CME 216, ME 343 - Spring 2020

Eric Darve, ICME

A perceptron is a function that takes as input a vector x and outputs a real number.

Its formula is given by

$$h_{w,b}(x) = \phi(w^T x + b)$$

If you choose ϕ to be a linear function, you can recognize that a perceptron is doing a linear regression of the data.

That is, it tries to fit a hyperplane to the data by minimizing the mean squared error.

With a non-linear function ϕ we can build more general surfaces that fit the data but this is clearly very limited.

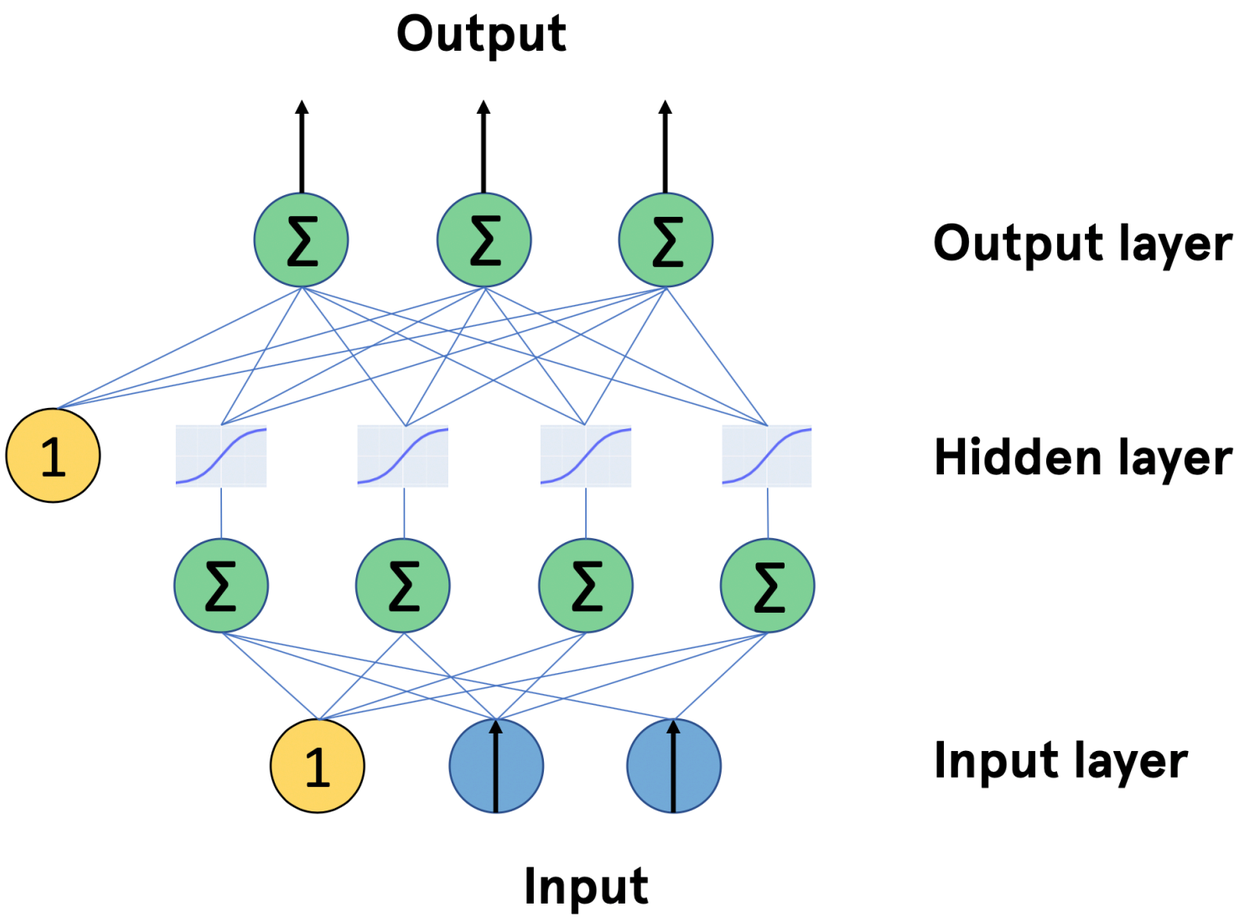
Consider for example the XOR function:

Input 1	Input 2	Output
0	0	0
1	1	0
0	1	1
1	0	1

Can you build a perceptron that reproduces this data?

Unfortunately, this is not possible.

This has led to the idea of stacking perceptrons together to create multilayered perceptrons or MLP.


Output
Σ
Σ
Σ
Output layer
1
Hidden layer
Σ
Σ
Σ
Σ
1
Input layer
Input

Caption of figure

The $\sum$ symbol corresponds to the linear matrix-vector multiplication $\boldsymbol{W}^{(i+1)}\boldsymbol{x}^{(i)}$.

$\boldsymbol{x}^{(i)}$ are the activation values for layer $\boldsymbol{i}$.

It's a vector. The size of $\boldsymbol{x}^{(i)}$ is the number of nodes in layer $\boldsymbol{i}$.

$$W^{(i+1)}x^{(i)}$$

$W^{(i+1)}$ is a matrix.

The number of columns of $W^{(i+1)}$ is the size of $x^{(i)}$.

The number of rows of $W^{(i+1)}$ is the size of the next layer, $x^{(i+1)}$.

The 1 corresponds to adding the bias $\boldsymbol{b}^{(i+1)}$:

$$\boldsymbol{W}^{(i+1)}\boldsymbol{x}^{(i)} + \boldsymbol{b}^{(i+1)}$$

$\boldsymbol{b}^{(i+1)}$ is now a vector. Its size is the same as $\boldsymbol{x}^{(i+1)}$.

After the linear transformation

$$W^{(i+1)}x^{(i)} + b^{(i+1)}$$

we apply a non-linear activation function ϕ:

$$z_j^{(i+1)} = [W^{(i+1)}x^{(i)} + b^{(i+1)}]_j$$

$$x_j^{(i+1)} = \phi(z_j^{(i+1)})$$

This type of structure is called an artificial neural network or ANN.

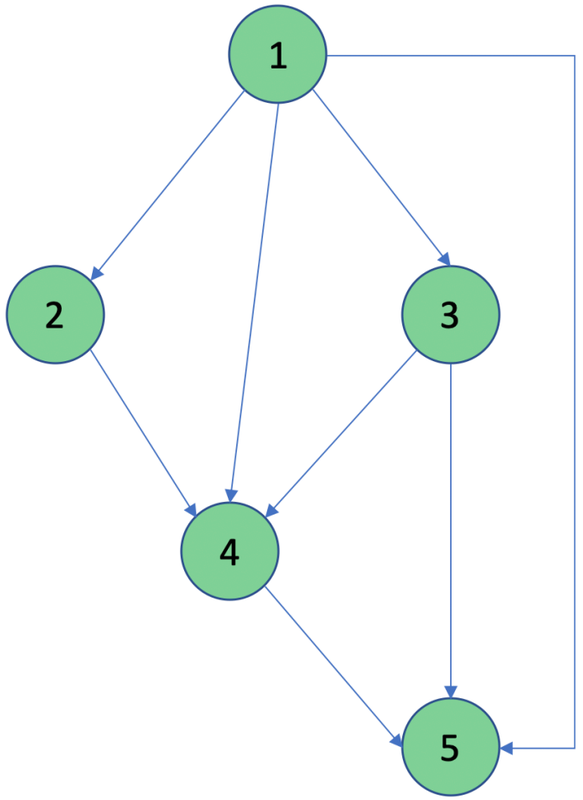
Generally speaking, an ANN is a general graph, that is directed (information only goes in one direction) and acyclic (no cycles).

Each edge e corresponds to a multiplication by some weight w_e.

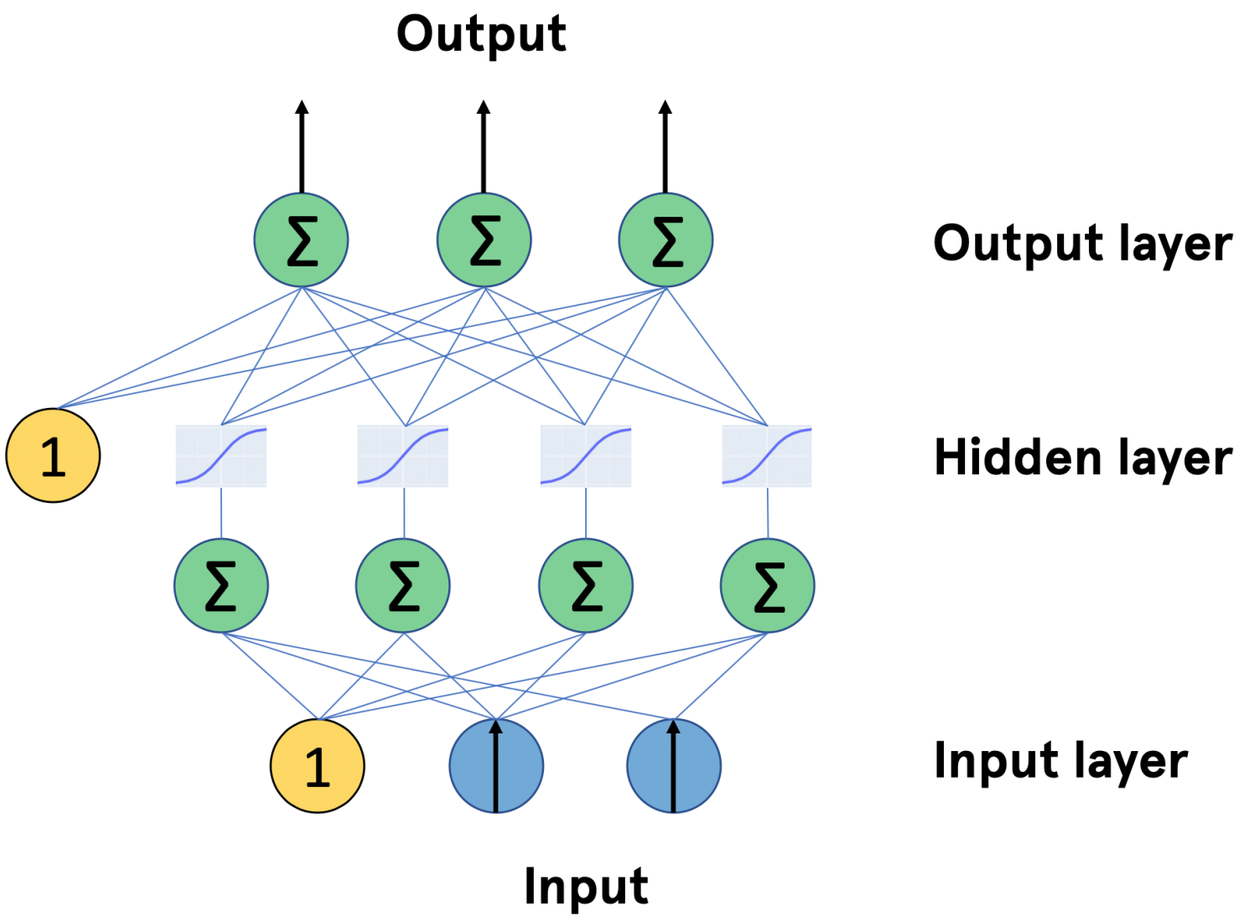
Multiple incoming edges are summed up together.

Then a non-linear function ϕ is applied to the result.

Example of DAG with 5 nodes


1
2
3
4
5

But in most cases, ANNs are organized in layers as we have seen previously.


Output
Σ
Σ
Σ
Output layer
1
Hidden layer
Σ
Σ
Σ
Σ
1
Input layer
Input

At each layer we perform the sequence of operations:

- Input: $\boldsymbol{x}^{(i)}$
- Multiply by weights and add bias:

$$z_j^{(i+1)} = [\boldsymbol{W}^{(i+1)}\boldsymbol{x}^{(i)} + \boldsymbol{b}^{(i+1)}]_j$$

- Apply non-linear function:

$$\boldsymbol{x}_j^{(i+1)} = \phi(\boldsymbol{z}_j^{(i+1)})$$

A deep neural network (DNN) is an artificial neural network that has many such layers (it is "deep").

It was discovered that making ANNs deep was the key to their success.

As more layers are added, the ANN is capable of performing more and more complex tasks.

At this point, this is a somewhat empirical observation although there are now several research papers that have investigated this question.

DNNs are behind many success stories such as

- AlphaGo,
- Apple's Siri voice recognition, and
- YouTube recommendations.

We will discuss this point in more details later on.

Training and optimization of DNNs

As we have seen previously with the perceptron, based on some training data (x_i, y_i), we can optimize the weights $W^{(i)}$ and biases $b^{(i)}$ to minimize the error.

Before we discuss how to do this, let's start with a simple implementation in Keras/TensorFlow of a DNN.