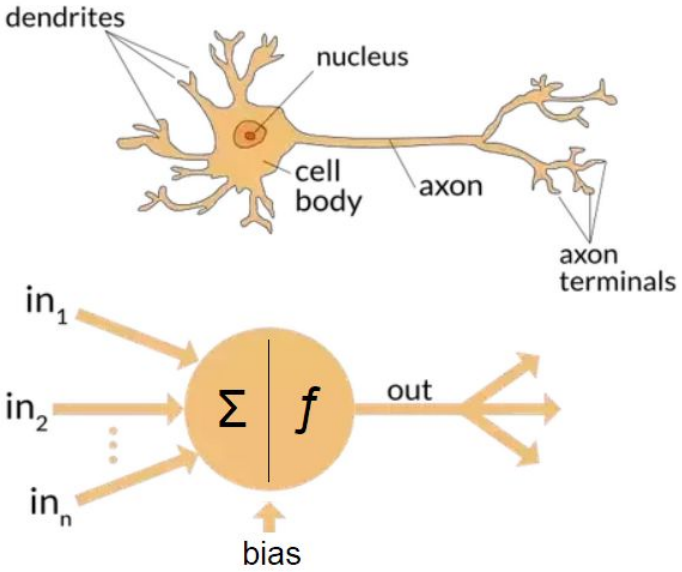
Artificial Neural Networks

Overview

- Neuron
- Perceptron
- Multi Layer Perceptron
- Loss functions
- Backpropagation
- Demo implementation

Neuron

ANNs are composed of artificial neurons which are conceptually derived from biological neurons.



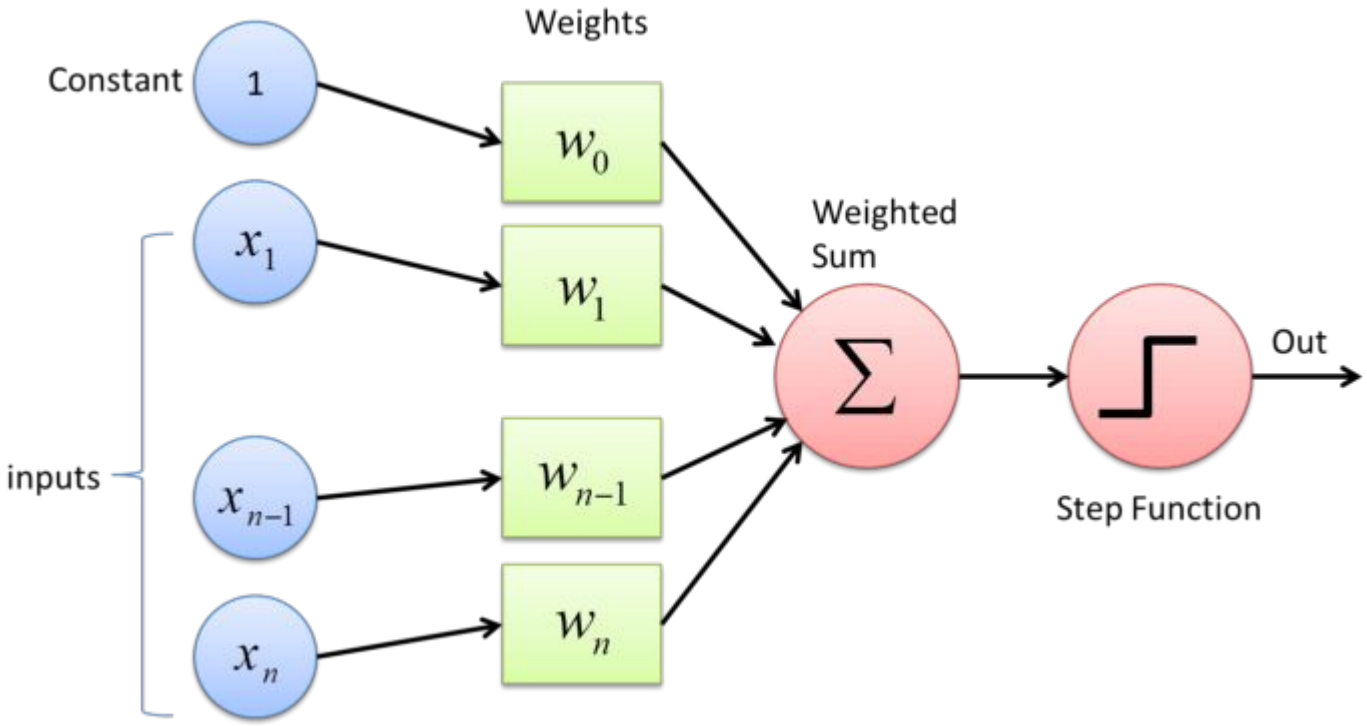
Perceptron

Perceptron is a single layer neural network

The perceptron consists of 4 parts:

1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function

Perceptron


Weights
Constant
1
w_0
x_1
w_1
Weighted
Sum
Σ
Out
inputs
x_{n-1}
w_{n-1}
Step Function
x_n
w_n

Perceptron

How does it work ?

- All the inputs ***x*** are multiplied with their weights ***w***.
- ***Add*** all the multiplied values and call them ***Weighted Sum***
- ***Apply*** that weighted sum to the correct ***Activation Function***.
- Get the output after applying Activation Function

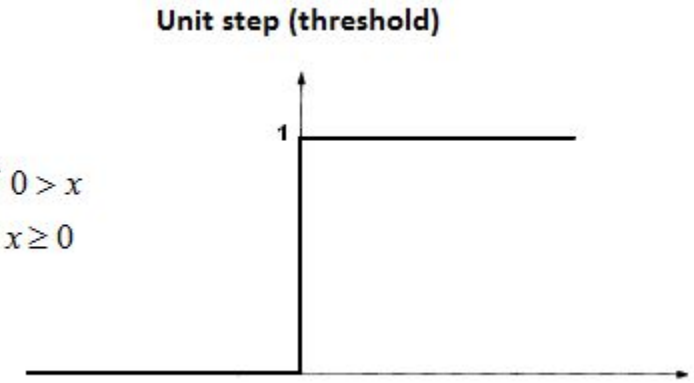
Perceptron

Activation Function:

The **activation function** of a node defines the output of that node given an input or set of inputs.

Eg :

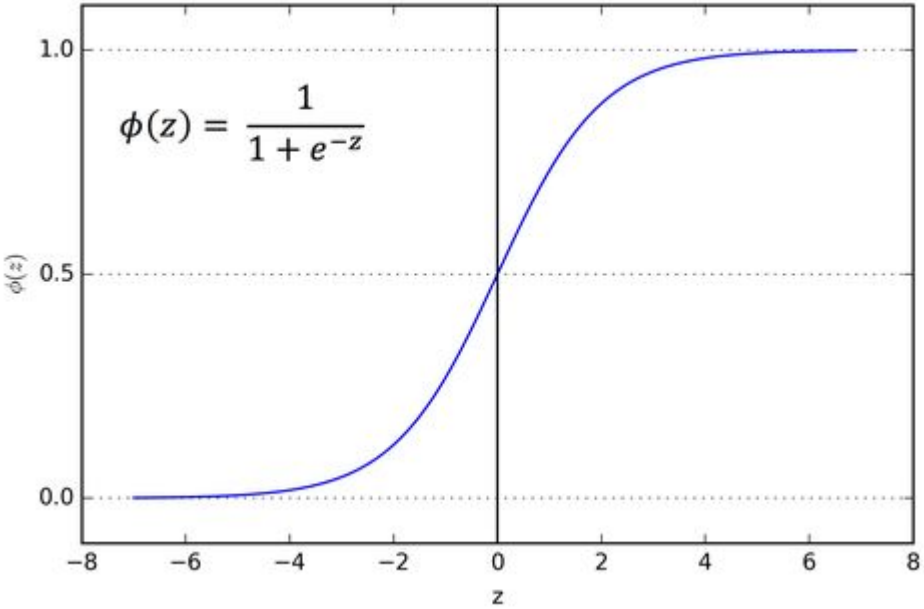
$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



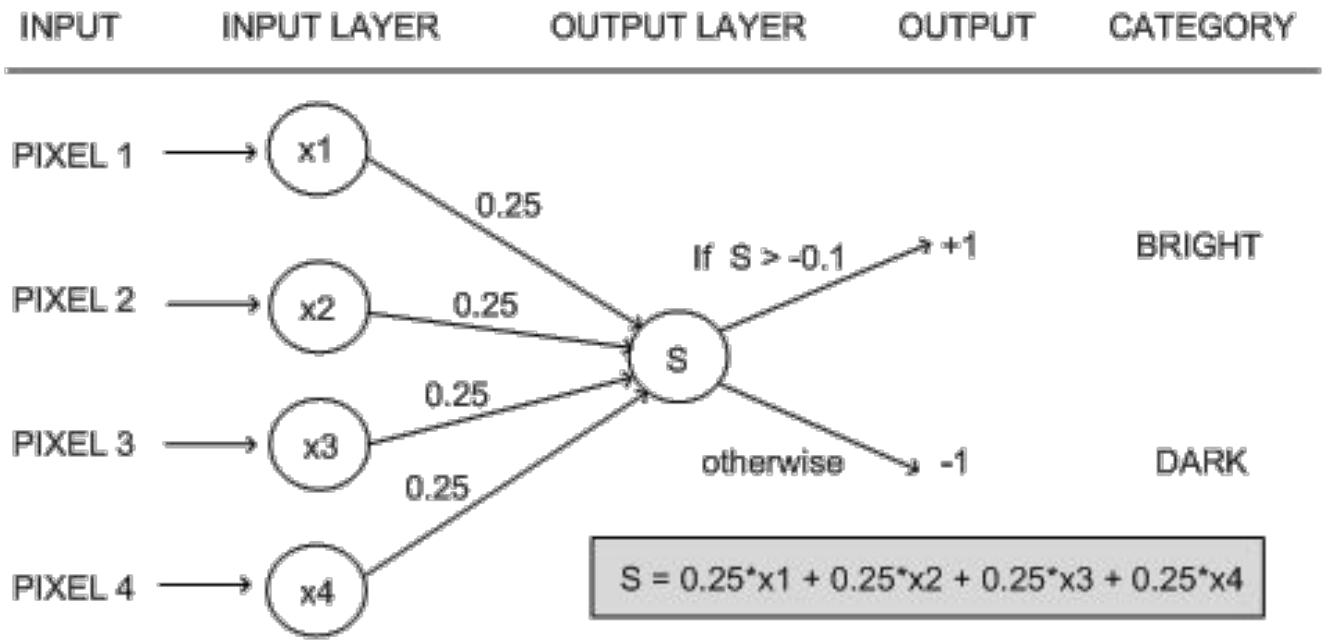
Perceptron

Sigmoid activation:

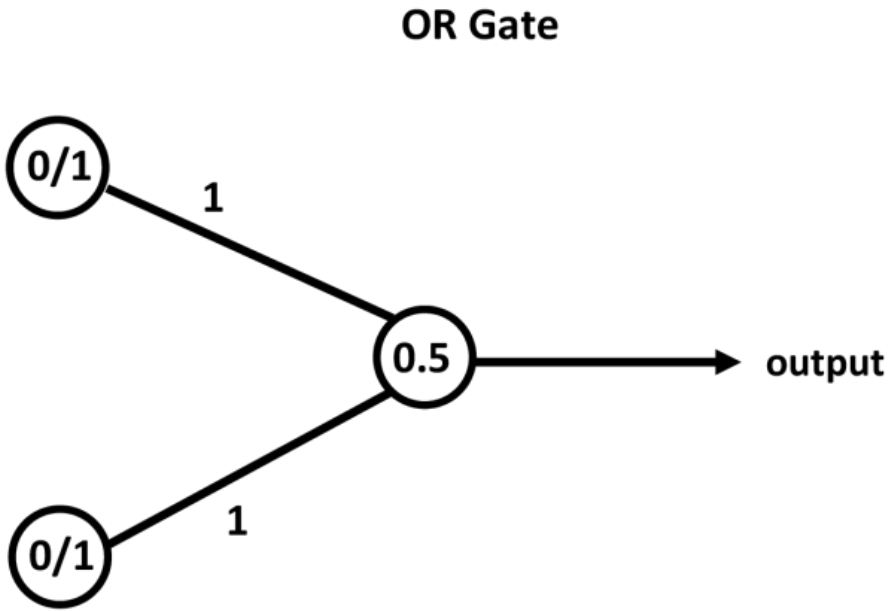
:



Perceptron

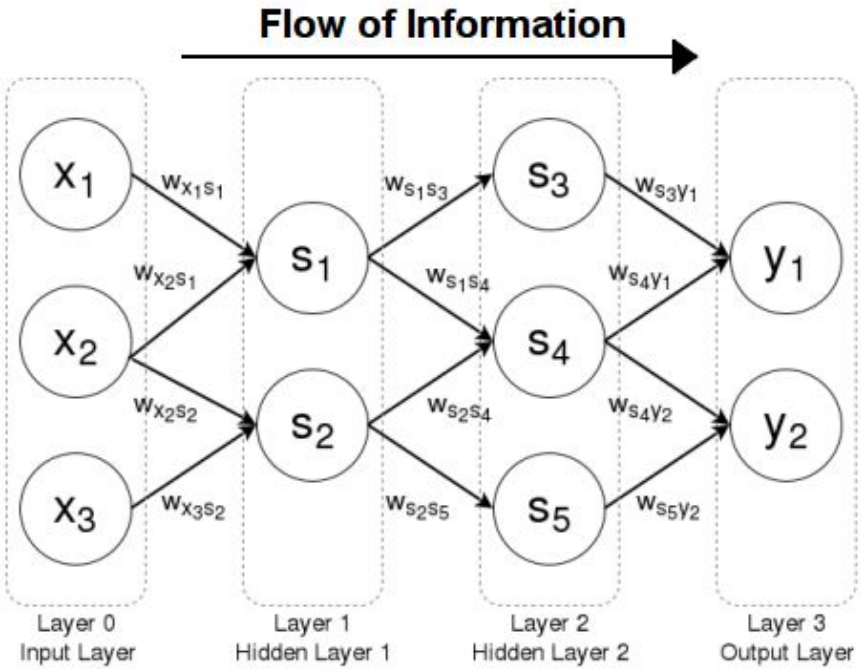

INPUT
INPUT LAYER
OUTPUT LAYER
OUTPUT
CATEGORY
PIXEL 1
PIXEL 2
PIXEL 3
PIXEL 4
x1
x2
x3
x4
0.25
0.25
0.25
0.25
S
If S > -0.1
+1
BRIGHT
otherwise
-1
DARK
S = 0.25*x1 + 0.25*x2 + 0.25*x3 + 0.25*x4

Perceptron



Multi Layer Perceptron

MLPs are neurons stacked in layers



Loss Functions

Machines learn by means of a loss function.

- It’s a method of evaluating how well specific algorithm models the given data.
- If predictions deviates too much from actual results, loss function be very large number.
- Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.

Loss Functions

Machines learn by means of a loss function.

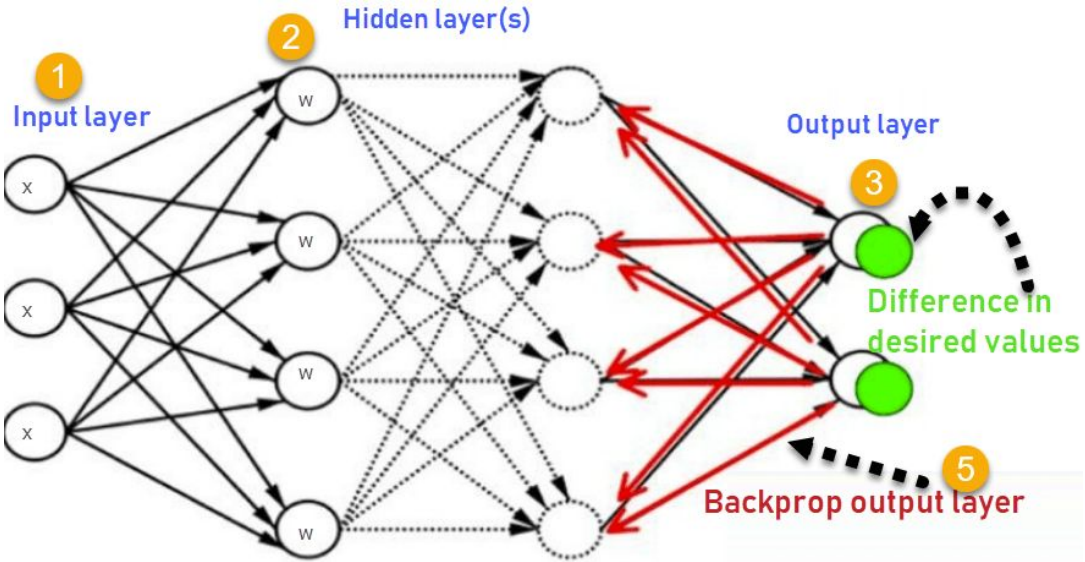
- It's a method of evaluating how well specific algorithm models the given data.
- If predictions deviates too much from actual results, loss function be very large number.
- Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.

Eg: Mean Squared Error Loss

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Backpropagation

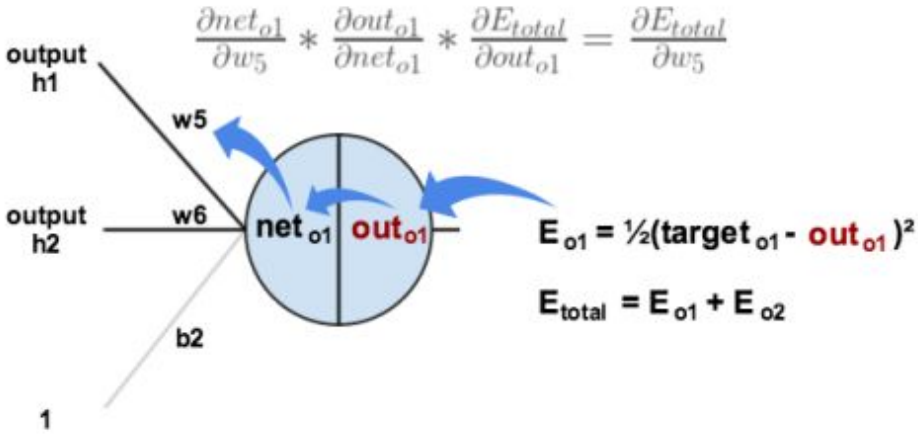
- Backpropagation is the essence of neural network training.
- It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration).
- Proper tuning of the weights allows you to reduce error rates and make the model reliable



Backpropagation

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole

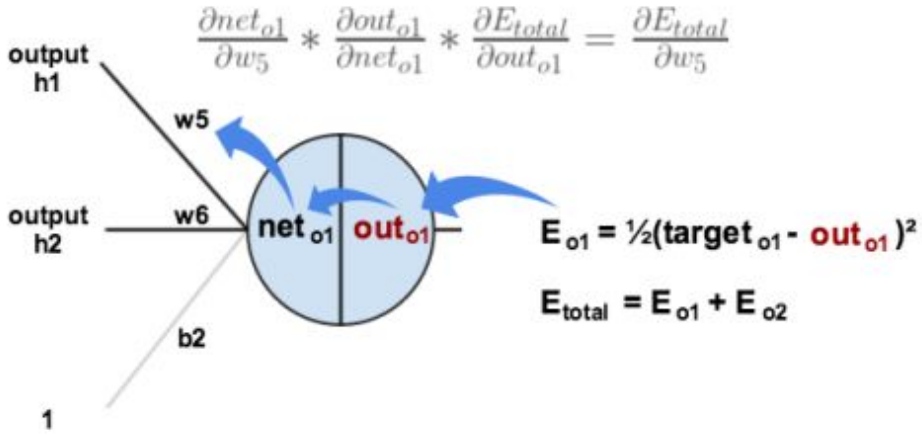
For output layer:



Backpropagation

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole

For output layer:



$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

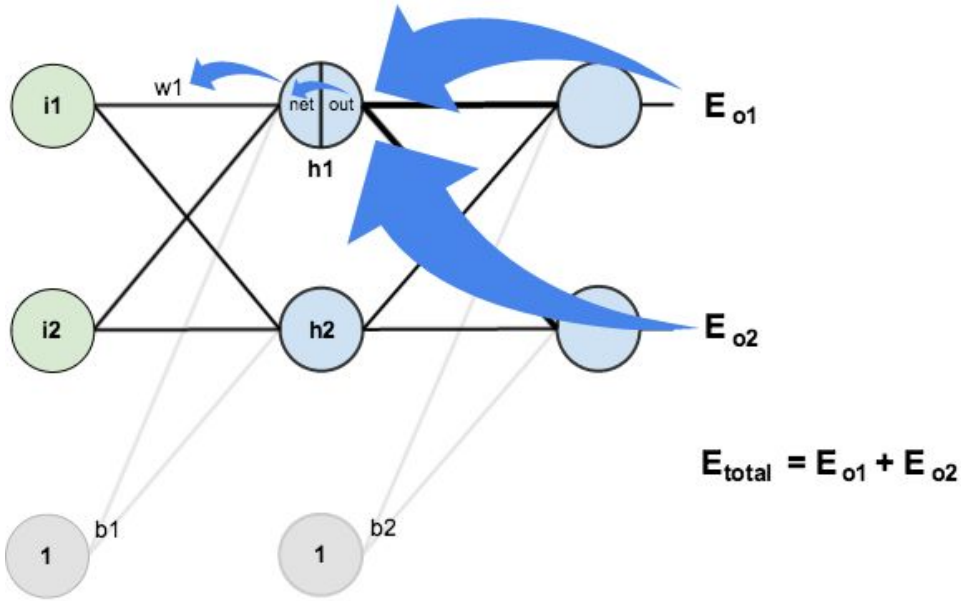
Backpropagation

For hidden layer:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



Demo Implementation (keras)

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```