Neural Networks in Sklearn

MUSTAFA HAJIJ

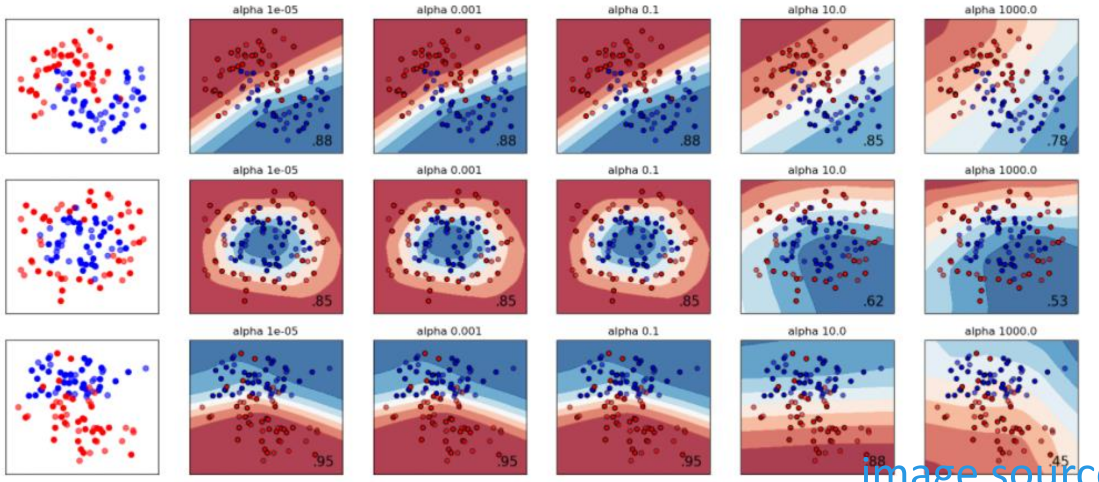


image source

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 $f: R^n \rightarrow R^m$.

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 $f: R^n \rightarrow R^m$.
- 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

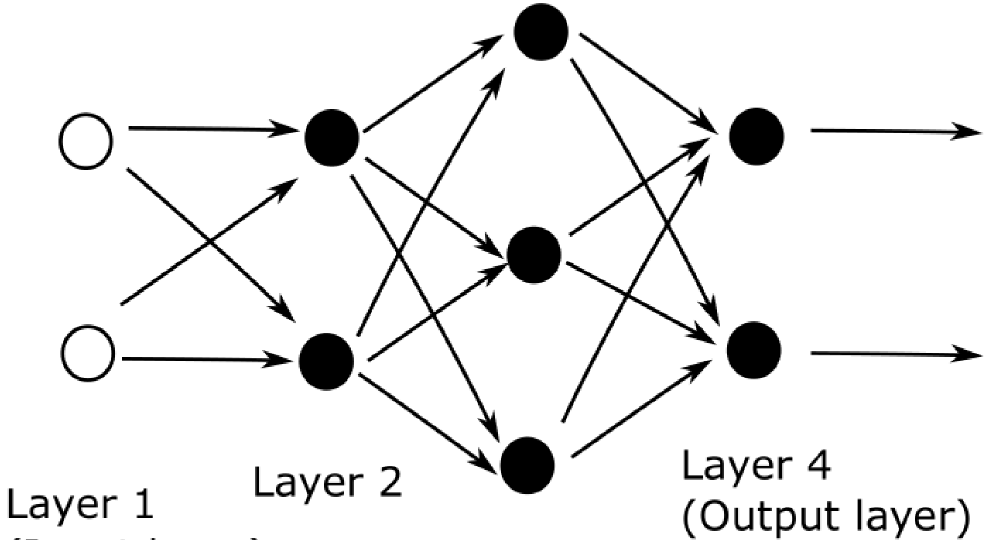
Until you finish the neural network and get the final output.

$x = a^{(0)} \rightarrow$ $\sigma(W^{(1)}a^{(0)}+b^{(1)})$ $\xrightarrow{a^{(1)}}$ $\sigma(W^{(2)}a^{(1)}+b^{(2)})$ $\xrightarrow{a^{(2)}}$ $\sigma(W^{(3)}a^{(2)}+b^{(3)})$ $\dashrightarrow$

$\dashrightarrow$ $\sigma(W^{(n)}a^{(n-1)}+b^{(n)})$ $\rightarrow$ $a^{(n)}$=y

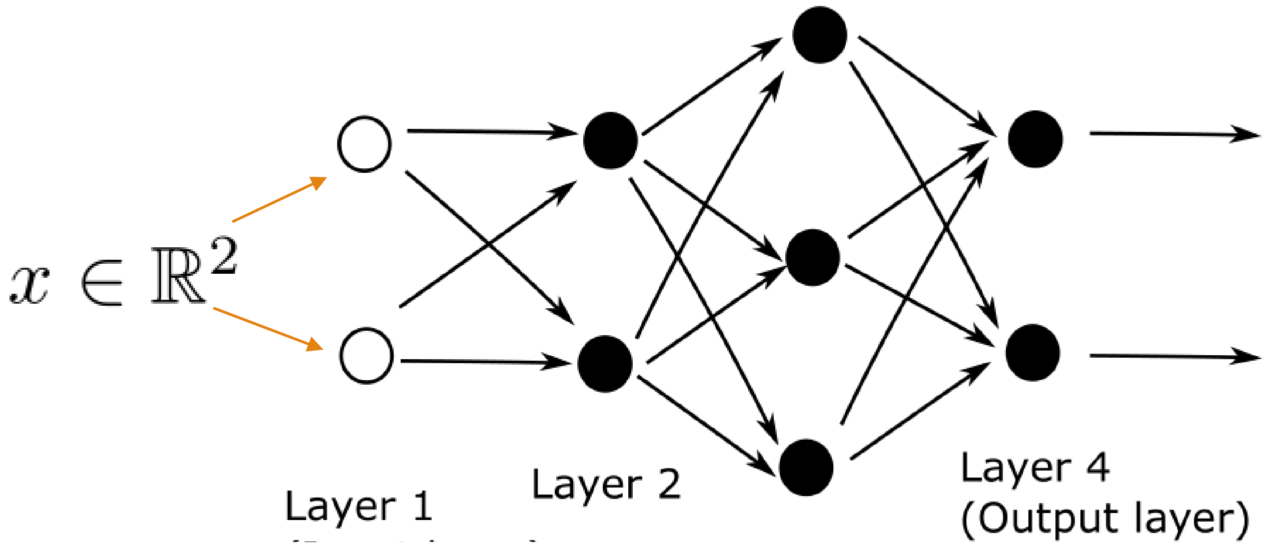
Example

We will use an example from this paper. (note that the convention of the index is a little different here)



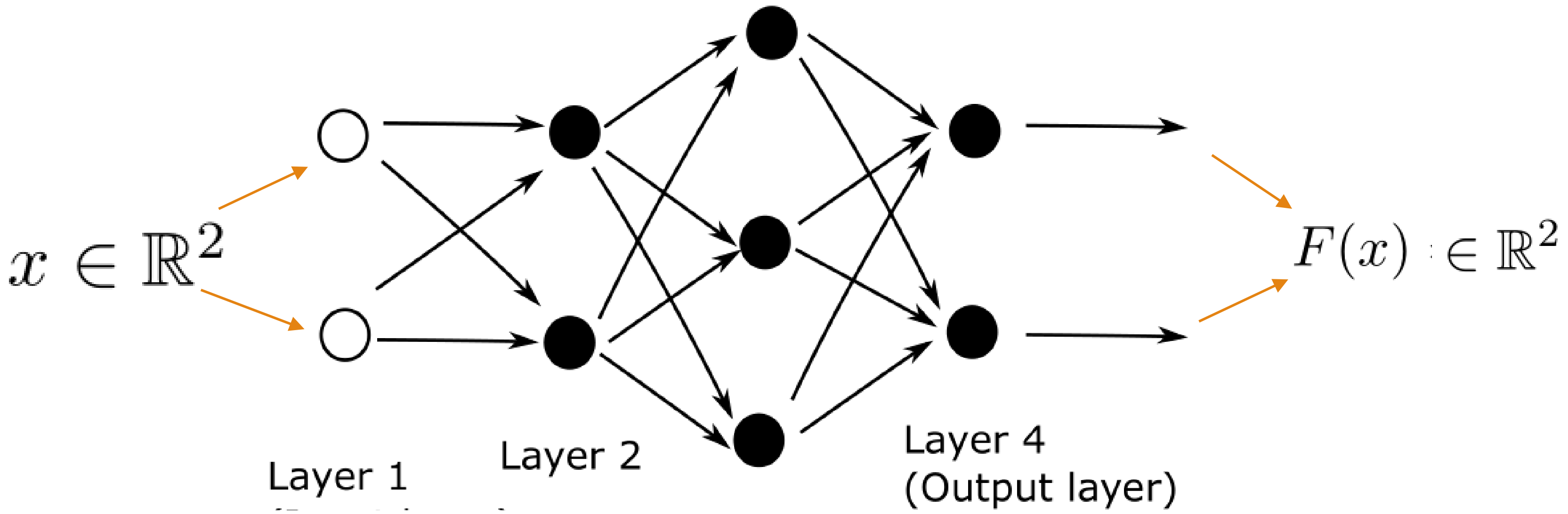
Example

We will use an example from this paper. (note that the convention of the index is a little different here)



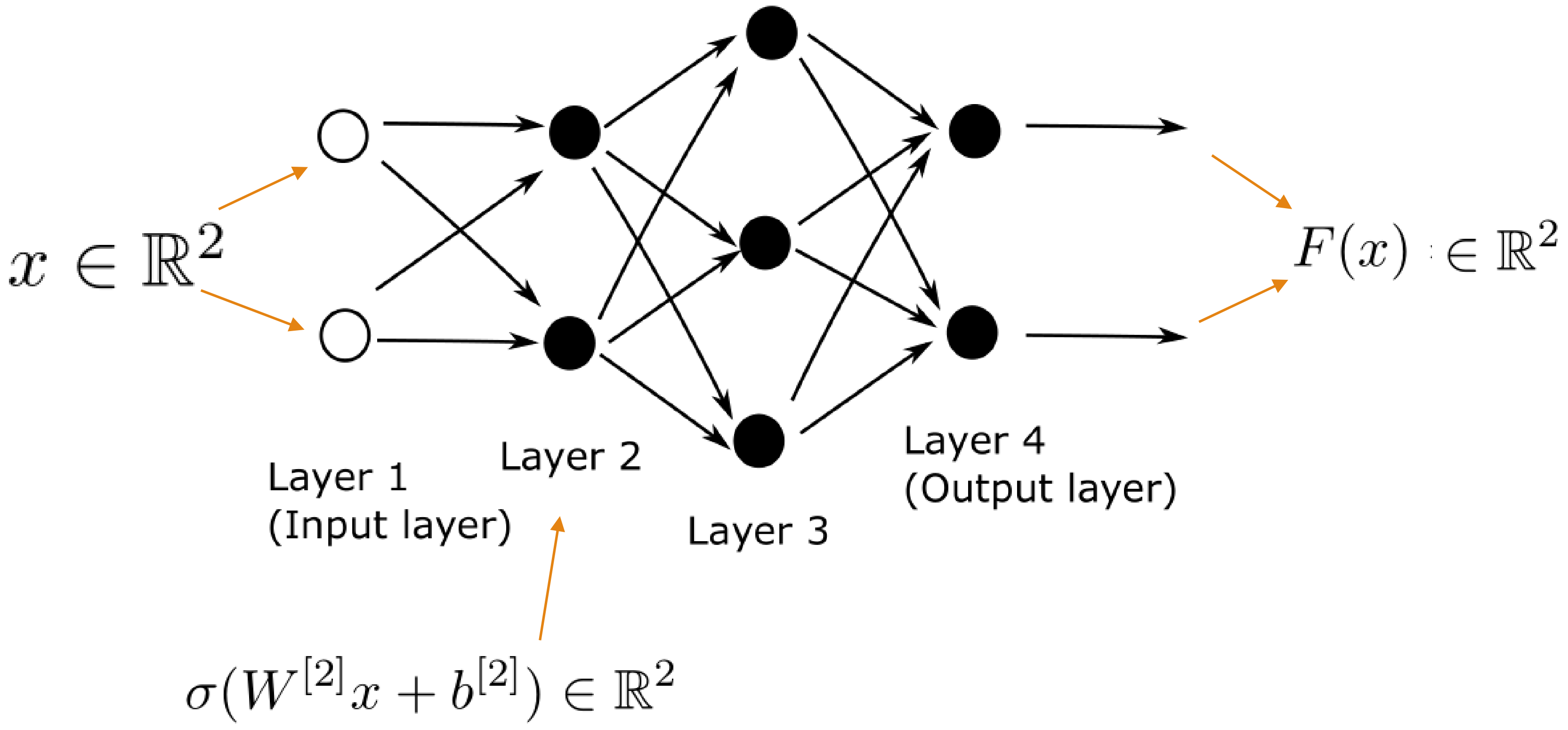
Example

We will use an example from this paper. (note that the convention of the index is a little different here)



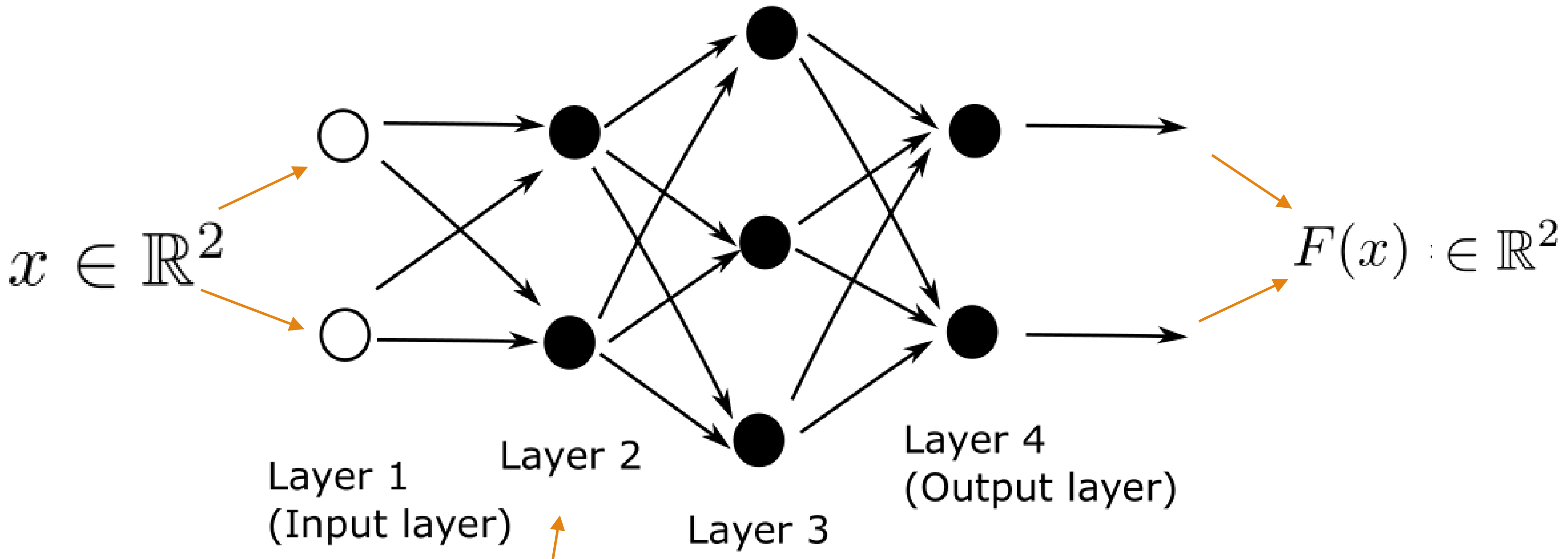
Example

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)

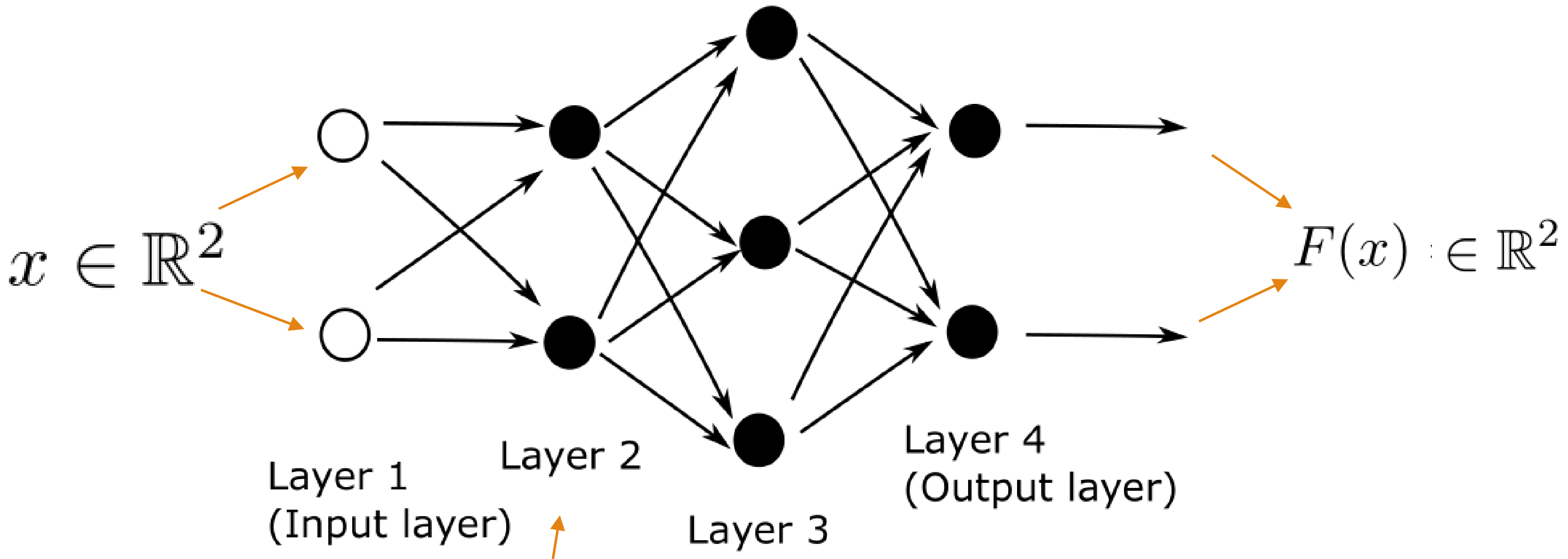


$$\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2$$

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

Example

We will use an example from this paper. (note that the convention of the index is a little different here)



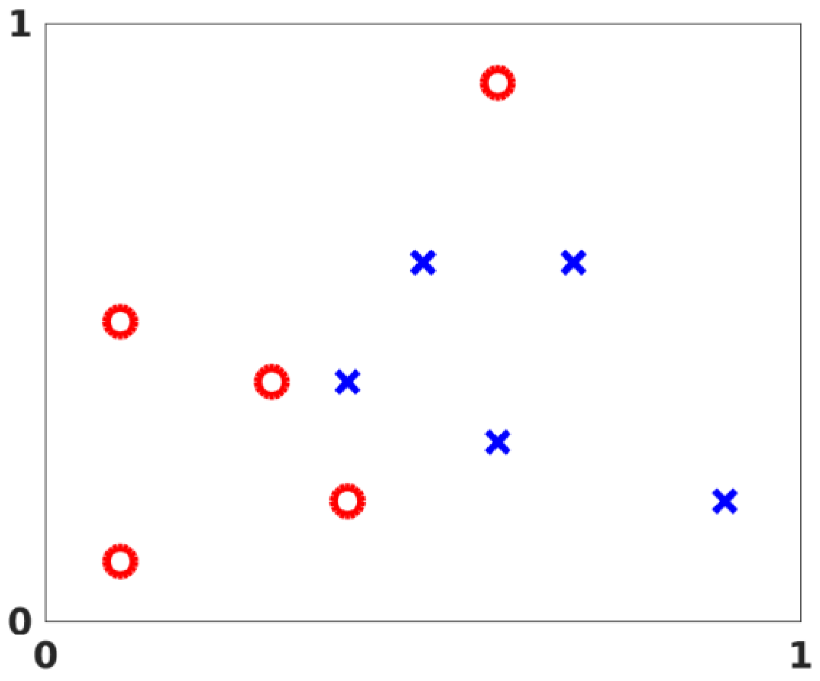
$$\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2$$

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

Final function representing the neural network

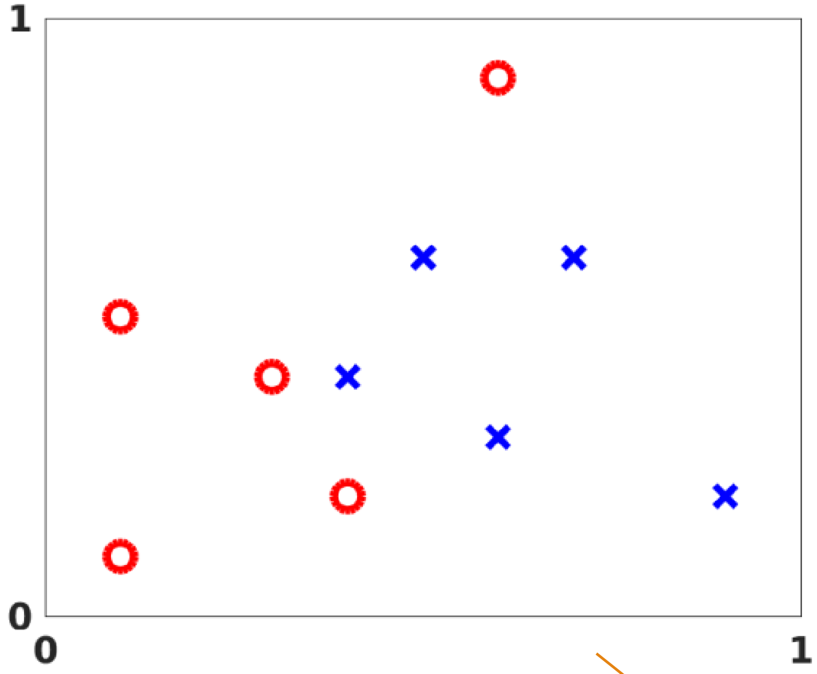
$$F(x) = \sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) + b^{[4]}\right) \in \mathbb{R}^2.$$

Example



Input : labeled data X

Example

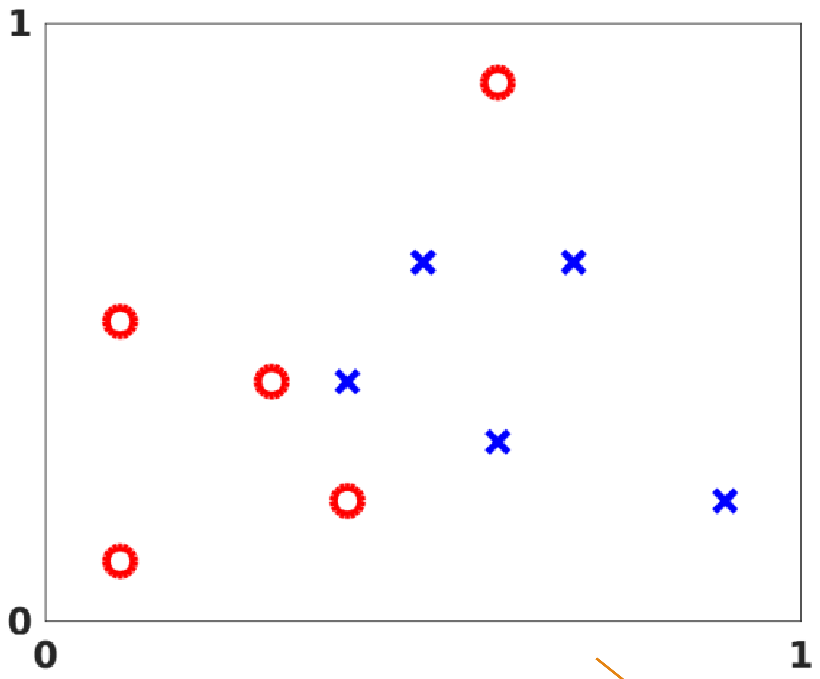


Input : labeled data X

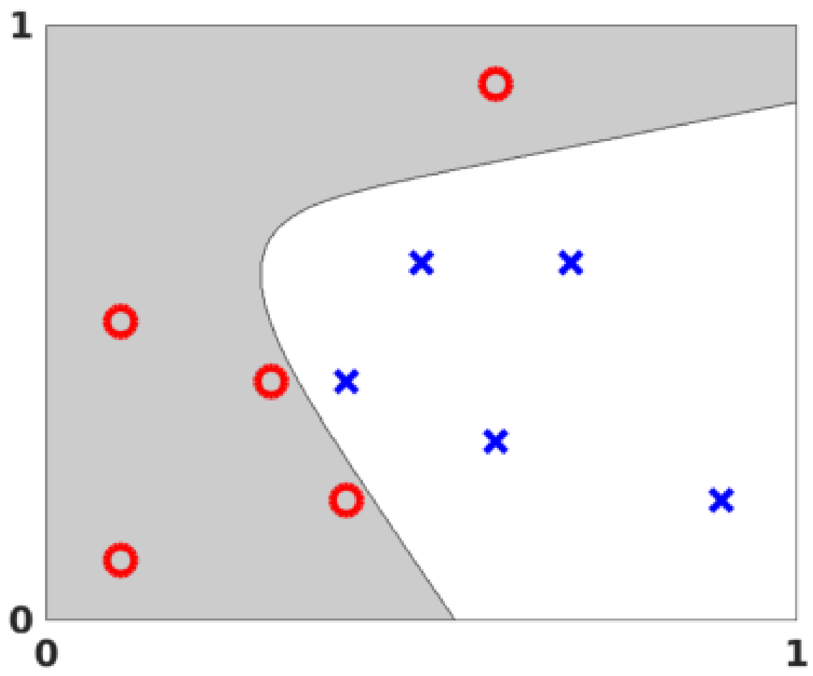
$$\mathrm{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \tfrac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2.$$

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

Example



Input : labeled data X



Minimize the cost function

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

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

How do we use a neural network as a classifier?

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).

How do we a neural network as a classifier?

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

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

How do we a neural network as a classifier?

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

How do we a neural network as a classifier?

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 ?

How do we a neural network as a classifier?

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 :

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

How do we a neural network as a classifier?

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.

How do we a neural network as a classifier?

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.
The result is a vector containing the probabilities that sample x belong to each class.

How do we a neural network as a classifier?

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 :

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

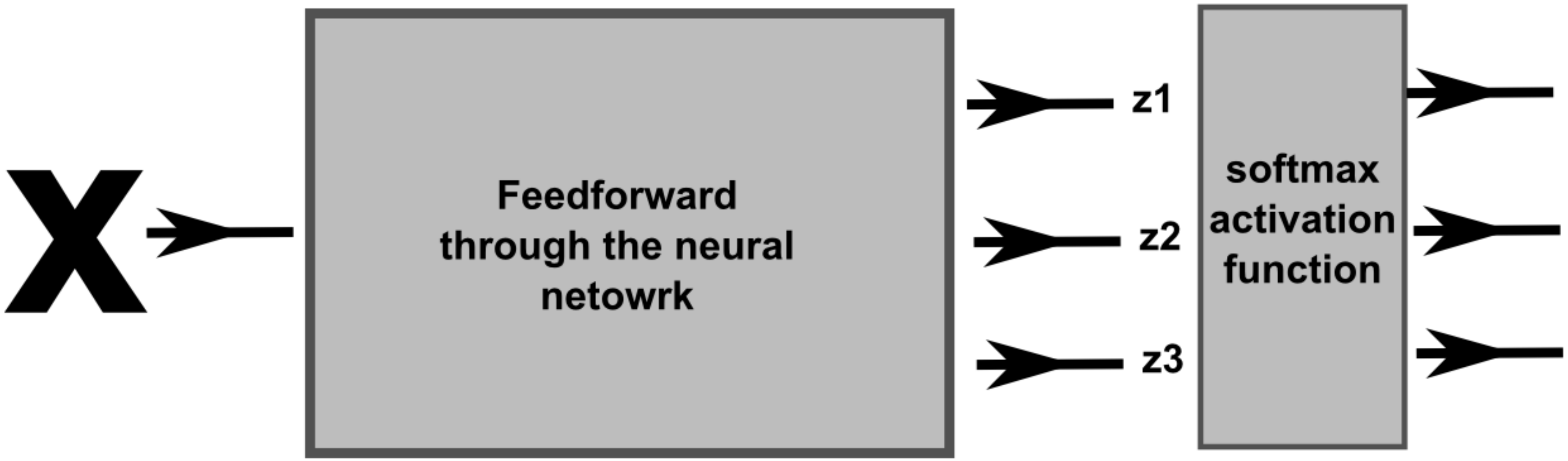
Here z_i represents the ith element of the input to softmax, which corresponds to class i.
The result is a vector containing the probabilities that sample x belong to each class.
The output is the class with the highest probability.

How do we a neural network as a classifier?

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 :

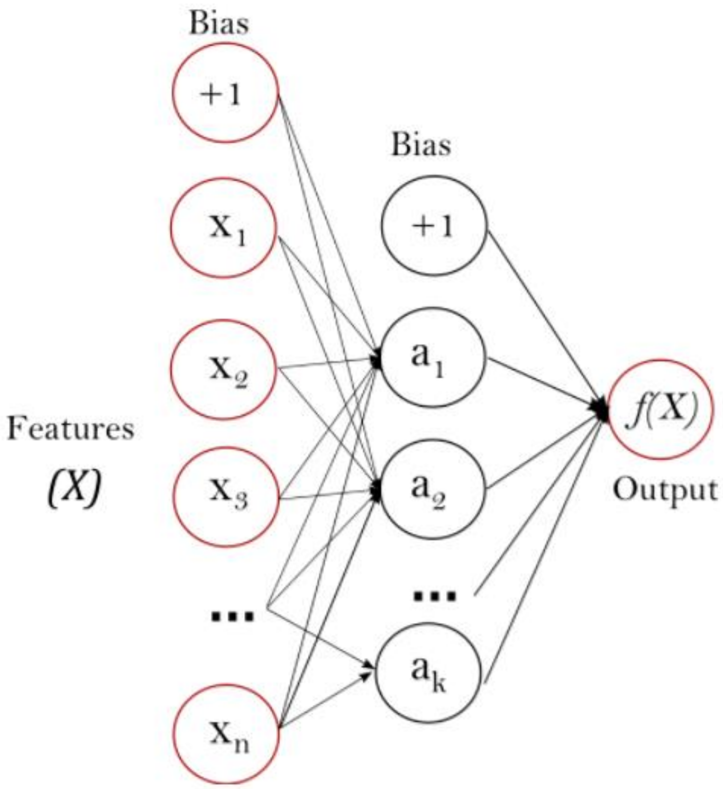
$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

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.



Neural networks in sklearn

In sklearn we can train Multi-layer Perceptron (MLP) for either classification or regression.



Neural networks as a classifier-example

Recall that a neural network is essentially a mathematical function $f: R^n \to R^m$. The number n is the number of features in the input (say number of pixels in the image) or this is the number of nodes in the input layer.

The number m is the number of node in the output. For example in the case of the digits classifier $m = 10$.

Let us study this example

Neural networks as a classifier-example

Import the libraries that we need

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
```

Neural networks as a classifier-example

Import the libraries that we need

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
```

Set the data, rescale, split into training and testing datasets.

```
mnist = fetch_mldata("MNIST original")
# rescale the data, use the traditional train/test split
X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Neural networks as a classifier-example

Import the libraries that we need

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
```

Set the data, rescale, split into training and testing datasets.

```
mnist = fetch_mldata("MNIST original")
# rescale the data, use the traditional train/test split
X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Define the neural network, fit the data. This neural network has only one hidden unit with 50 units.

```
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

mlp.fit(X_train, y_train)
```

Neural networks as a classifier-example

Import the libraries that we need

```
fig, axes = plt.subplots(4, 4)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin,
               vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

The matrix : mlp.coefs_[0] represents the matrix $W^{(1)}$ in our earlier notation. The node above visualizes the columns of this matrix as images.

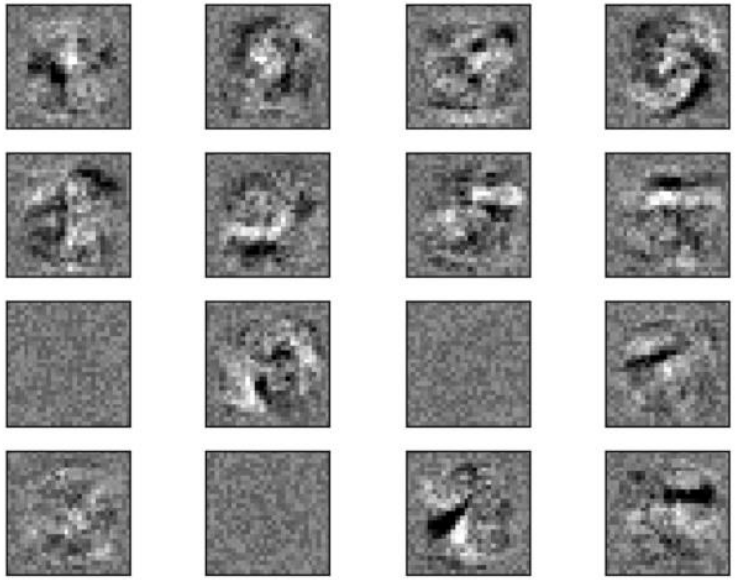
Neural networks as a classifier-example

Import the libraries that we need

```
fig, axes = plt.subplots(4, 4)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin,
               vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

The matrix : mlp.coefs_[0] represents the matrix $W^{(1)}$ in our earlier notation. The node above visualizes the columns of this matrix as images.

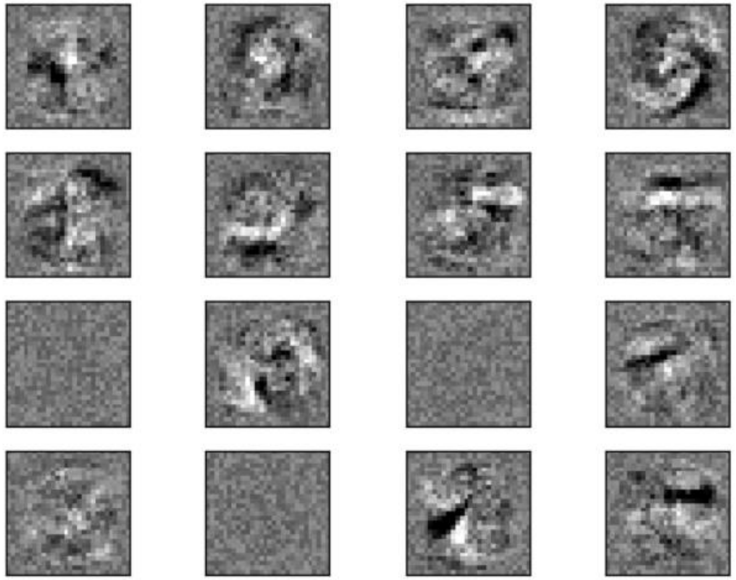


Neural networks as a classifier-example

Import the libraries that we need

```
fig, axes = plt.subplots(4, 4)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin,
               vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```



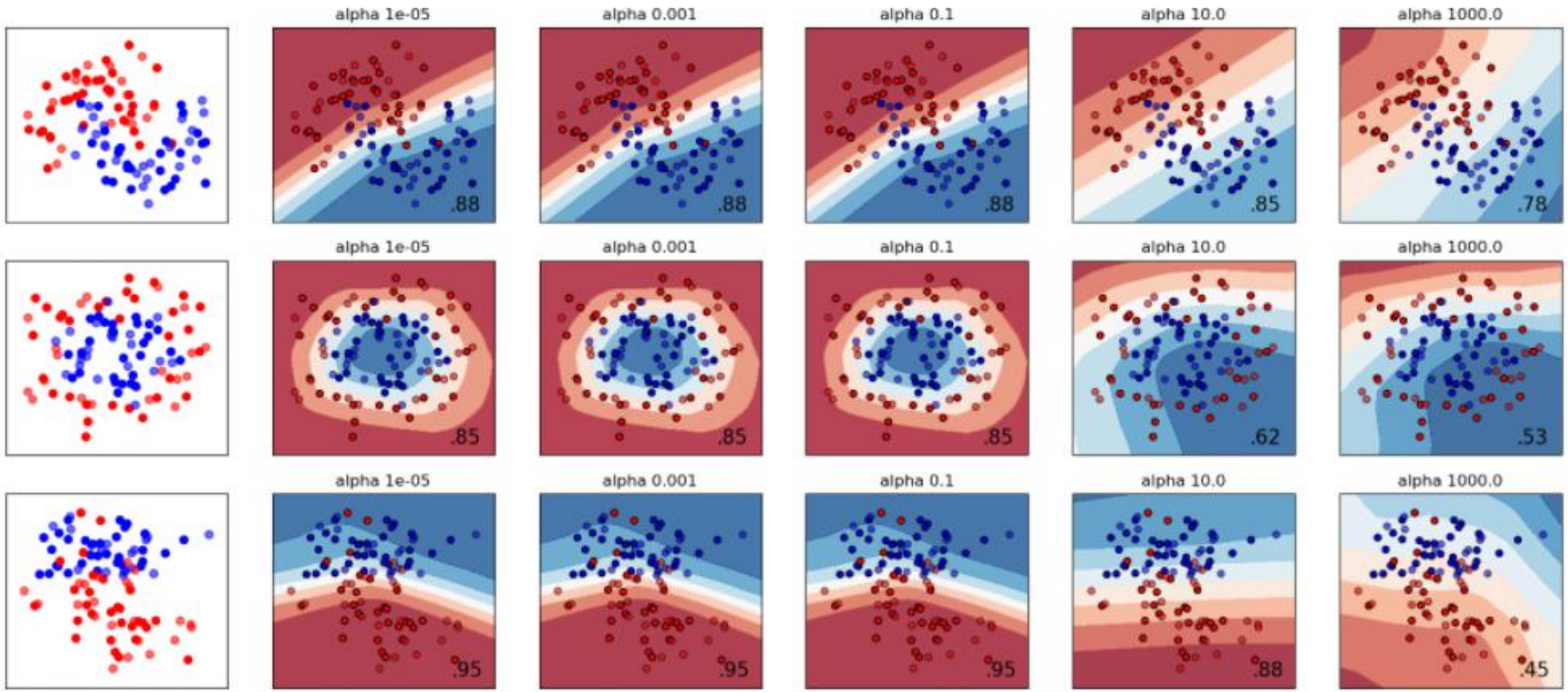
The matrix : mlp.coefs_[0] represents the matrix $W^{(1)}$ in our earlier notation. The node above visualizes the columns of this matrix as images.

Question : what is the shape of the matrix mlp.coefs_[0] ?

If you do know the answer watch this again

Regularization

Alpha is a parameter for regularization term, aka penalty term, that combats overfitting by constraining the size of the weights. Increasing alpha may fix high variance (a sign of overfitting) by encouraging smaller weights, resulting in a decision boundary plot that appears with lesser curvatures. Similarly, decreasing alpha may fix high bias (a sign of underfitting) by encouraging larger weights, potentially resulting in a more complicated decision boundary. Source



sklearn example