Neural Network as a Classifier

MUSTAFA HAJIJ

How do we use a neural network as a classifier?

Last time we learned the following:

- The building block of a neural network

How do we use a neural network as a classifier?

Last time we learned the following:

- The building block of a neural network
- How to build a neural network.

How do we use a neural network as a classifier?

Last time we learned the following:

- The building block of a neural network
- How to build a neural network.
- Neural network is essentially a mathematical function $f: R^n \to R^m$.

How do we use a neural network as a classifier?

Last time we learned the following:

- The building block of a neural network
- How to build a neural network.
- Neural network is essentially a mathematical function $f: R^n \to R^m$.
- Given an input x, how to feedforward x through a neural network and obtain an output f(x)

How do we use a neural network as a classifier?

Last time we learned the following:

- The building block of a neural network
- How to build a neural network.
- Neural network is essentially a mathematical function $f: R^n \to R^m$.
- Given an input x, how to feedforward x through a neural network and obtain an output f(x)
- How to train a neural network :
    - Define a cost function
    - For each example in the training set feedforward that example and compute the error
    - Use backpropagation to adjust the weights of the network so that it behaves better with respect to the input example

How do we use a neural network as a classifier?

Last time we learned the following:

- The building block of a neural network
- How to build a neural network.
- Neural network is essentially a mathematical function $f: R^n \to R^m$.
- Given an input x, how to feedforward x through a neural network and obtain an output f(x)
- How to train a neural network :
    - Define a cost function
    - For each example in the training set feedforward that example and compute the error
    - Use backpropagation to adjust the weights of the network so that it behaves better with respect to the input example

Lets recall the feedforward algorithm before first.

Feedforward Neural Network

How do we compute a feedforward neural network on an input x ?

Feedforward Neural Network

Start with an input $x = a^{(0)}$. In the picture, this is represented by the first layer of nodes. We will call this layer 0.

$$x = a^{(0)}$$

Feedforward Neural Network

We apply the weight $W^{(1)}$ coming from the edges between layer 0 and layer 1 and add the biases and then apply the Activation function on the resulting vector coordinate-wise.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})}$$

$W^{(1)}$: Edges between layer 0 and layer 1
$a^{(0)}$: input
$b^{(1)}$: biases applied to layer 1
σ : activation function

Feedforward Neural Network

We will call the output of this computation $a^{(1)}$. This is now represented by the nodes in layer 1.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}}$$

$W^{(1)}$: Edges between layer 0 and layer 1
$a^{(0)}$: input
$b^{(1)}$: biases applied to layer 1
σ : activation function

Feedforward Neural Network

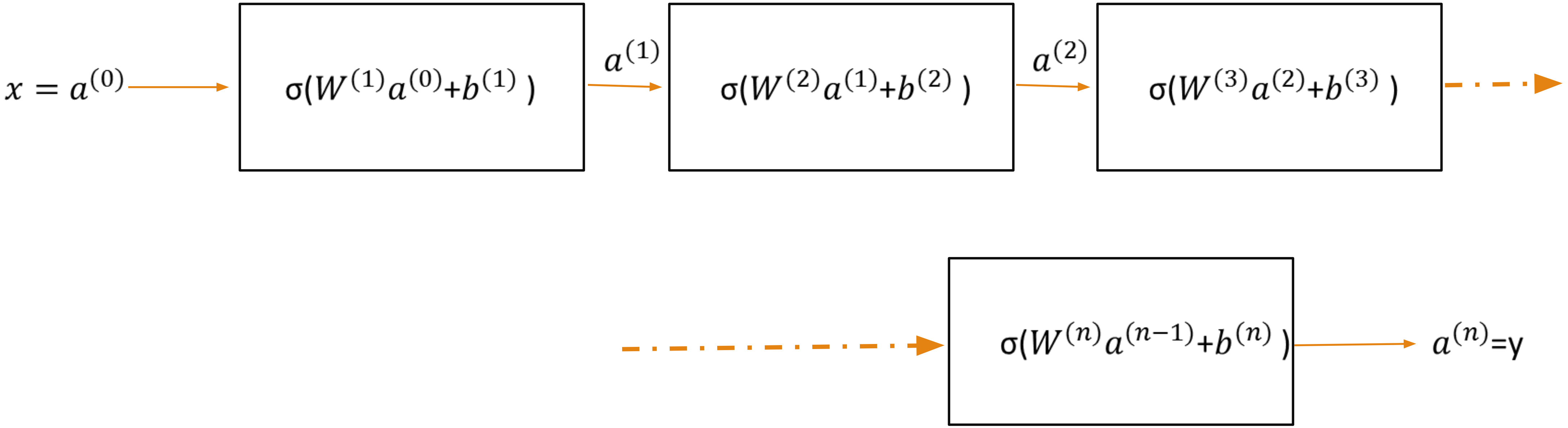
Repeat.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}} \boxed{\sigma(W^{(2)}a^{(1)}+b^{(2)})} \xrightarrow{a^{(2)}}$$

$W^{(2)}$: Edges between layer 1 and layer 2
$a^{(1)}$: input from layer 1
$b^{(2)}$: biases applied to layer 2
σ : activation function

Feedforward Neural Network

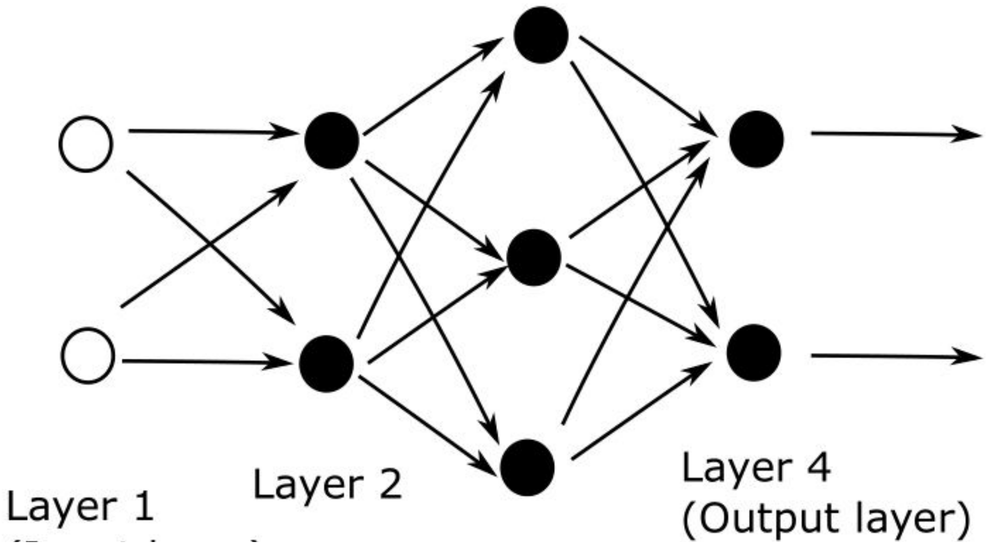
Until you finish the neural network and get the final output.



Example

We will use an example from <u>this</u> paper.

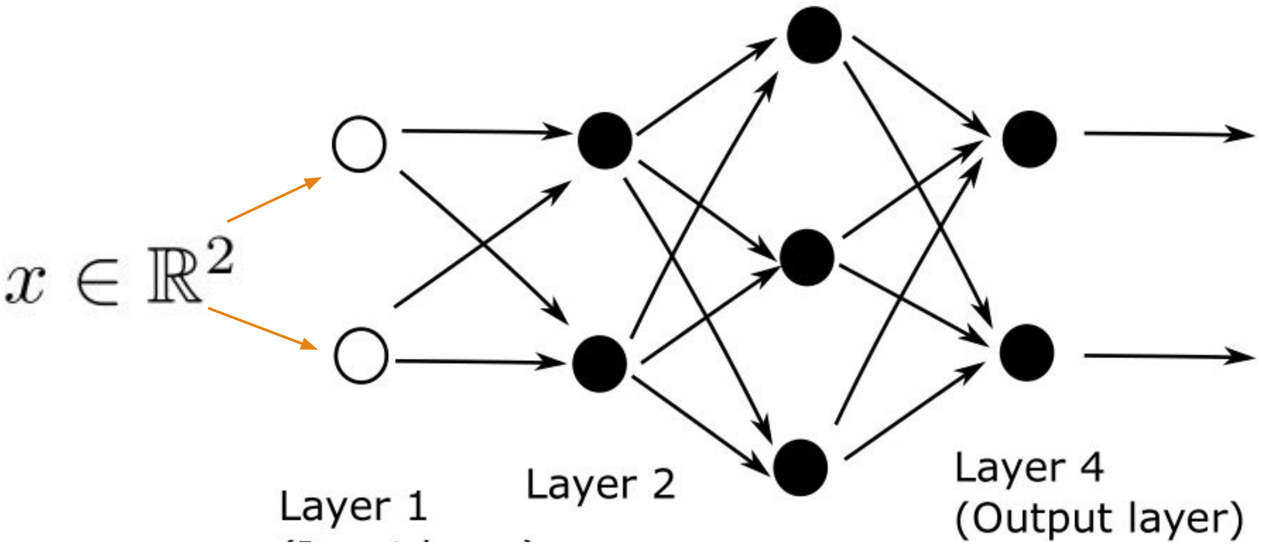
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

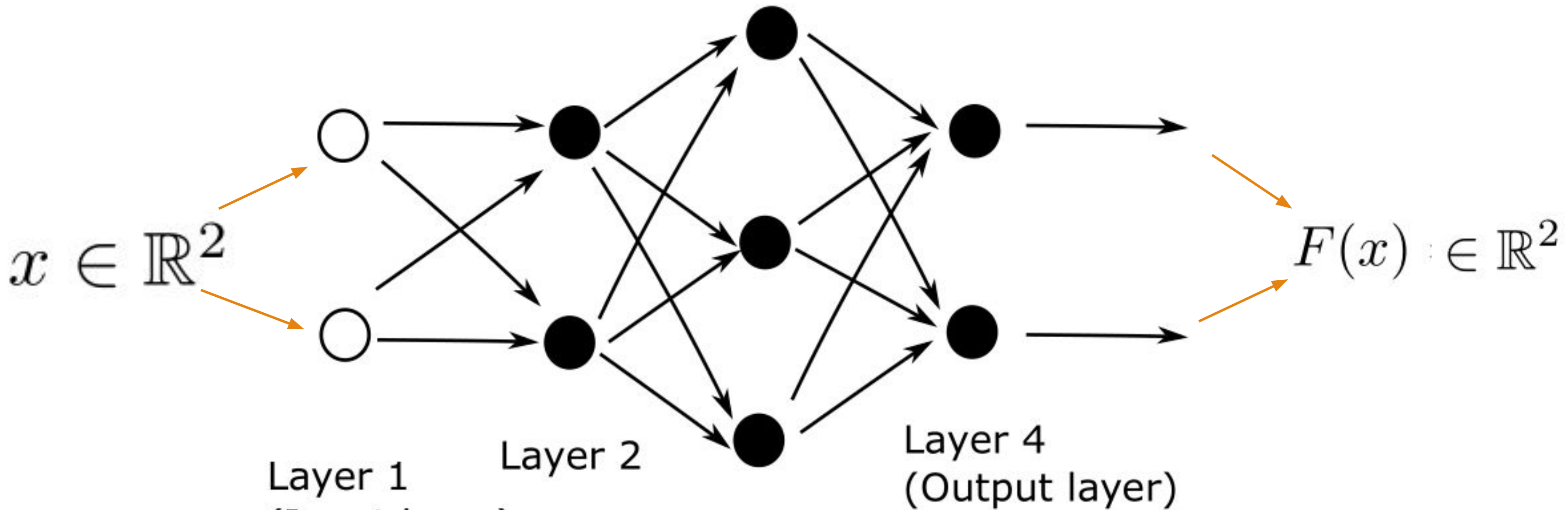
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

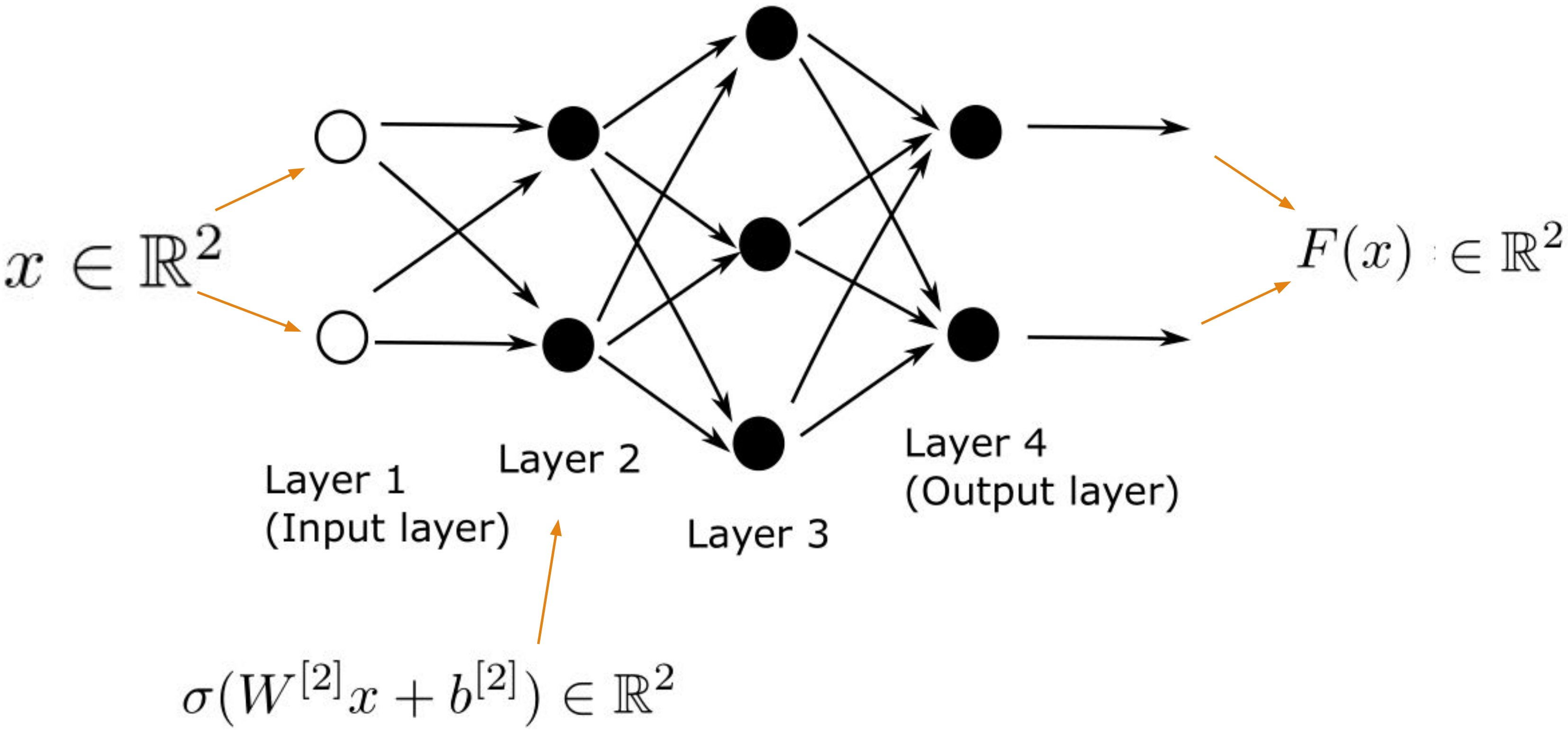
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

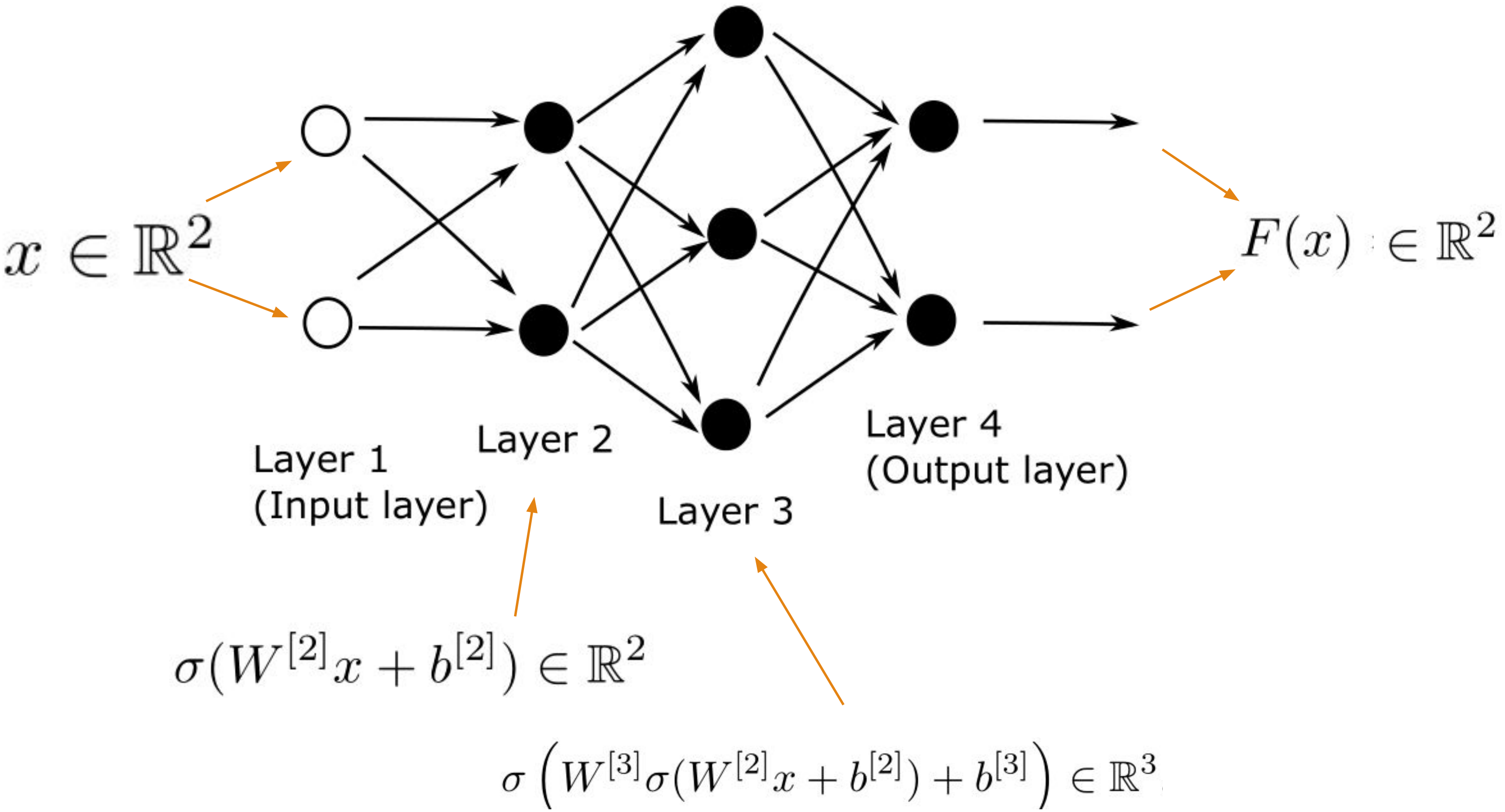
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

(note that the convention of the index is a little different here)

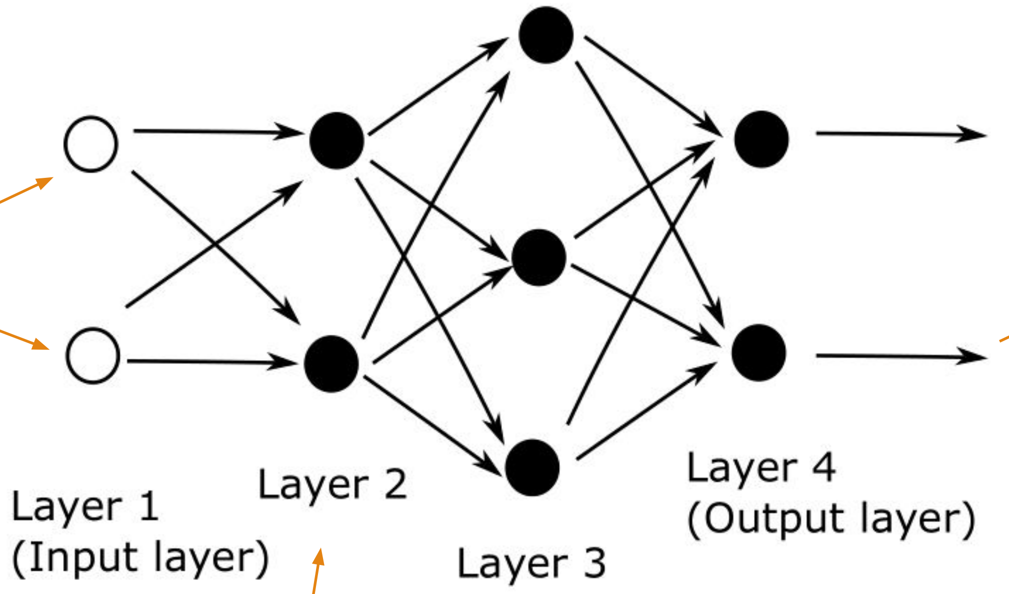


Example

We will use an example from this paper.

(note that the convention of the index is a little different here)

$x \in \mathbb{R}^2$



$F(x) \in \mathbb{R}^2$

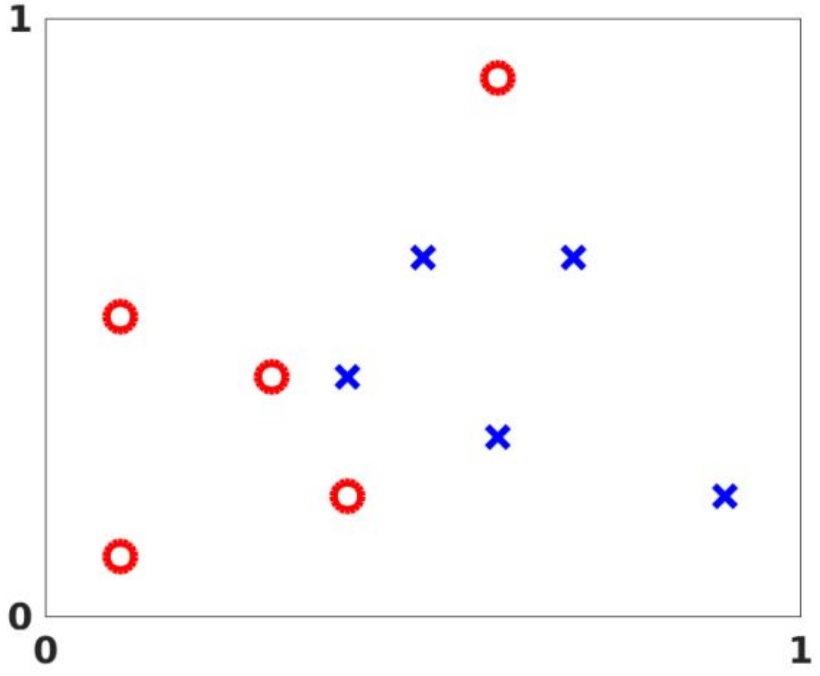
$\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2$

$\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) \in \mathbb{R}^3$

Final function representing the neural network

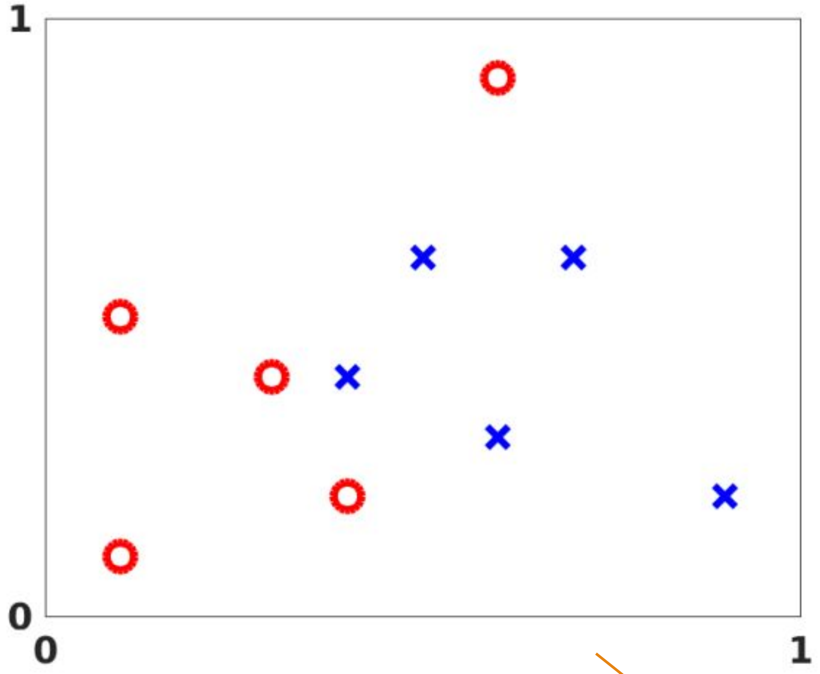
$$F(x) = \sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) + b^{[4]}\right) \in \mathbb{R}^2.$$

Example



Input : labeled data X

Example

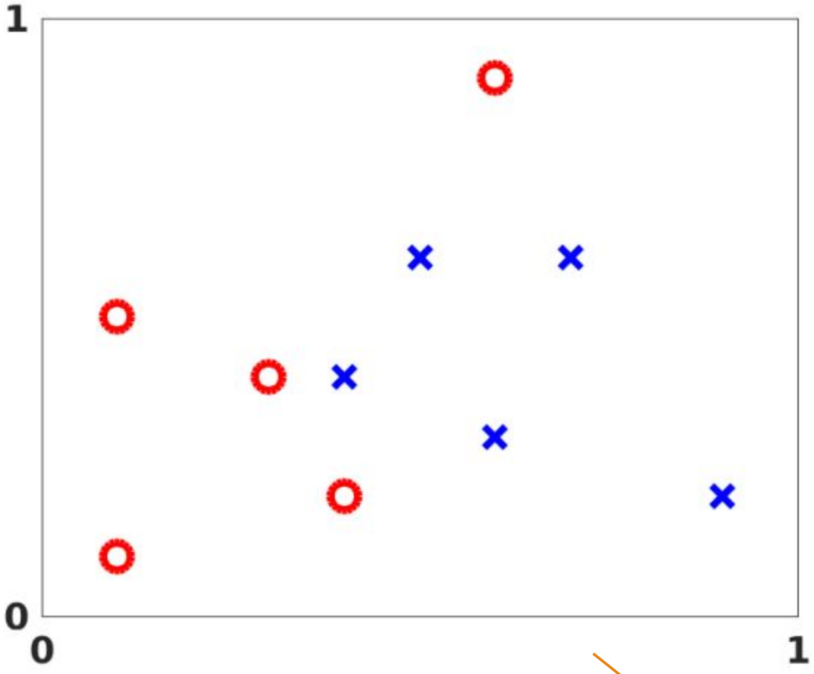


Input : labeled data X

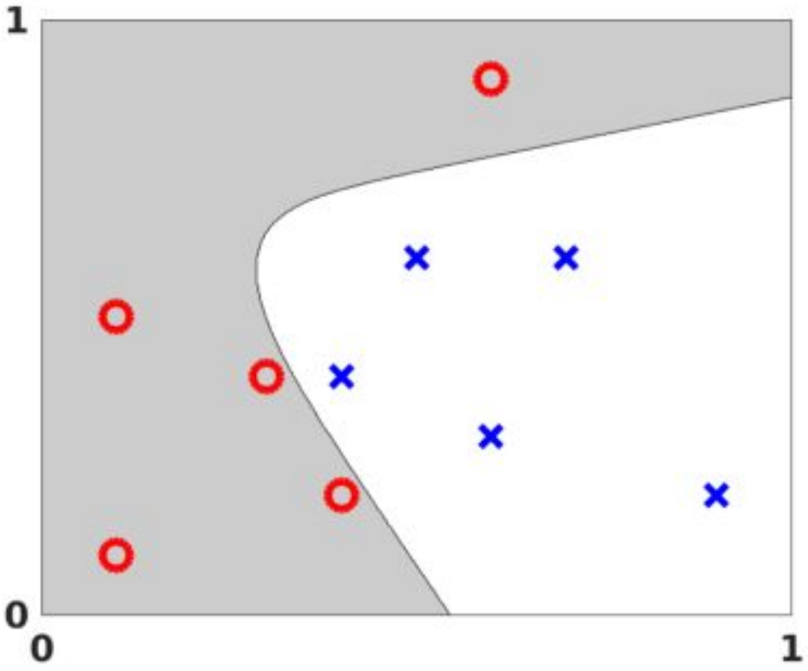
$$\mathrm{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \tfrac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2.$$

the difference between the output given by the network and the actual label

Example



Input : labeled data X



Minimize the cost function

$$\mathrm{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \frac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2.$$

the difference between the output given by the network and the actual label

Binary classification

Now suppose that we have data set that consists of images of cats and dogs and we built a neural network that takes as input an image from this data set and gives out a vector in R^1 (a real number).
How exactly do we use this vector for our classification task ? In general the output f(x) coming from the neural network
Does not match the class $\{\pm 1\}$ of the input point x (it could be any real number).


Binary classification

This function takes a tensor of size input_size and returns a real number.

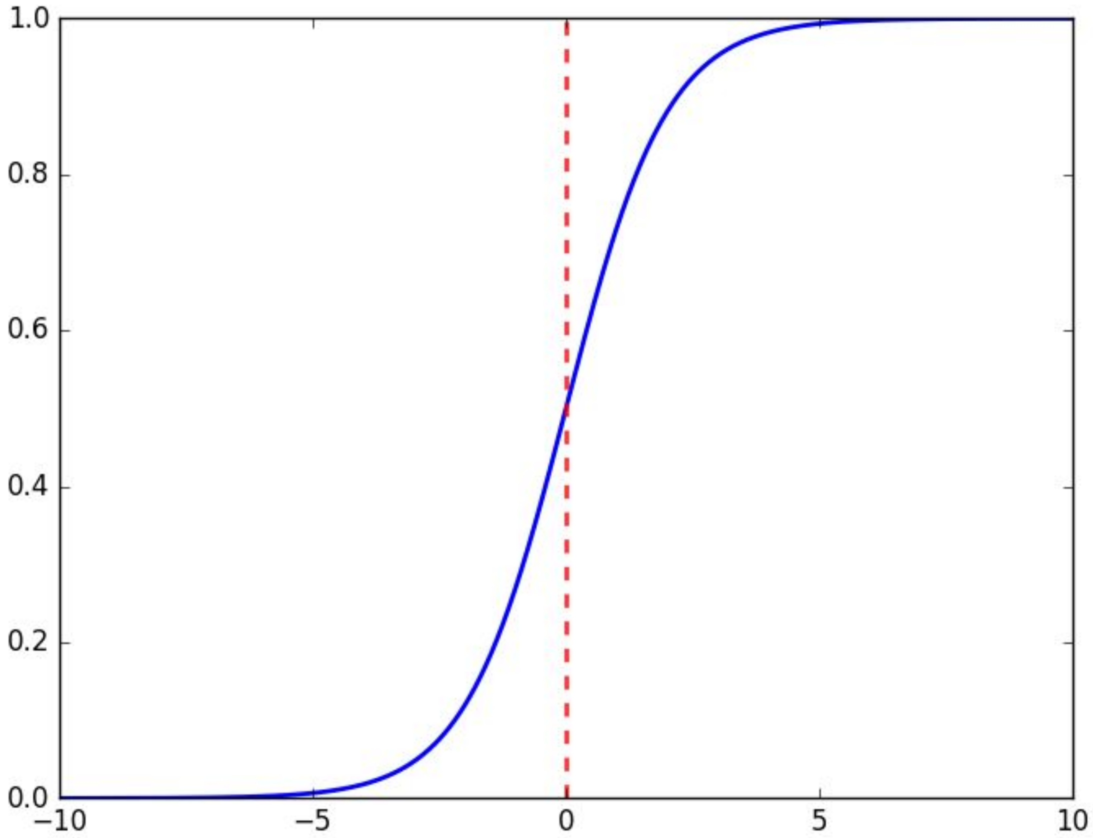
```
import torch
import torch.nn as nn

class Net(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

How can we constrain the output to be between -1 and +1?

Binary classification

To obtain the required binary classification, we pass the output f(x) through another function :

$$g(z) = 1/(1 + e^{-z})$$



The graph of the sigmoid function

Binary classification

To obtain the required binary classification, we pass the output f(x) through another function :

$$g(z) = 1/(1 + e^{-z})$$

This function returns an output between 0 and 1. The binary classification is set as follows :

If ($g(z) \geq 0.5$) assign the input the positive class
Else assign the input to the negative class

Binary classification

To obtain the required binary classification, we pass the output f(x) through another function :

$$g(z) = 1/(1+e^{-z})$$

This function returns an output between 0 and 1. The binary classification is set as follows :

If ($g(z) \geq 0.5$) assign the input the positive class
Else assign the input to the negative class

But what do we do in the multi-class classification ?

Multi-class classification : the softmax function

In the case of multi-class classification, we use the softmax activation function.
Suppose that we have k classes then the softmax activation function is define by :

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

Here z_i represents the ith element of the input to softmax, which corresponds to class i.

Multi-class classification : the softmax function

In the case of multi-class classification, we use the softmax activation function.
Suppose that we have k classes then the softmax activation function is define by :

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

Here z_i represents the ith element of the input to softmax, which corresponds to class i.
The result is a vector containing the probabilities that sample x belong to each class.

Multi-class classification : the softmax function

In the case of multi-class classification, we use the softmax activation function.
Suppose that we have k classes then the softmax activation function is define by :

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

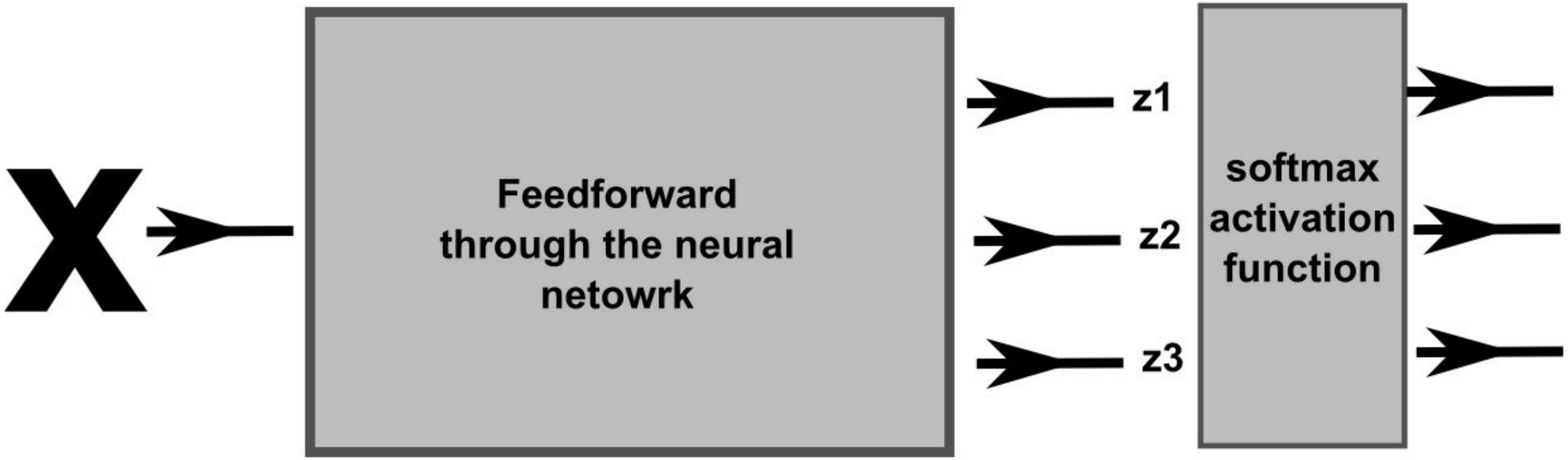
Here z_i represents the ith element of the input to softmax, which corresponds to class i.
The result is a vector containing the probabilities that sample x belong to each class.
The output is the class with the highest probability.

Multi-class classification : the softmax function

In the case of multi-class classification, we use the softmax activation function.
Suppose that we have k classes then the softmax activation function is define by :

$$\operatorname{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

Here z_i represents the ith element of the input to softmax, which corresponds to class i.
The result is a vector containing the probabilities that sample x belong to each class.
The output is the class with the highest probability.



The softmax function in Python

The softmax function is a mathematical function used to convert a vector of real numbers into a probability distribution.

It takes an input vector and returns another vector of the same length, where each element is transformed to a value between 0 and 1, representing the probability of that element being selected. In simple terms, the softmax function normalizes the input vector and makes it easier to interpret as probabilities. Here's a Python example:

```
import numpy as np

def softmax(x):
    exp_values = np.exp(x)
    probabilities = exp_values / np.sum(exp_values)
    return probabilities

input_vector = np.array([2.0, 1.0, 0.5])
output_vector = softmax(input_vector)
print(output_vector)
[0.62842832 0.2312239  0.14034778]
```