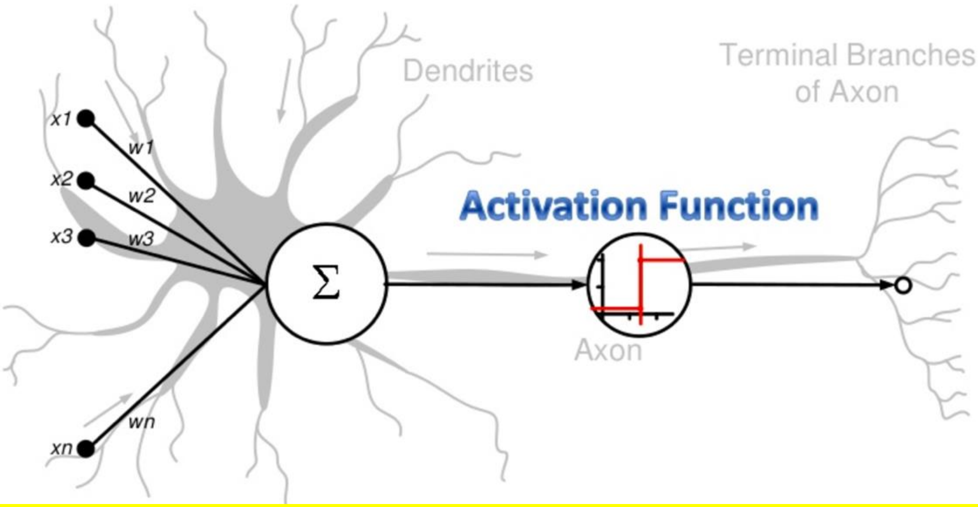
Neural Networks and Deep learning

Neural Network for Classification

- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it

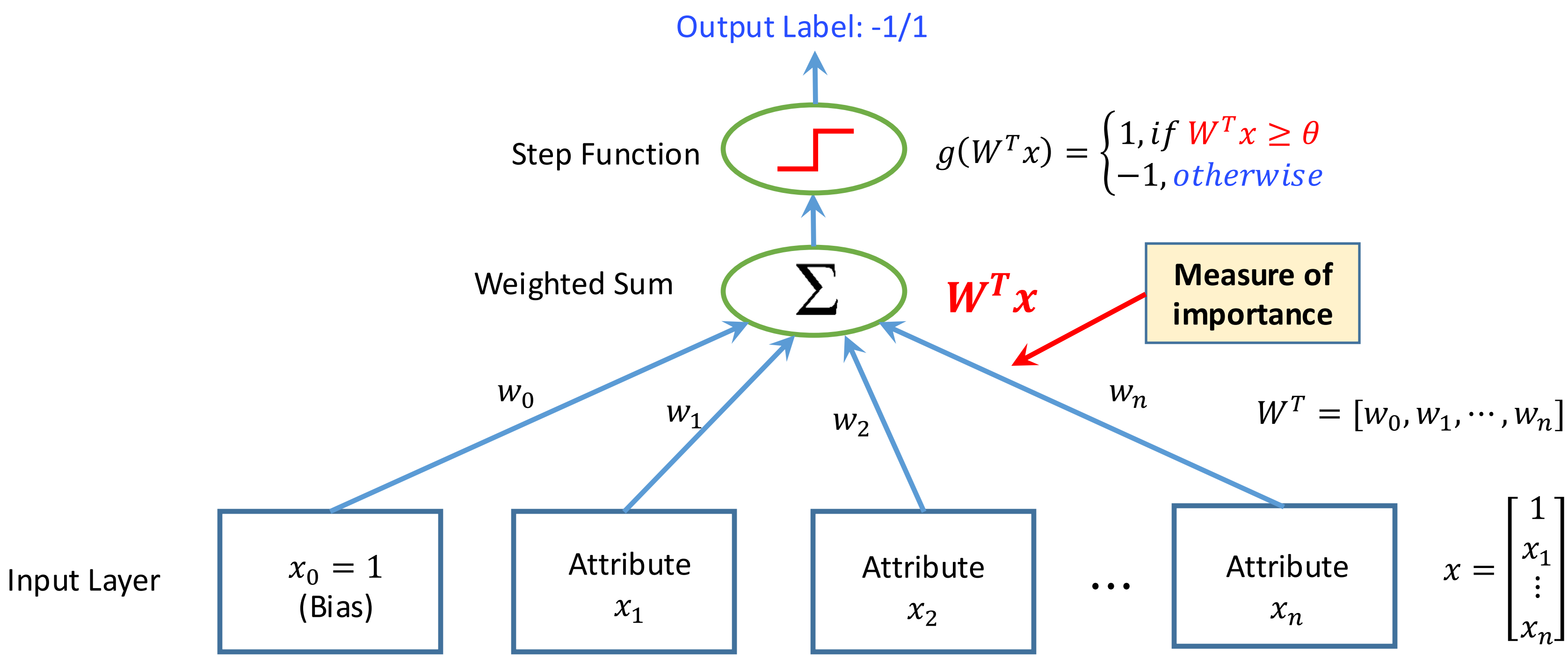
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples



Artificial Neural Networks as an analogy of Biological Neural Networks

Perceptron: Predecessor of a Neural Network

- A perceptron is a neuron, and connects its inputs to the output



Invented in 1957 by Frank Rosenblatt. The original perceptron model does not have a non-linear activation function

Perceptrons

- Very intuitive, and easy to implement.
- A good entry point to the (re-discovered) modern state-of-the-art machine learning algorithms: deep learning.

Artificial Neurons and the McCulloch-Pitts Model

- The initial idea of the perceptron dates back to the work of Warren McCulloch and Walter Pitts in 1943
- Drew an analogy between biological neurons and simple logic gates with binary outputs

A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

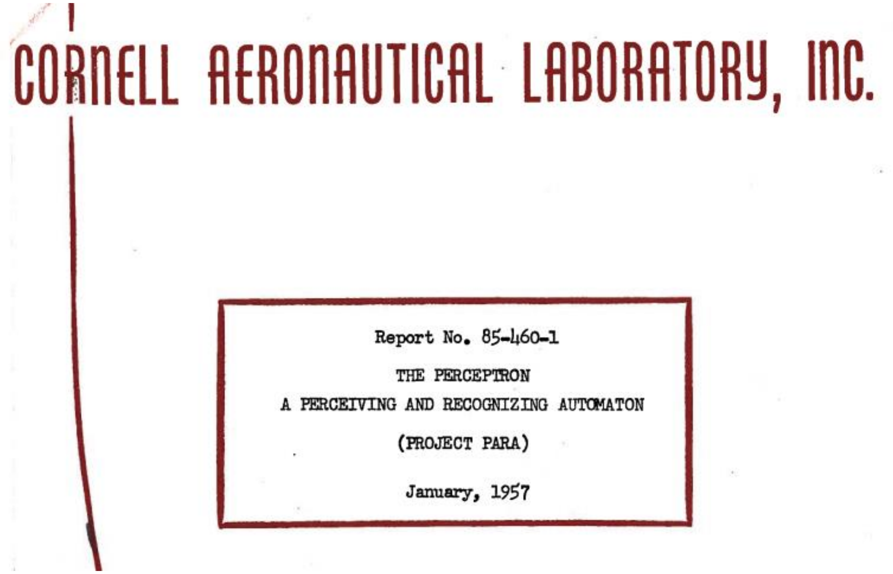
WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

Frank Rosenblatt's Perceptron

- Frank Rosenblatt published the first concept of the Perceptron learning rule in 1957.
- The main idea was to define an algorithm in order to learn the values of the weights *w* that are then multiplied with the input features in order to make a decision whether a neuron fires or not.



CORNELL AERONAUTICAL LABORATORY, INC.

Report No. 85-460-1

THE PERCEPTRON

A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf

Perceptron

- Data $x = \begin{bmatrix} 1 \\ A_1 \\ \vdots \\ A_n \end{bmatrix}$

- Weights:

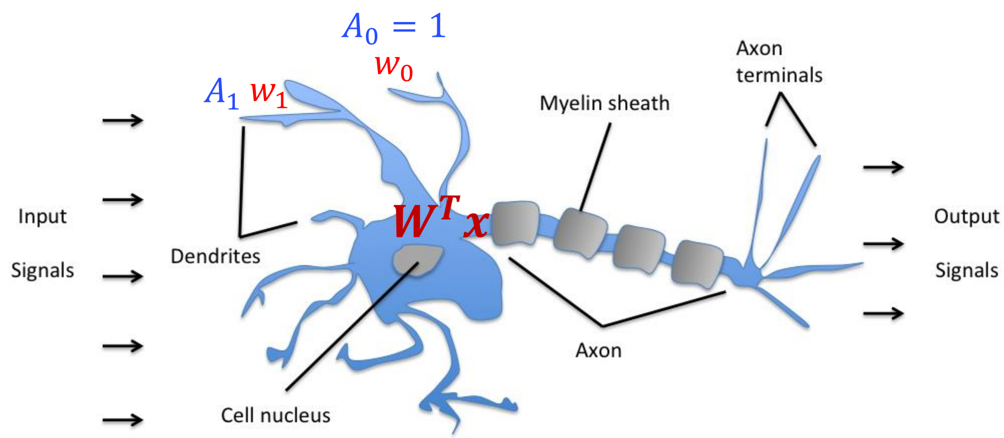
$$W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \qquad W^T = [w_0, w_1, \cdots, w_n]$$

  - Learn the values of the weights w that are then multiplied with the input features

- Accumulation

$$W^T x = w_0 + w_1 A_1 + w_2 A_2 + \cdots + w_n A_n$$

The signals of variable magnitudes arrive at the dendrites.



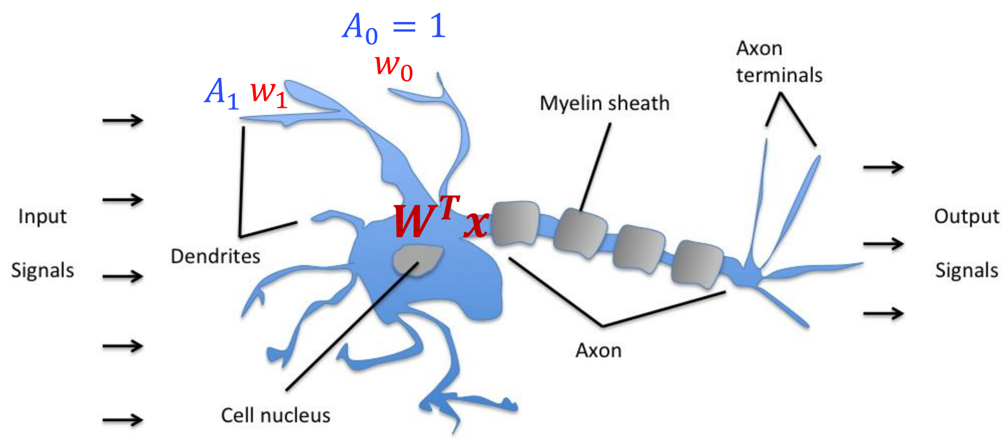
Schematic of a biological neuron.

Those input signals are then accumulated in the cell body of the neuron.

Perceptron

- If the accumulated signal exceeds a certain threshold, a output signal is generated that will be passed on by the axon.

$if\ W^T x \geq \theta, pass\ (predict\ 1);$
$otherwise\ predict$ -1



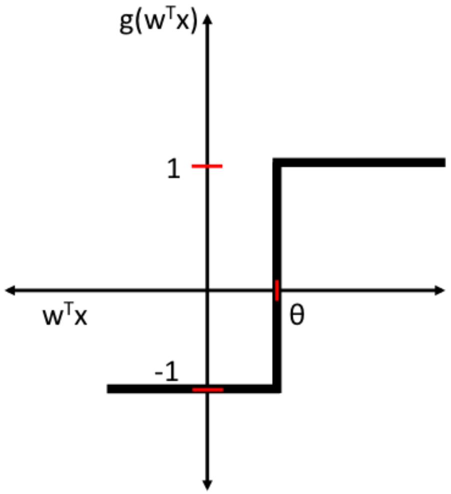
Schematic of a biological neuron.

$$g(W^T x) = \begin{cases} 1, if\ W^T x \geq \theta \\ -1, otherwise \end{cases}$$ activation function

Activation Function

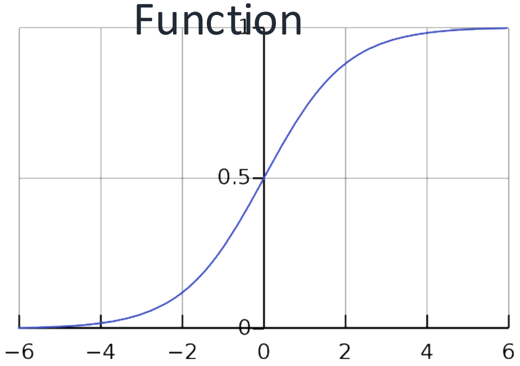
- Unit Step Function

$$g(W^T x) = \begin{cases} 1, if\ W^T x \geq \theta \\ -1, otherwise \end{cases}$$



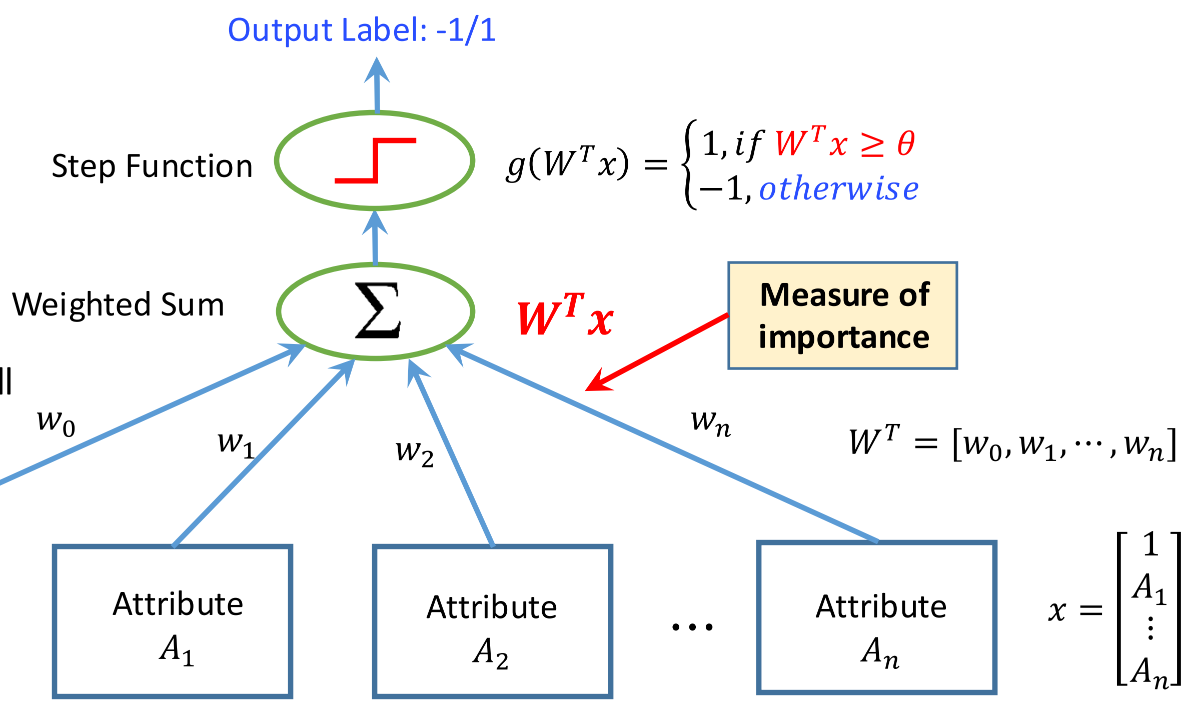
Unit step function.

Sigmoid Function



The Perceptron Learning Rule

- Step 2: For each training sample $\boldsymbol{x_i}$:
  - Calculate the *output* value.
  - Update the *weights*.
- Step 1: Initialize the *weights* to small random numbers.



The Perceptron Learning Rule

- The output value is the class label predicted by the unit step function: $g(W^T x)$
- The weight update can be written more formally as

$$w_j^{(t+1)} = w_j^{(t)} - \eta\big(y_i - g(W^T x)\big)(-A_j)$$

$$= w_j^{(t)} + \eta\underbrace{\boxed{\big(y_i - g(W^T x)\big)A_j}}_{\Delta w_j}$$

y_i is the ground truth label
A_j is the j-th feature of input x_i
η is the learning rate

- The value of Δw_j

$$\Delta w_j = \eta\big(-1 - (-1)\big)A_j = 0 \qquad \Delta w_j = \eta(1 - 1)A_j = 0$$

$$\Delta w_j = \eta(-1 - 1)A_j = \eta(-2)A_j \qquad \Delta w_j = \eta(1 - (-1))A_j = \eta(2)A_j$$

Sample Code

```python
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

breast_cancer = load_breast_cancer()

# create X (features) and y (response)
X = breast_cancer.data
y = breast_cancer.target

# split data with train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

print('# data =',len(X))
print('# training data =', len(X_train))
print('# testing data =', len(X_test))
```

```
# data = 569
# training data = 455
# testing data = 114
```

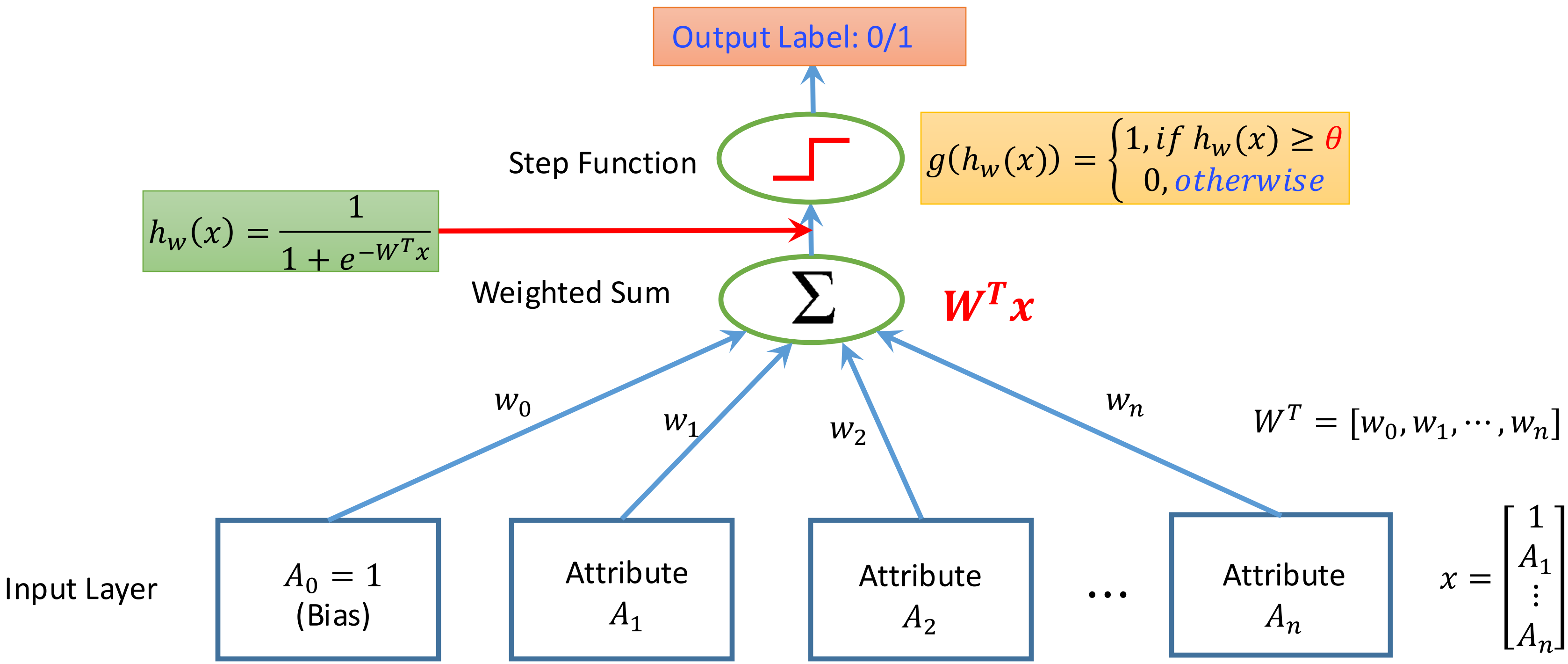
```python
from sklearn.linear_model import Perceptron
from sklearn import metrics

clf = Perceptron()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('Confusion Matrix:')
print(metrics.confusion_matrix(y_test, y_pred))
print('Accuracy =', metrics.accuracy_score(y_test, y_pred))
print('Precision =', metrics.precision_score(y_test, y_pred))
print('Recall =', metrics.recall_score(y_test, y_pred))
print('F1 =', metrics.f1_score(y_test, y_pred))
```

```
Confusion Matrix:
[[37  2]
 [17 58]]
Accuracy = 0.8333333333333334
Precision = 0.9666666666666667
Recall = 0.7733333333333333
F1 = 0.8592592592592593
```

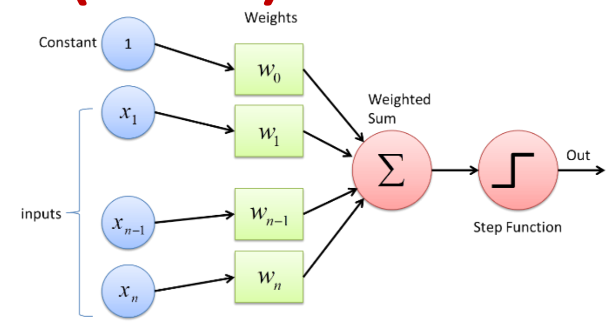
Perceptron v.s. Logistic Regression



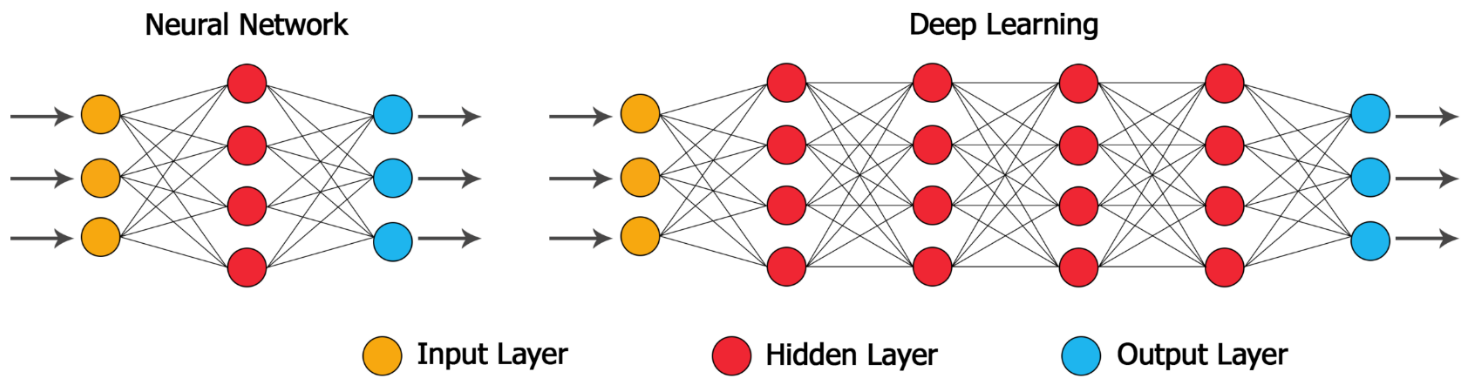
Problems with Perceptrons

- Cannot handle non-linear data
- Not differentiable
- Only work for binary classification problem
    - numeric output
    - multiple outputs

Multi Layered Perceptrons (MLP)

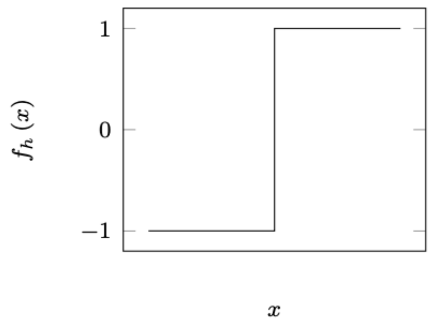


- Generalizing to Multiple Labels
  - Distinguishing between multiple categories
  - Solution: Add another layer - Multi Layer Neural Networks

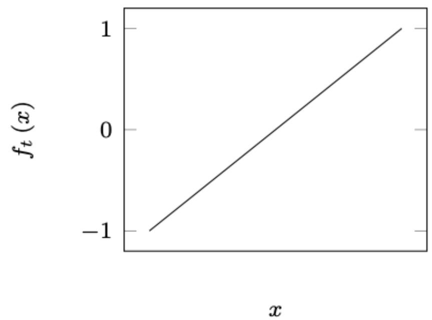


Multi Layered Perceptrons (MLP)

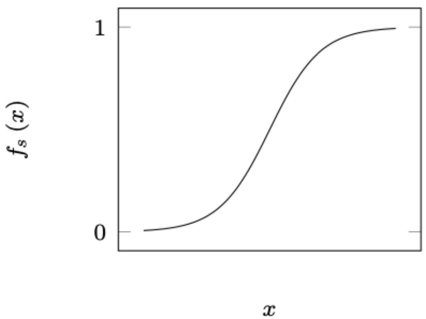
- Multi-class classification is more applicable than binary classification
  - To handle non-linear data
  - Smooth, differentiable threshold function



unit step function



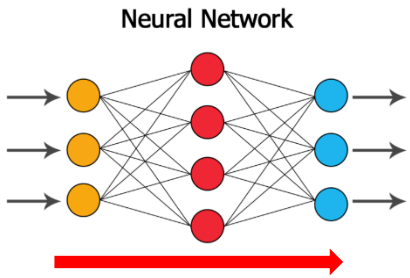
linear function



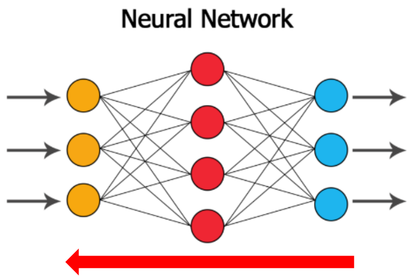
Sigmoid function

Neural Network Optimization

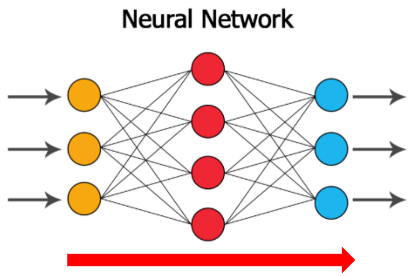
- Feed Forward Neural Networks



- Error Backpropagation

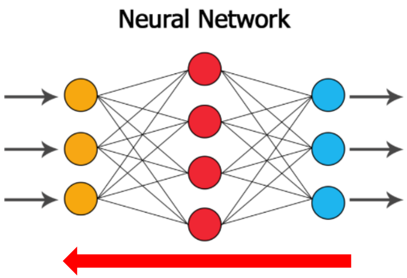


Feed Forward Neural Networks



- Information only flows in one direction (forward)
- Each hidden node "collects" the inputs from all input nodes and computes a weighted sum of the inputs and then applies the sigmoid function to the weighted sum.
- The output of each hidden node is forwarded to every output node.
- The output node "collects" the inputs (from hidden layer nodes) and computes a weighted sum of its inputs and then applies the sigmoid function to obtain the final output.
- The class corresponding to the output node with the largest output value is assigned as the predicted class for the input.

Backpropagation



- Assume that the network structure is predetermined (number of hidden nodes and interconnections)
- Objective function for n training examples with k categories:

$$L = \sum_{i=1}^{n} \sum_{l=1}^{k} (y_{il} - o_{il})^2$$

y_{il} – Target value associated with the l–th class for input (x_i)

$y_{il} = 1$ when l is true class for x_i, and 0 otherwise

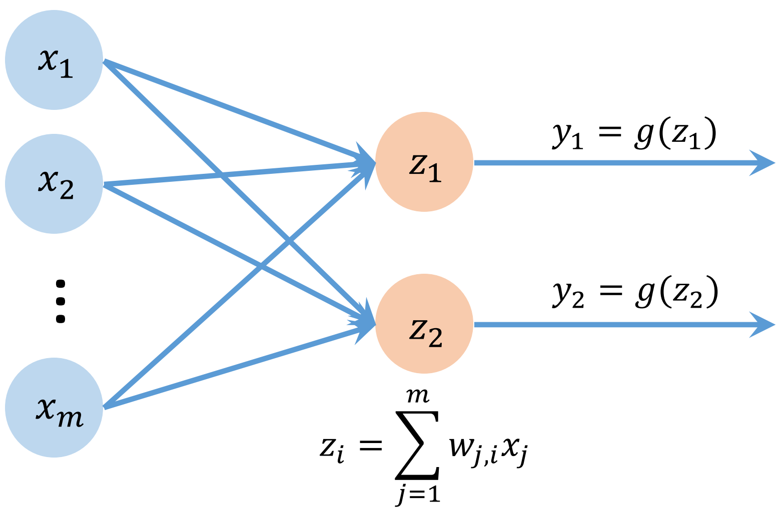
o_{il} – Predicted value associated with the l–th class for input (x_i)

Gradient Descent

- Initialize all weights to small values
- For each training example, ⟨x, y⟩:
  - **Propagate input forward** through the network
  - **Propagate errors backward** through the network

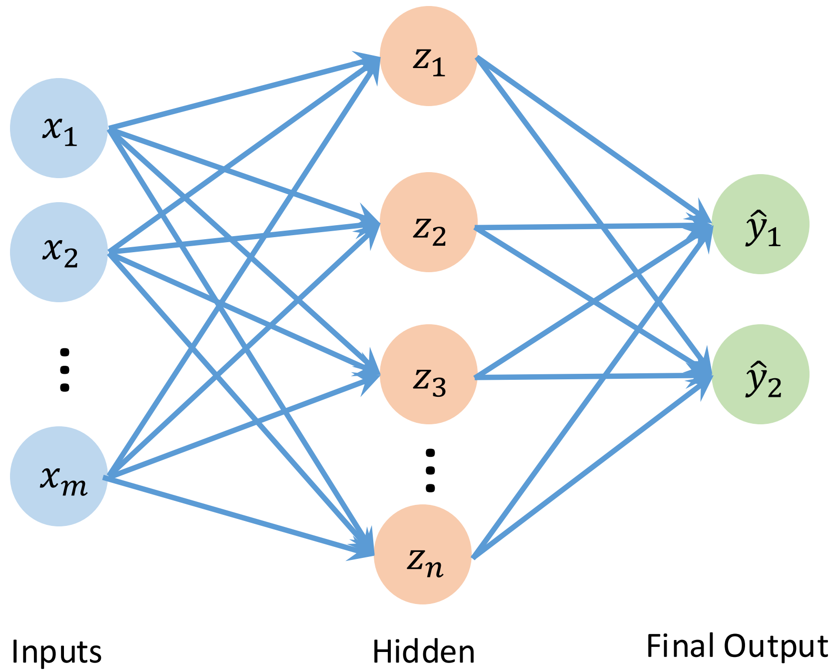
Multiple Output Perceptron

- Because all inputs are densely connected to all outputs, these are called Dense layer.

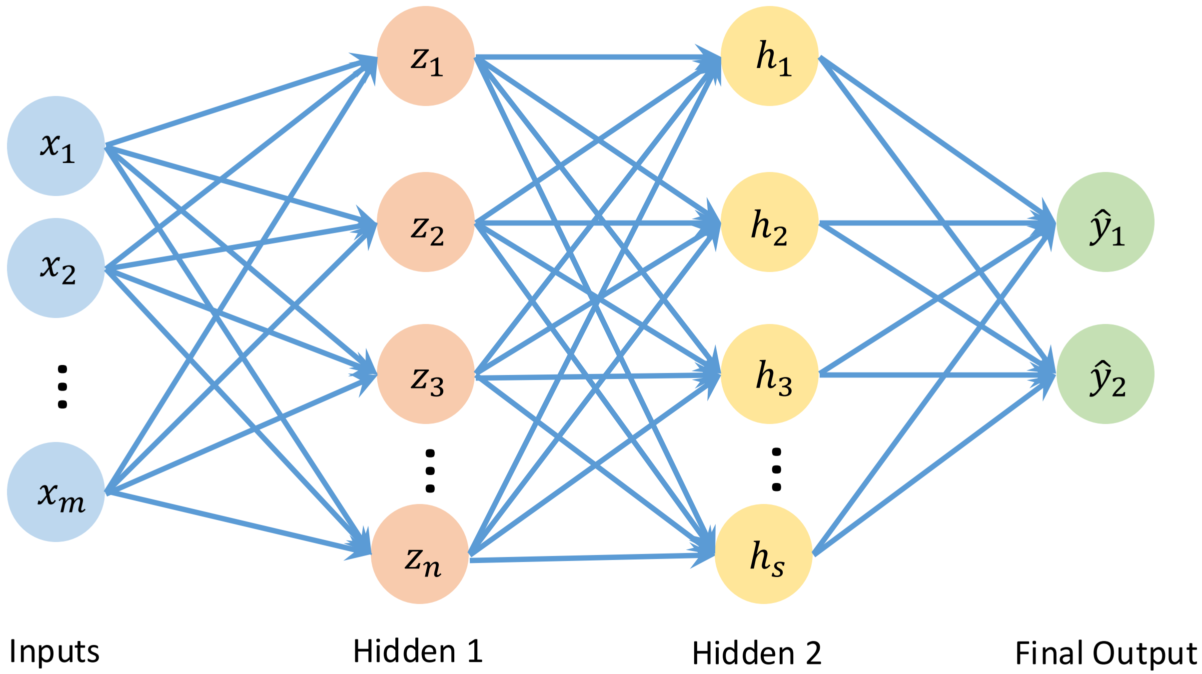


g *is a nonlinear function.*

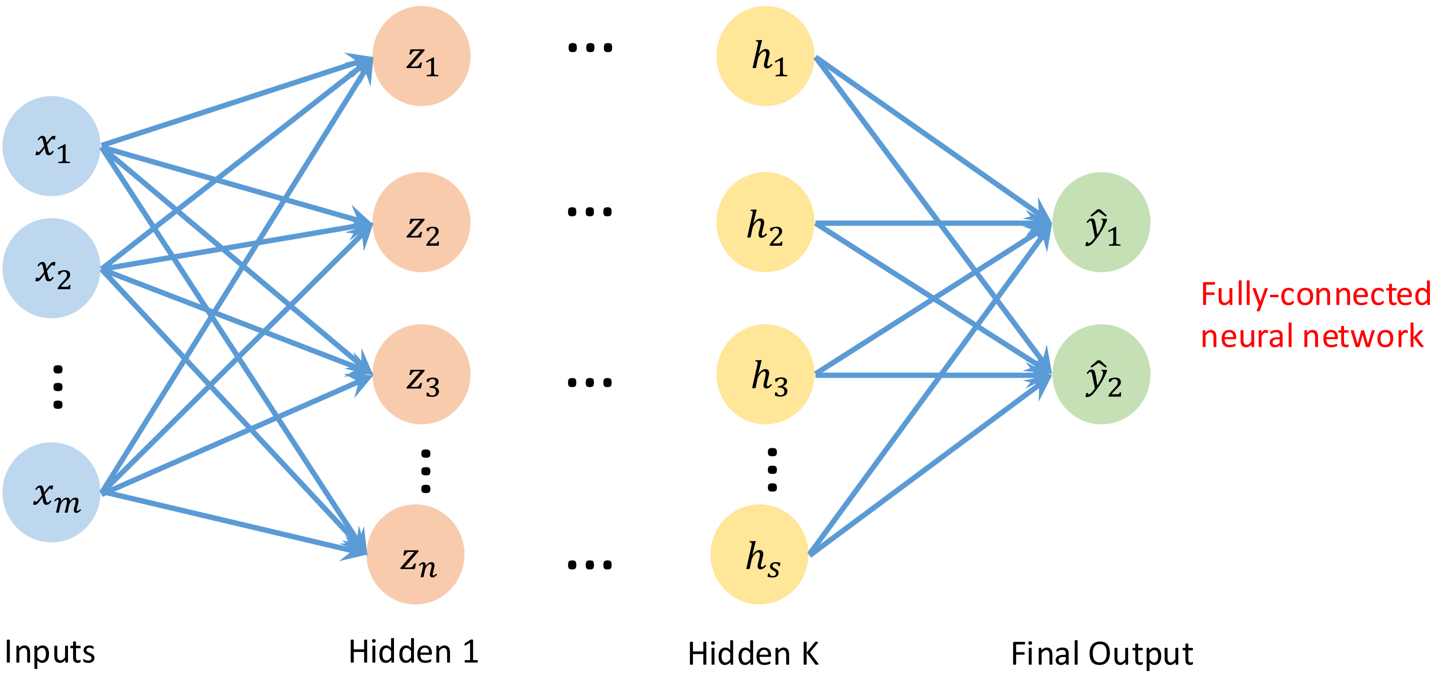
Single Layer Neural Network



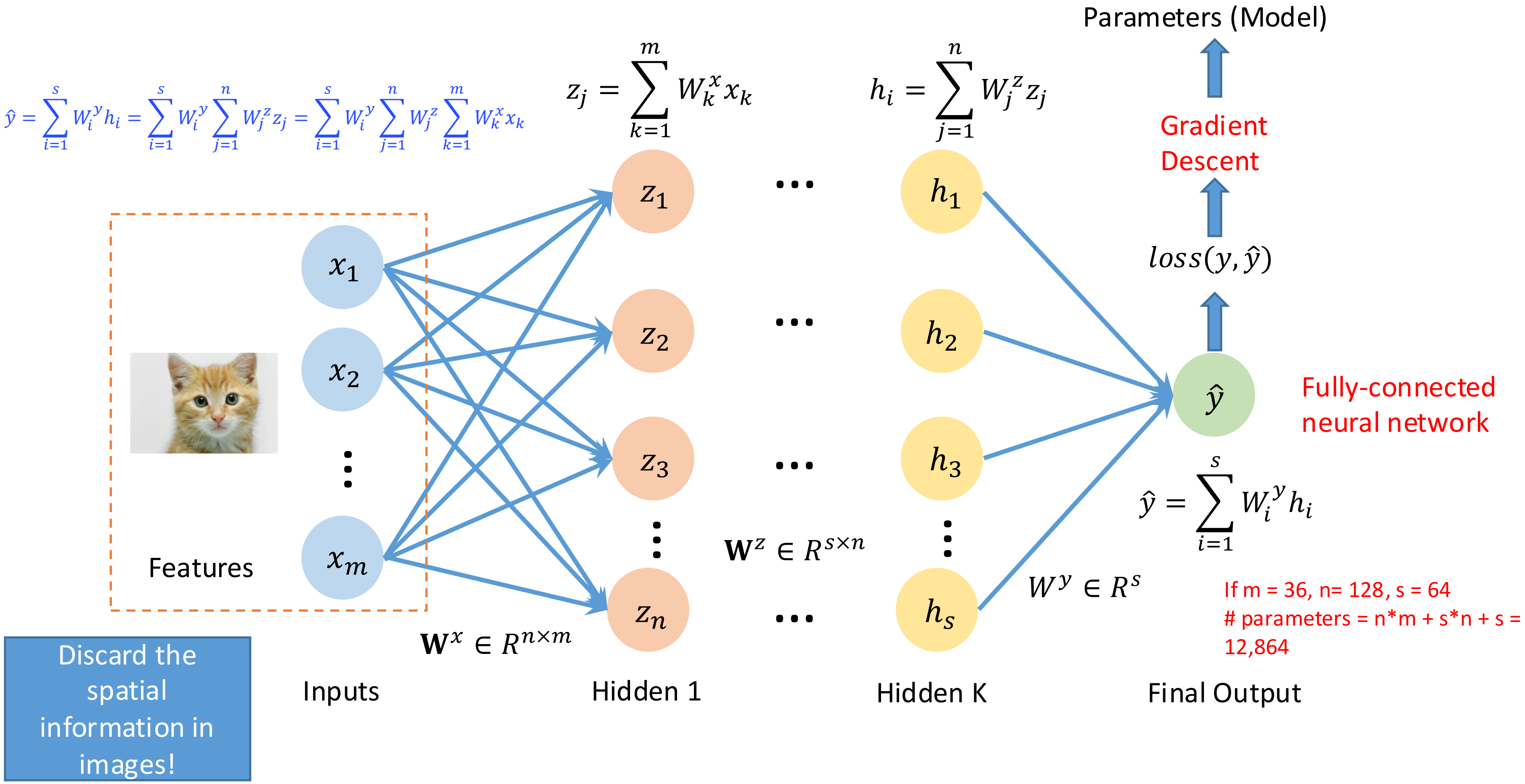
Two-Layer Neural Network



Deep Neural Network



Deep Neural Network



Three Steps for Deep Learning