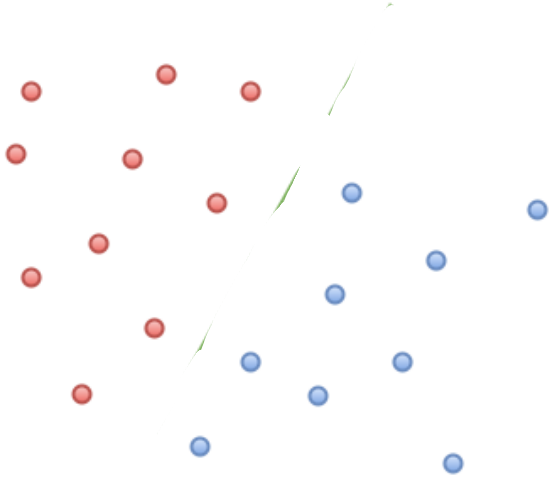
$$h(\vec{x}) = \text{sign}\left(\vec{\theta}^T \vec{x}\right) = \text{sign}(\vec{w} \cdot \vec{x} + b)$$





$y^{(i)}(\vec{\theta}^T\ \vec{x}^{(i)}) > 0 \iff \vec{x}^{(i)}$ classified correctly


+1
-1
$\mathbf{w}_t$
$\mathbf{w}_t - \mathbf{x}$
x
$\mathbf{w}_{t+1}$



- **Training the Perceptron** $\mathcal{D} = \left(\vec{x}^{(i)}, y^{(i)}\right)$ for $i = 1 \ldots n$.

Start with initial $\vec{\theta} = \vec{0}$

For $i = 1, \ldots, n$

Repeat $\vec{\theta}^{next} := \vec{\theta} + \alpha \left(y^{(i)} - h_\theta\left(\vec{x}^{(i)}\right)\right) \vec{x}^{(i)}$

The perceptron updates its weights only on misclassified points.

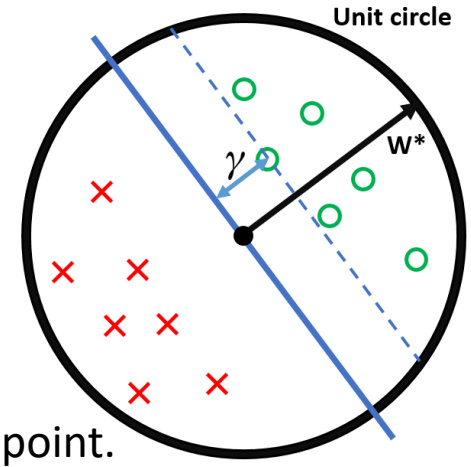
```python
def perceptron_sgd(X, Y):
    w = np.zeros(len(X[0]))  #Initialize the weight vector for the perceptron with zeros
    eta = 1                  #Set the learning rate to 1
    epochs = 20              #Set the number of epochs

    for t in range(epochs):
        for i, x in enumerate(X):
            if (np.dot(X[i], w)*Y[i]) <= 0:
                w = w + eta*X[i]*Y[i]

    return w

w = perceptron_sgd(X,y)
print(w)
```

The perceptron is a form of stochastic gradient decent on the loss function

$$J(\vec{\theta}) = -\sum_{i=1}^{n} \left(y^{(i)} - h_\theta(\vec{x}^{(i)})\right)(\vec{\theta}^T \vec{x}^{(i)})$$

We consider the **online learning setting** for the perceptron. The algorithm has to make predictions continuously even while it's learning. Specifically, the algorithm first sees $\vec{x}^{(1)}$ and is asked to predict what it thinks $y^{(1)}$ is. After making its prediction, the true value of $y^{(1)}$ is revealed to the algorithm and the algorithm may use this information to perform some learning. The algorithm then see $\vec{x}^{(2)}$ and keep going.

In the online learning setting, we are interested in the total number of errors made by the algorithm during this process. Thus, it models applications in which the algorithm has to make predictions even while it's still learning.



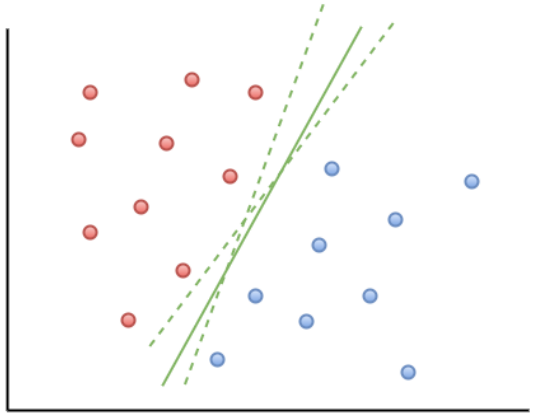
Theorem (Block, 1962, and Novikoff, 1962).

Suppose inputs are scaled to live within the **unit** sphere.

A separating hyperplane is defined by **unit** vector $\vec{\theta}$

$\gamma = \min_{\mathcal{D}} |\vec{\theta}^T \vec{x}^{(i)}|$ is the distance from hyperplane to the closed point.

Then, the Perceptron algorithm makes at most $\frac{1}{\gamma^2}$ mistakes.

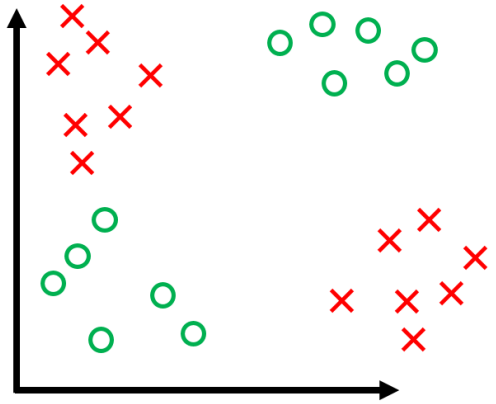


- When the data is separable, there are many solutions and which one is found depends on the starting value.
- The finite number of steps can be large, practically, if the gap is small the time to find it is large.
- When the data are not separable, the algorithm does not converge, and instead falls into a cycle.

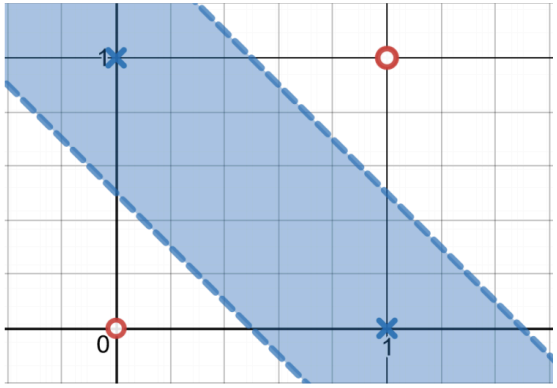
Video illustration for perceptron:

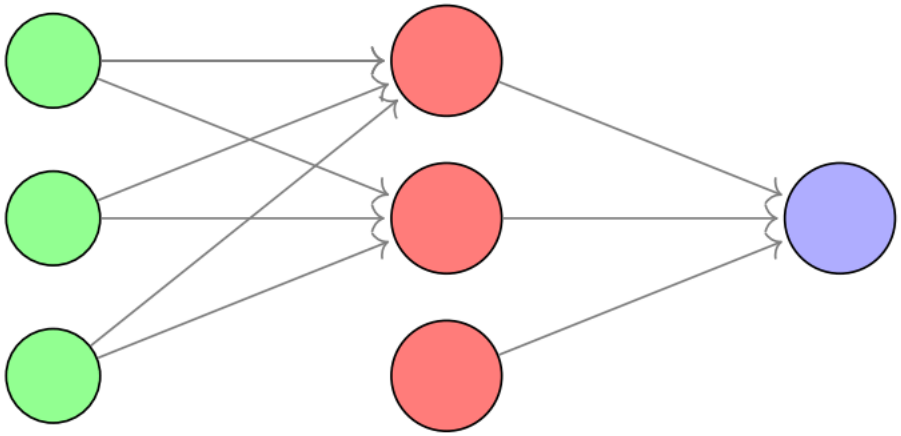
https://www.youtube.com/watch?v=xpJHhHwR4DQ

Famous example of a simple non-linearly separable data set, the XOR problem (Minsky 1969):



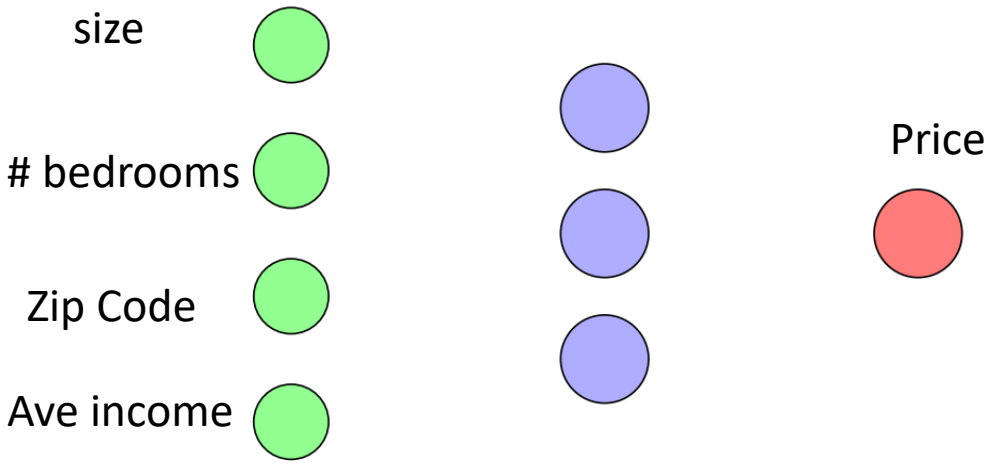
Although now unsurprising (no linear classier can solve xor) the exceptions for the perceptron were high and when this problem was uncovered in 1969, it leads most researchers to abandon neural networks in favor of functional and logical methods.

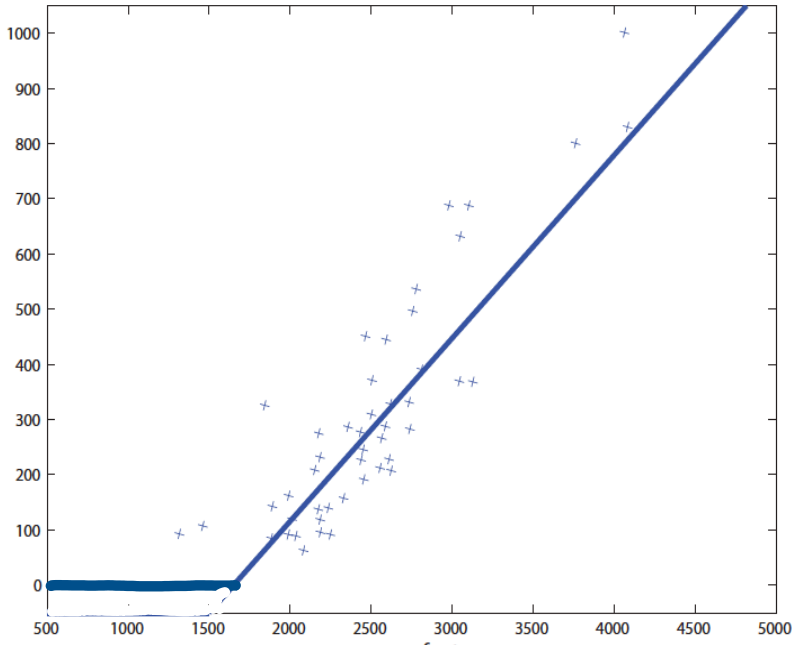




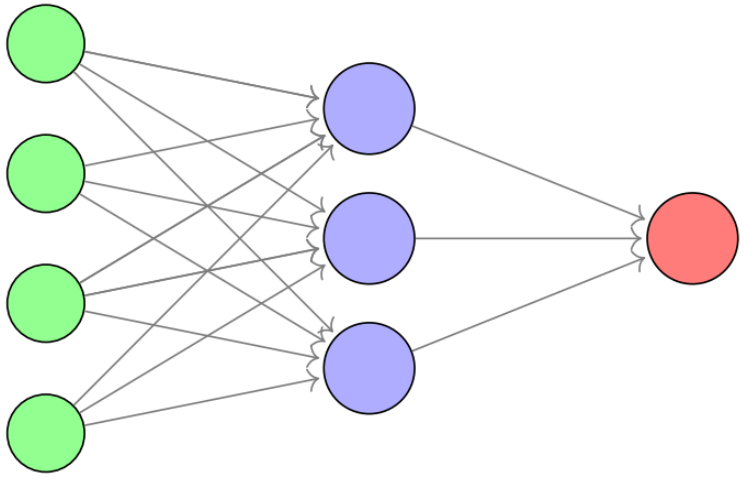
A visual proof that neural nets can compute any function: http://neuralnetworksanddeeplearning.com/chap4.html

- House price example

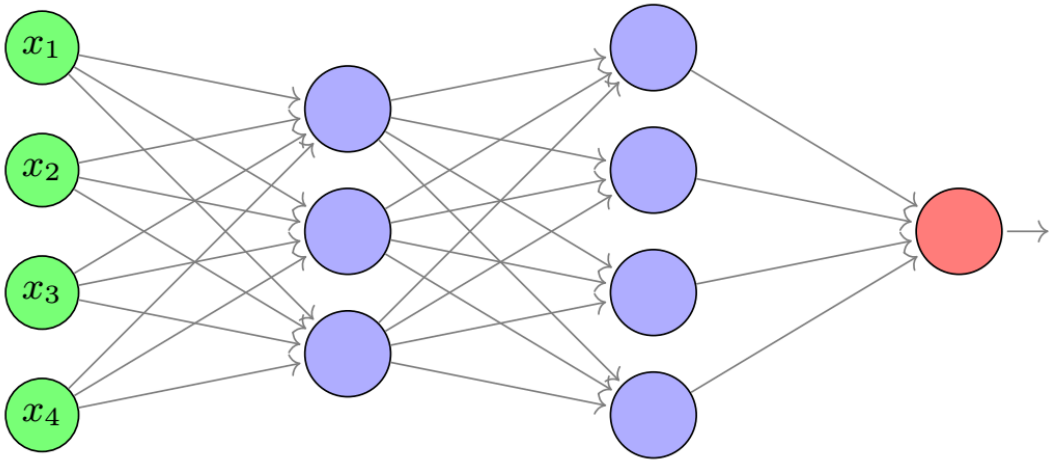


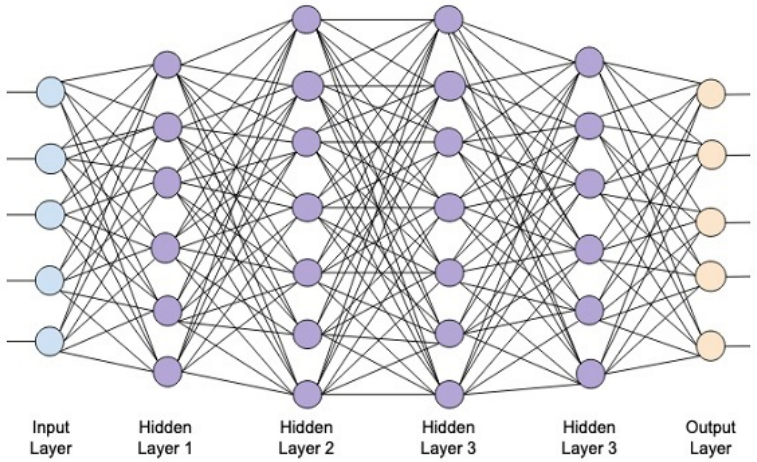


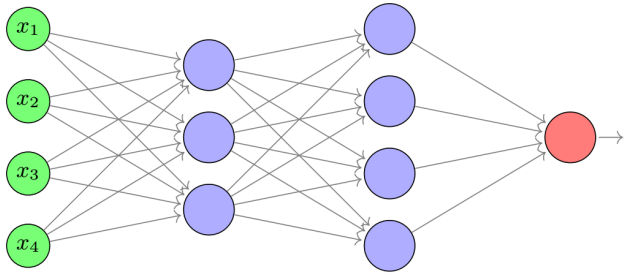
- Fully-connected neural networks

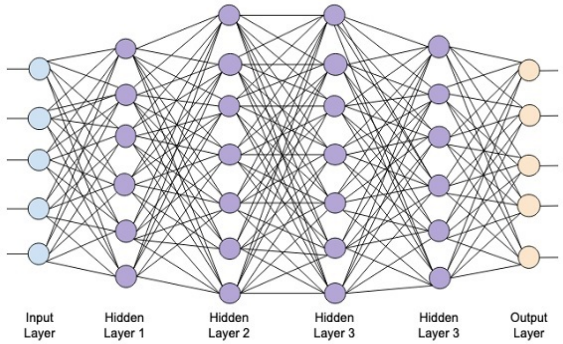


- Multi-layers Neural Networks.






x_1
x_2
x_3
x_4


Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Hidden Layer 3
Output Layer

- **Back-propagation (**Reverse autodifferention**)**

In 1986, (*Learning representations by back-propagating errors, Nature, 323(9): 533-536)* D. E. Rumelhart popularized the idea of **back propagation** to compute gradients. It is not a learning method, but a computational trick. It is actually a simple implementation of chain rule of derivatives.

BP algorithms as stochastic gradient descent algorithms (Robbins–Monro 1950; Kiefer- Wolfowitz 1951) with Chain rules of Gradient maps

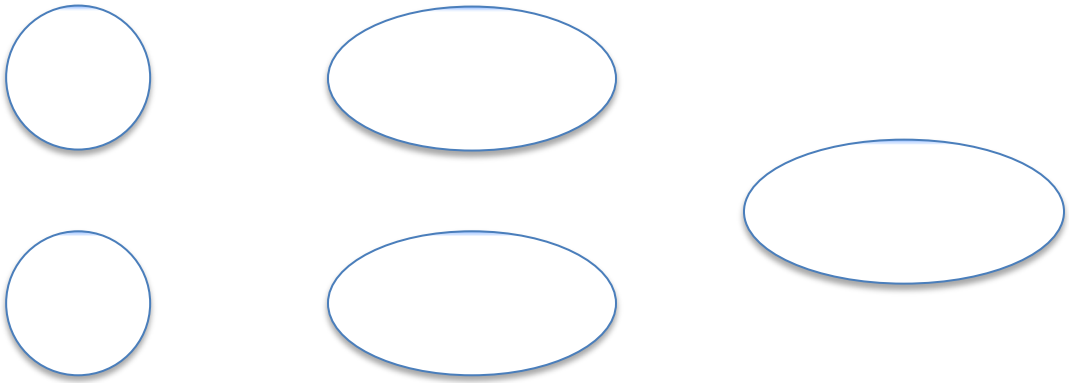
Goal: Minimize the loss function J

Need to calculate the gradient.

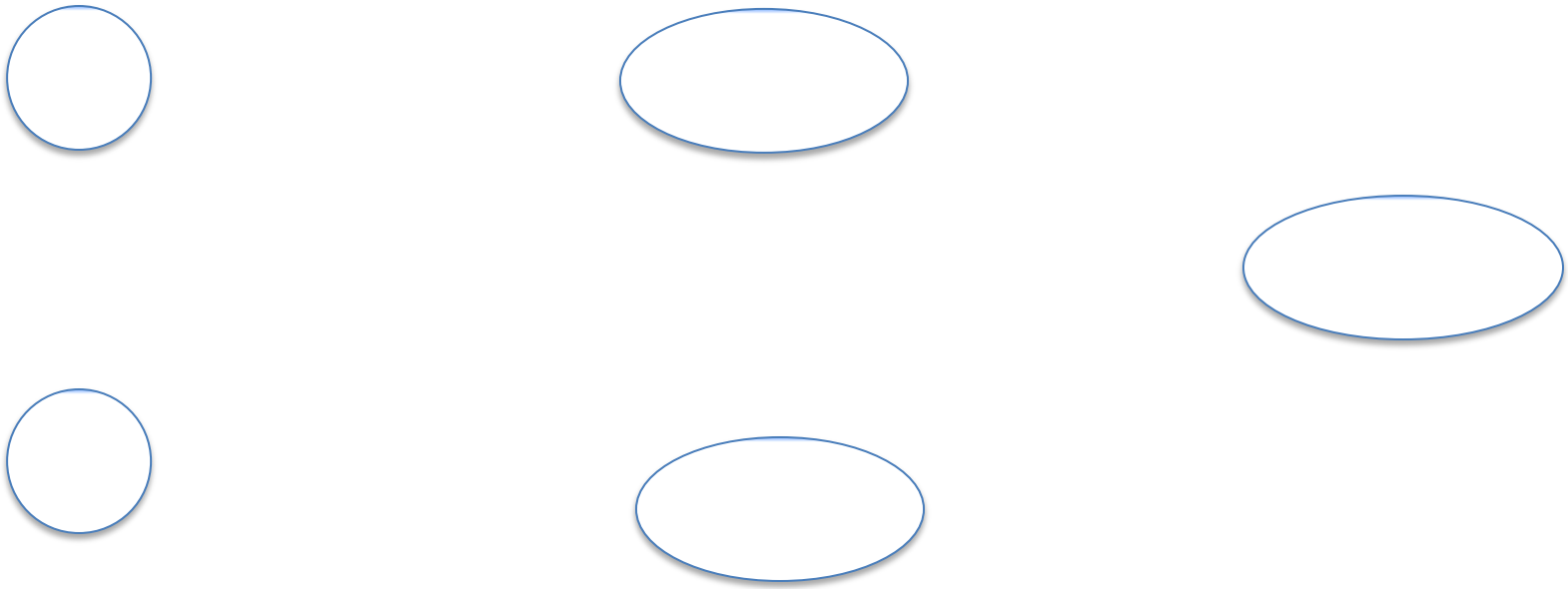
- The Chain Rule:

- Computation Graphs

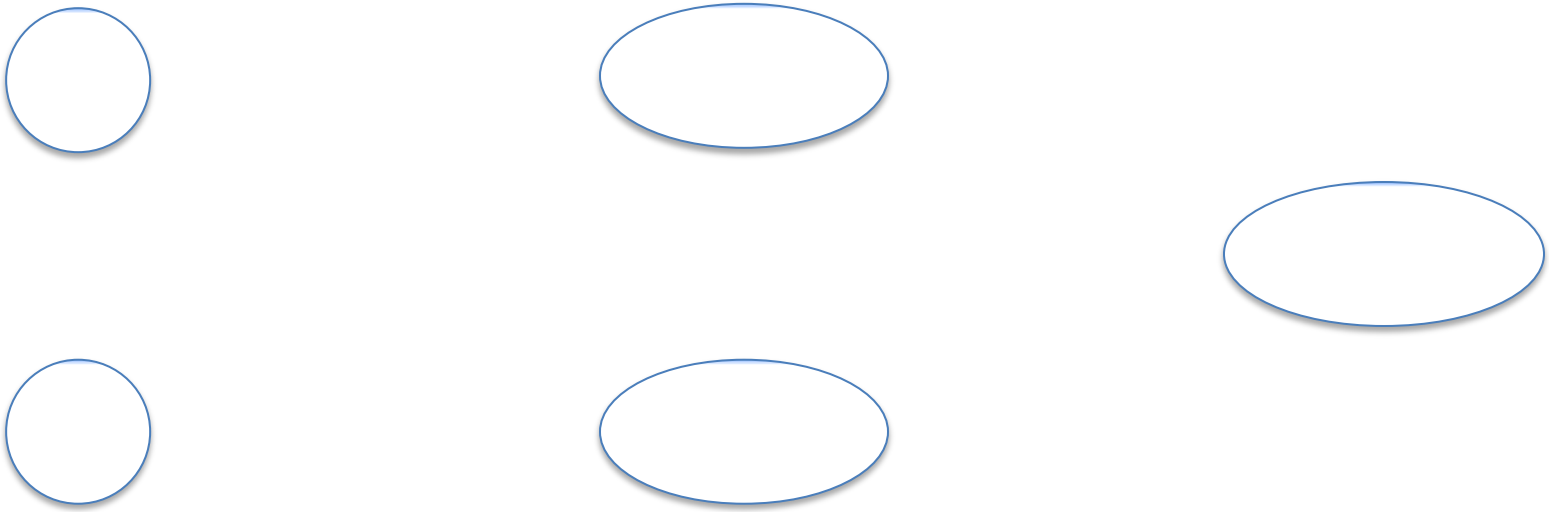
Function



Forward mode:



Backward mode:

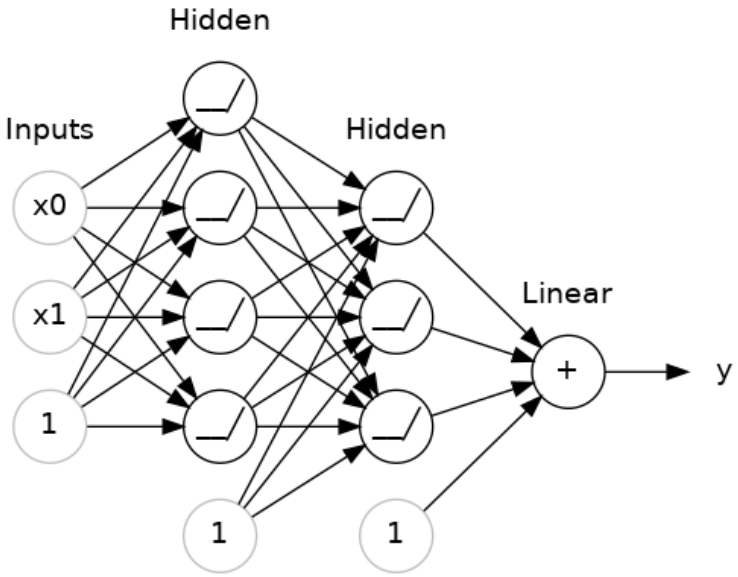


Neural Network by Pyhton:

TensorFlow2: https://www.tensorflow.org/

Keras on TensorFlow: https://keras.io/examples/

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

model = keras.Sequential([
        # the hidden ReLU layers
        layers.Dense(units=4, activation='relu', input_shape=[2]),
        layers.Dense(units=3, activation='relu'),
        # the linear output layer
        layers.Dense(units=1),
])
```



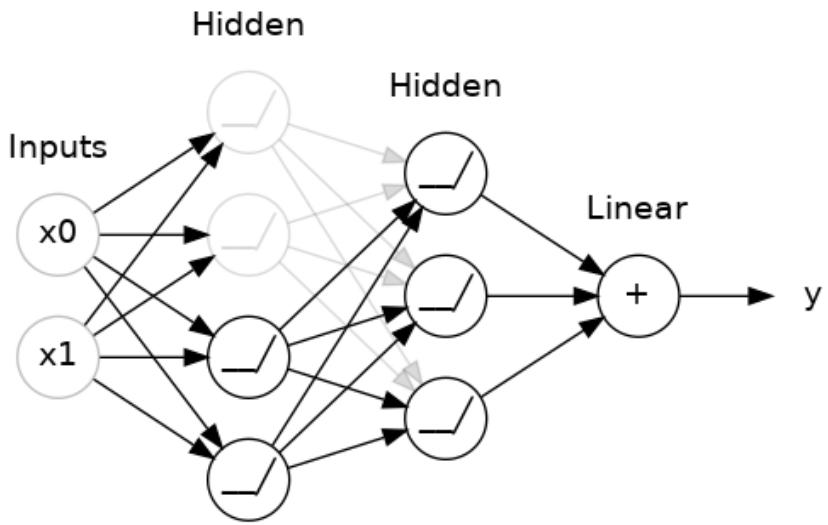
➢ Dropout and Batch Normalization

1. Dropout layer can help correct overfitting. We randomly *drop out* some fraction of a layer's input units every step of training. The weight patterns tend to be more robust.

2. Batch Normalization is something like scikit-learn's StandardScaler or MinMaxScaler.

Batch normalization layer looks at each batch as it comes in, first normalizing the batch with its own mean and standard deviation, and then also putting the data on a new scale with two trainable rescaling parameters. Batch normalization, in effect, performs a kind of coordinated rescaling of its inputs.

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

model = keras.Sequential([
        # the hidden ReLU layers
        layers.Dense(units=4, activation='relu', input_shape=[2]),
        layers.Dropout(0.3), # apply 30% dropout to the next layer
        layers.BatchNormalization(),
        layers.Dense(units=3, activation='relu'),
        layers.Dropout(0.3), # apply 30% dropout to the next layer
        layers.BatchNormalization(),
        # the linear output layer
        layers.Dense(units=1),
])
```



➢ Early Stopping

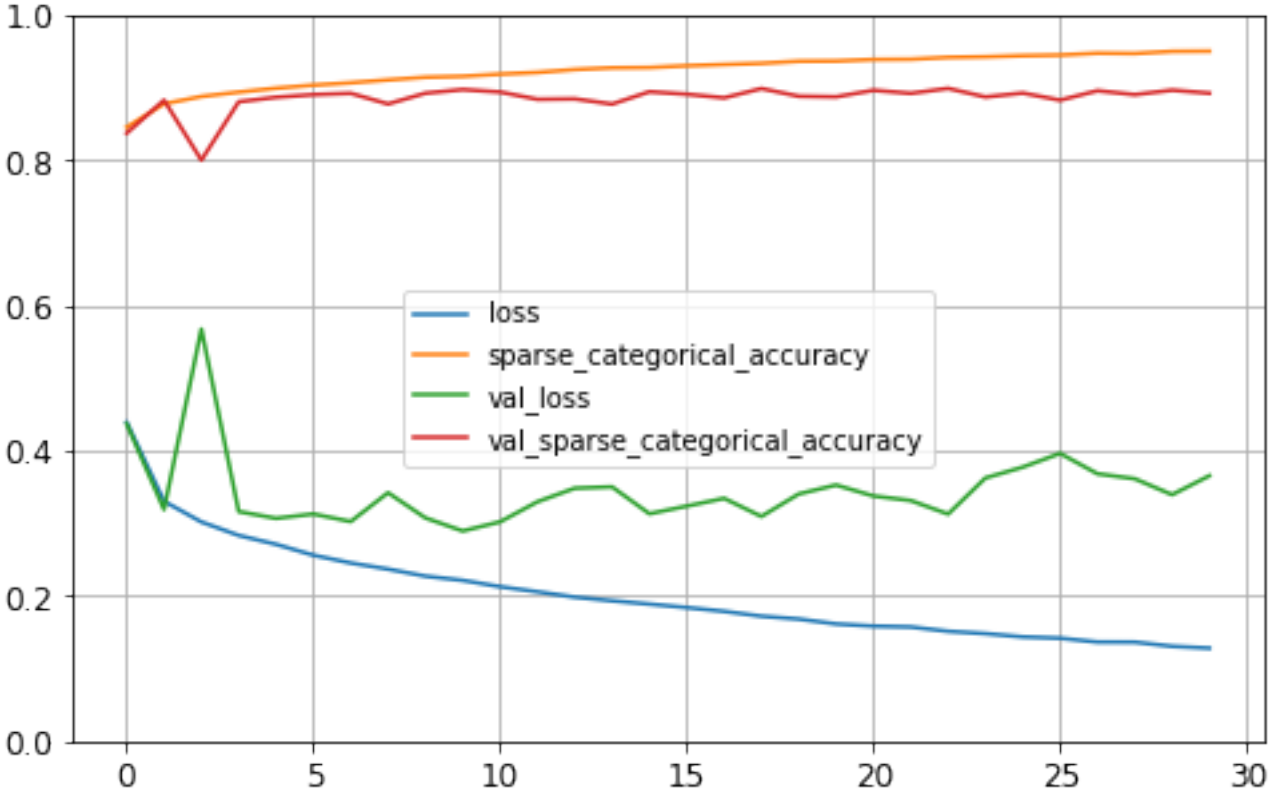
```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import layers, callbacks

early_stopping = EarlyStopping(
        min_delta=0.001, # minimium amount of change to count as an improvement
        patience=20, # how many epochs to wait before stopping
        restore_best_weights=True,
)
```

These parameters say: "If there hasn't been at least an improvement of 0.001 in the validation loss over the previous 20 epochs, then stop the training and keep the best model you found."
It can sometimes be hard to tell if the validation loss is rising due to overfitting or just due to random batch variation. The parameters allow us to set some allowances around when to stop.

```
model.compile(loss="sparse_categorical_crossentropy",
            optimizer="sgd",
            metrics=["accuracy"])
```

```
history = model.fit(X_train, y_train, epochs=30,
                validation_data=(X_valid, y_valid))
```

```
import pandas as pd

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)

plt.show()
```



A visual proof that neural nets can compute any function

http://neuralnetworksanddeeplearning.com/chap4.html

Play with neural network:

http://playground.tensorflow.org/

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Online book about neural network:

http://neuralnetworksanddeeplearning.com/chap3.html

MIT Introduction to Deep Learning | 6.S191

https://www.youtube.com/watch?v=5tvmMX8r_OM