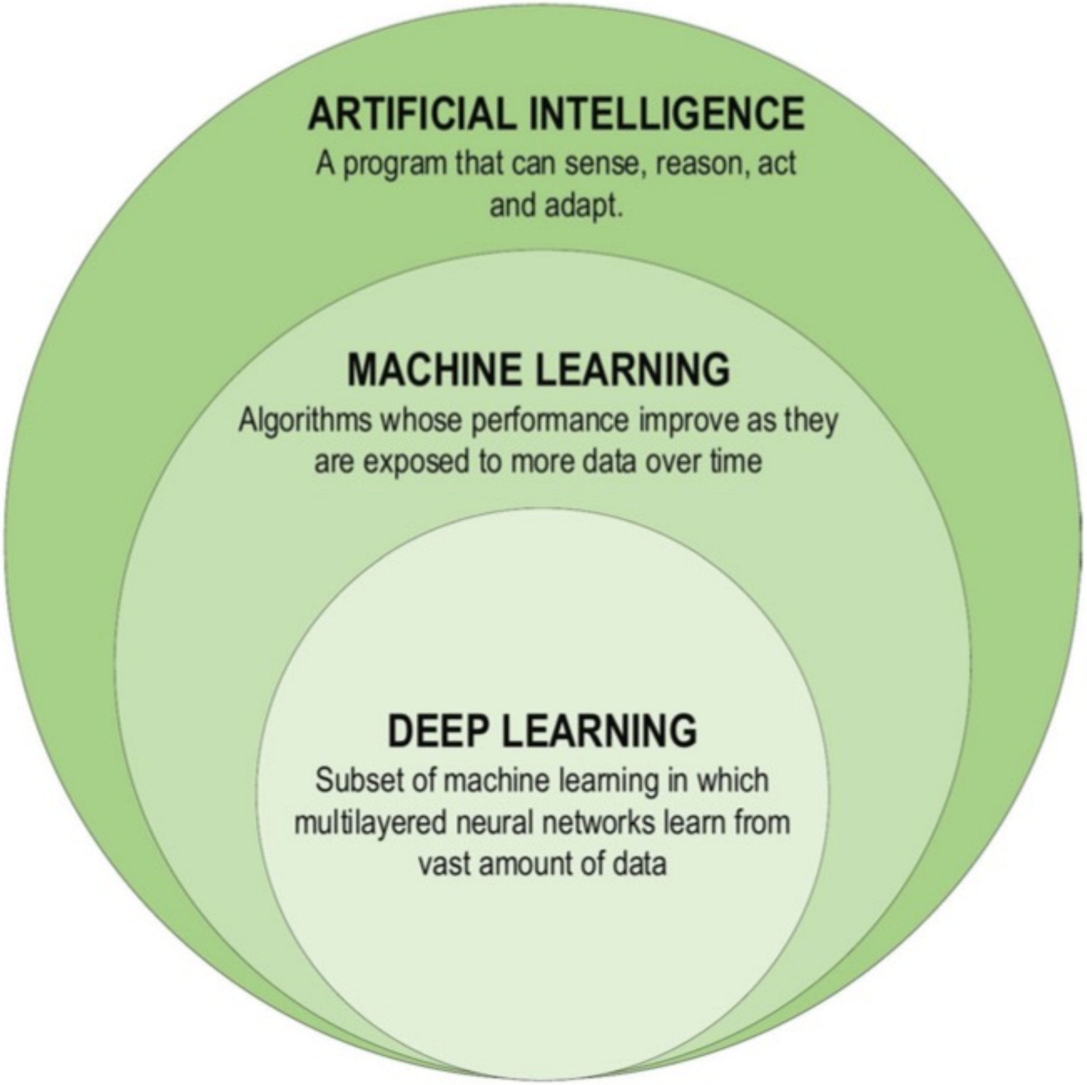
# NEURAL NETWORKS

# 1.1 Introduction

.



ARTIFICIAL INTELLIGENCE

A program that can sense, reason, act and adapt.

MACHINE LEARNING

Algorithms whose performance improve as they are exposed to more data over time

DEEP LEARNING

Subset of machine learning in which multilayered neural networks learn from vast amount of data

# NN Architectures

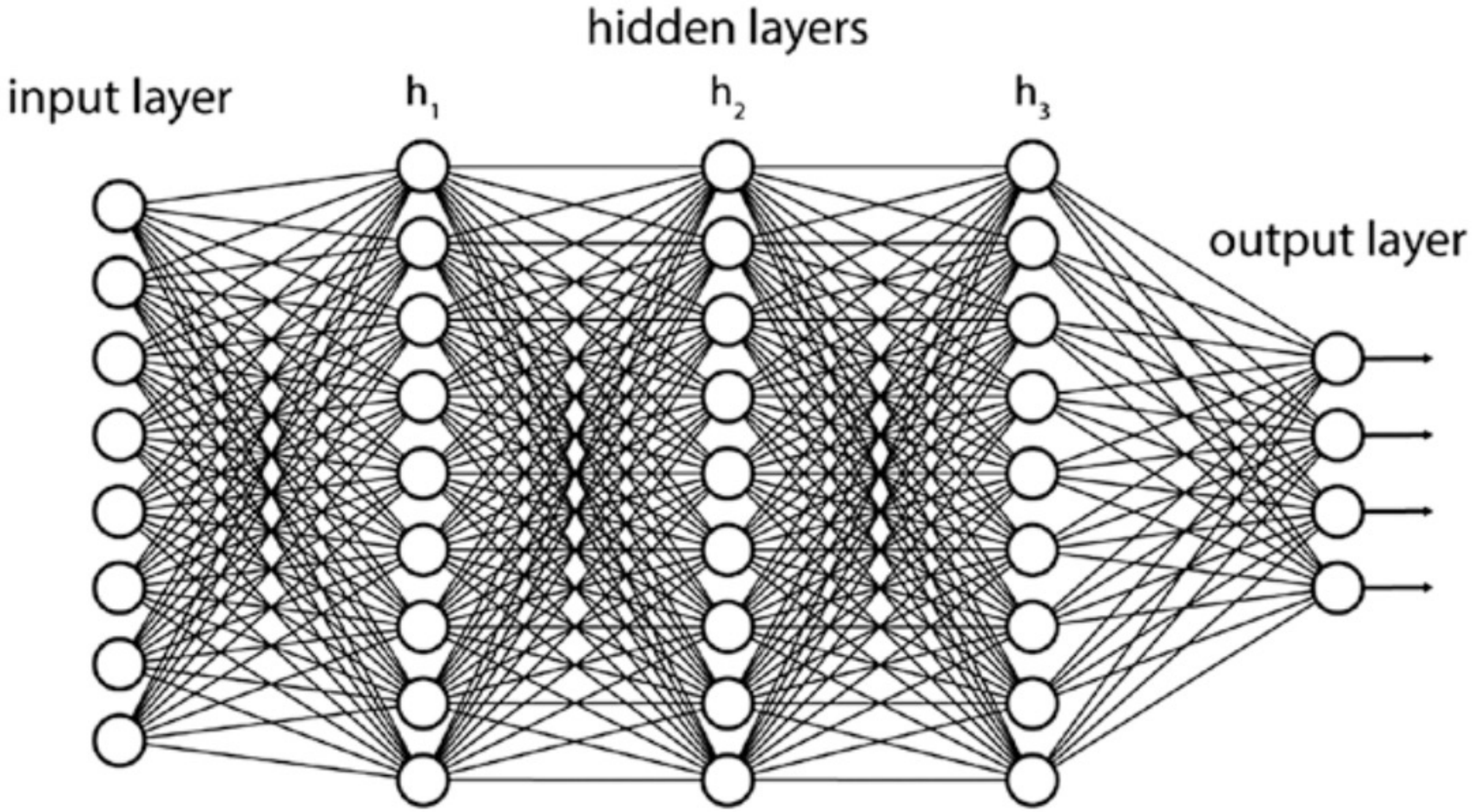
# Neural Network Architectures

- The way the NN nodes connect allowing for the input data to be transformed into new meaningful representation of the data defines the NN architecture.
- The most simple NN architecture is the

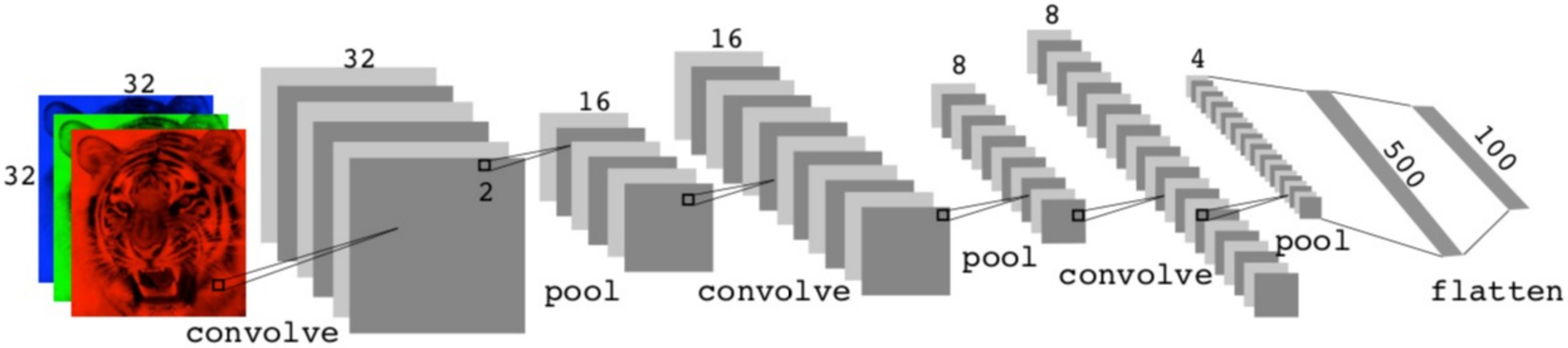
# Neural Network Architectures

- Densely connected neural network (MLP)
- Convolutional Neural Networks (convnets, CNN)
- Recurrent Neural Networks (RNN)
- Long-Short Term Memory Networks (LSTM)
- Transformer Neural Networks
- Generative AI Networks

# MLP (Sequence of connected layers)

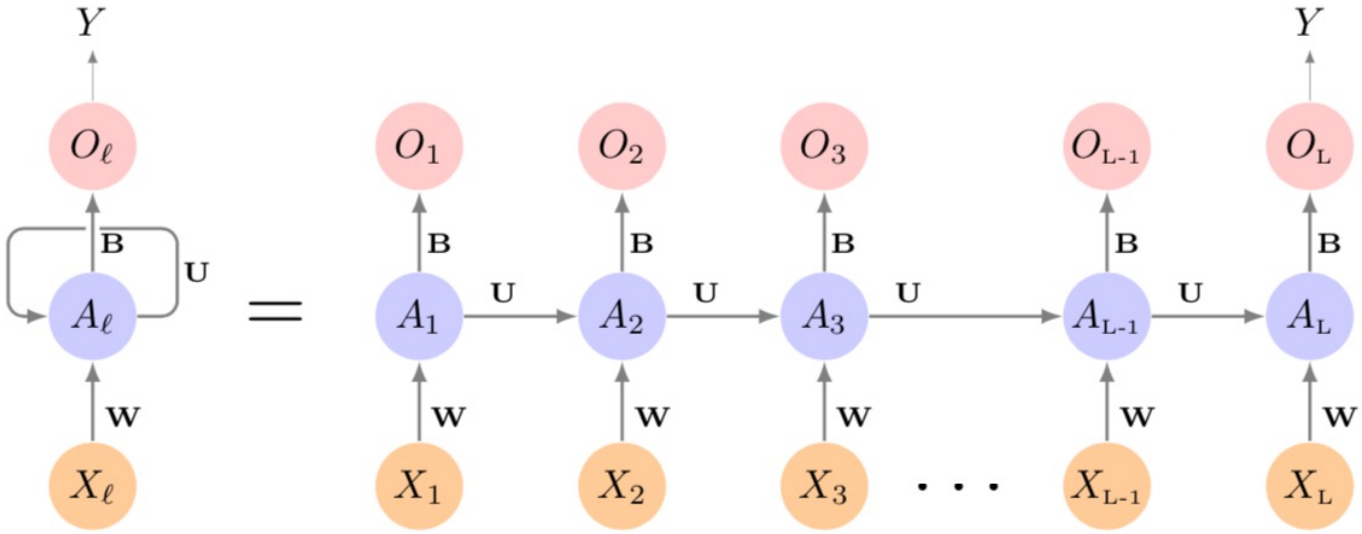


# Deep CNN model for the CIFAR100 image classification



Convolution layers are interspersed with 2 × 2 max-pool layers, which reduce the size by a factor of 2 in both dimensions.

# Simple Recurrent Neural Network (RNN)



output layer

hidden layer

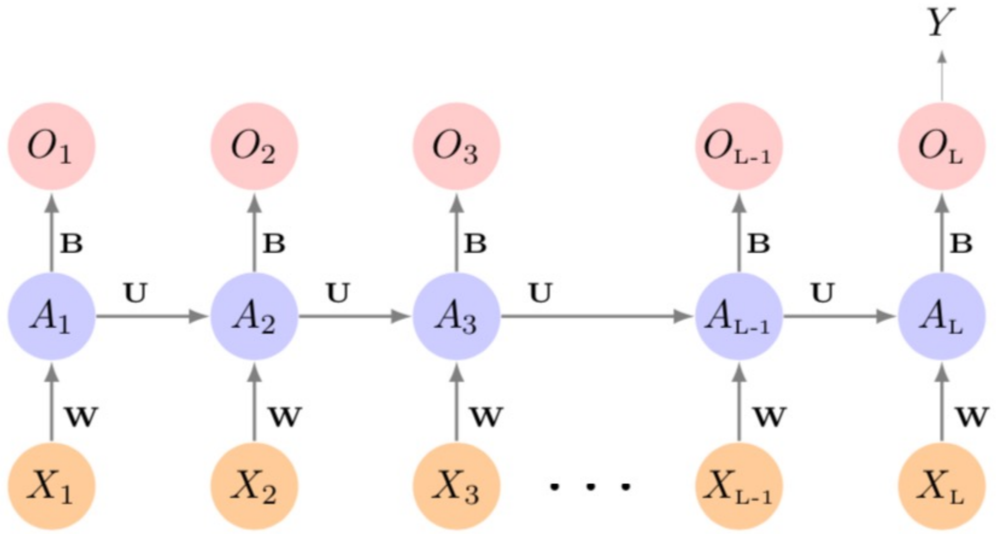
input layer

short representation
of the network

The network is unrolled
into a more explicit version

- The input is a sequence of vectors $X_1,..,X_L$ the target is a single response Y
- The weights W, U and B are estimated as the sequence is processed

# Simple Recurrent Neural Network (RNN)



output layer

hidden layer

input layer

- The NN processes the input sequence X sequentially
- Each $X_i$ feeds into the hidden layer, which has as input the activation vector $A_i$ from the previous element in the sequence producing the current activation vector $A_i$
- The output layer produces a sequence of predictions $O_i$ from the current activation $A_i$, but typically only the last of these, $O_L$, is of relevance

# Introduction

- sklearn is used for MLP (not for other DL architecture)
- Keras is the library for building most deep learning models
- Providing high-level operations for quick and easy implementation
- Tensor libraries are used for low-level operations (tensor manipulation and differentiation)

| | |
|---|---|
| Tensorflow | (by Google) |
| CNTK | (by Microsoft) |
| Theano | (by U. of Montreal) |

# Perceptron

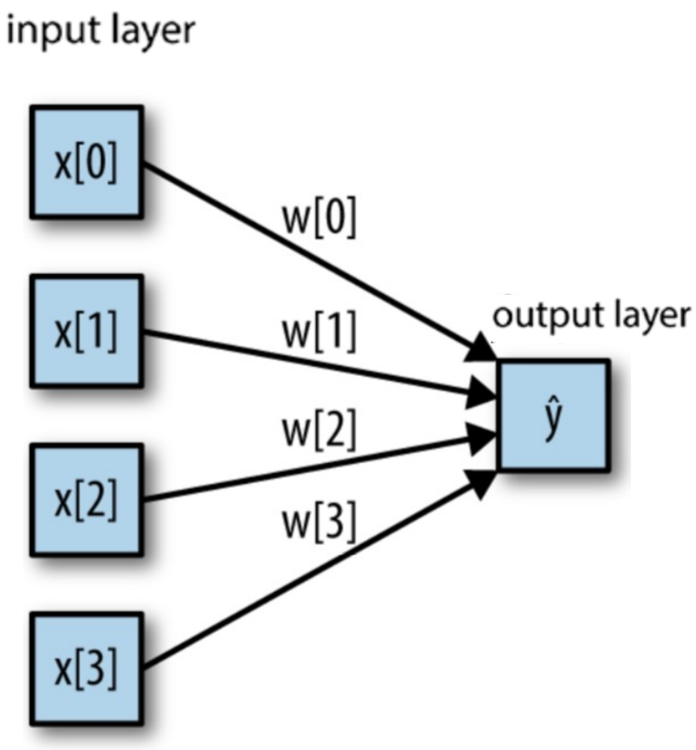
# Perceptron

The Perceptron is the simplest NN model

It is used for classification when there are

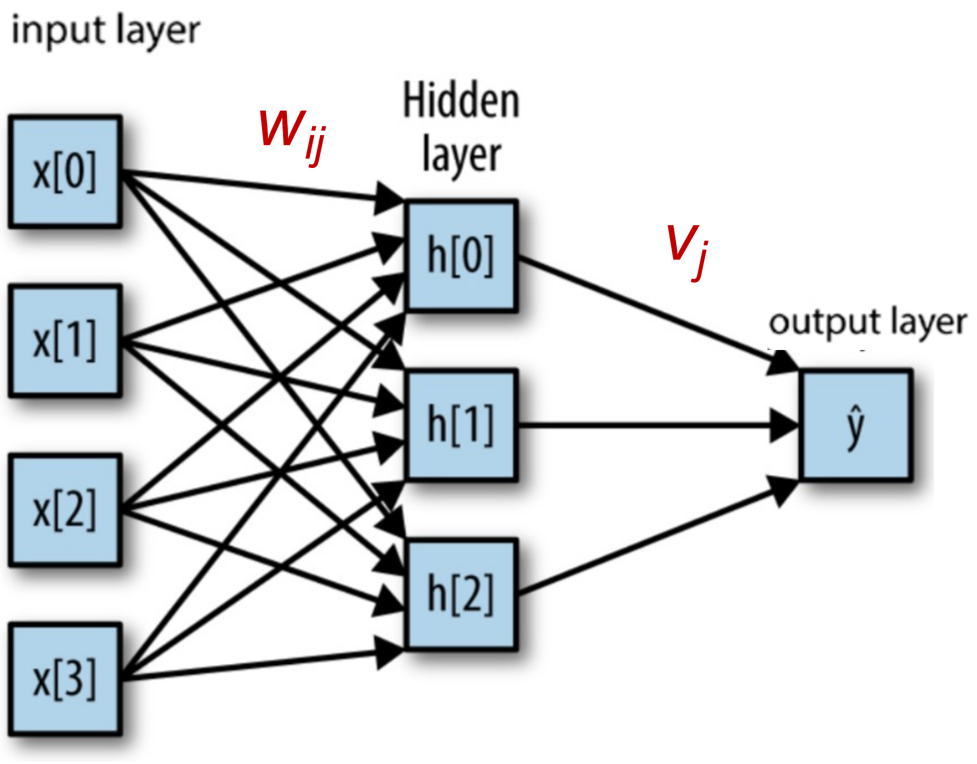
- Two categories
- Linearly separable data

# Perceptron (1 input and 1 output layer)



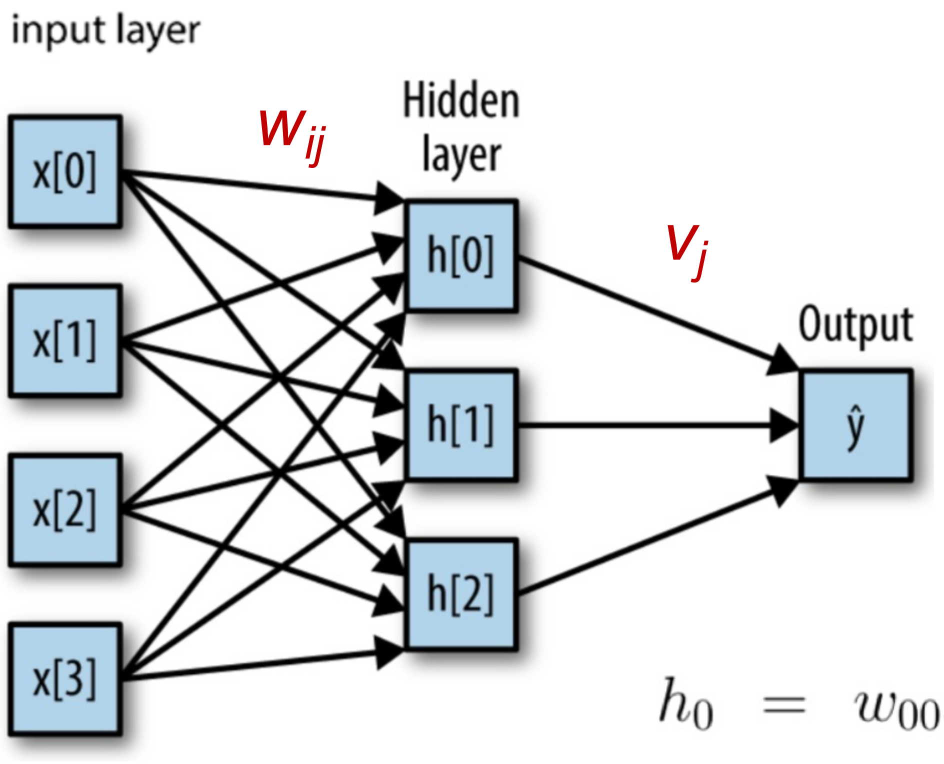
for each row $\hat{y} = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$

# Multilayer Perceptron (MLP)



A perceptron supplemented with one or more hidden layers

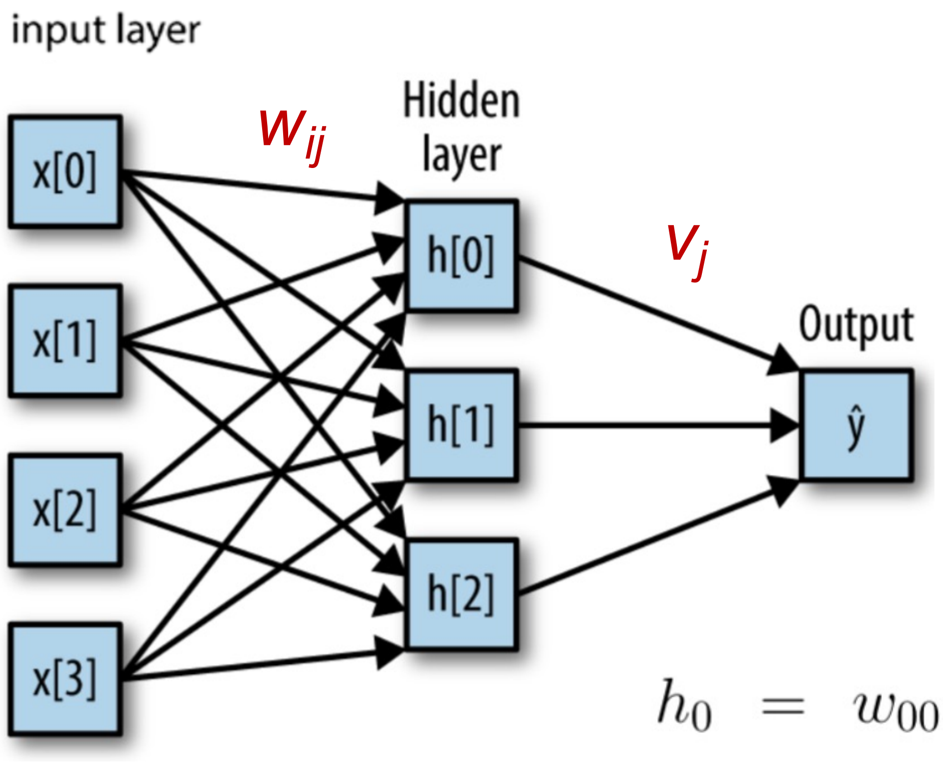
# Multilayer Perceptron (MLP)



A perceptron supplemented with one or more hidden layers

$$
\begin{aligned}
h_0 &= w_{00}x_0 + w_{10}x_1 + w_{20}x_2 + w_{30}x_3 + b_0 \\
h_1 &= w_{01}x_0 + w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1 \\
h_2 &= w_{02}x_0 + w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2
\end{aligned}
$$

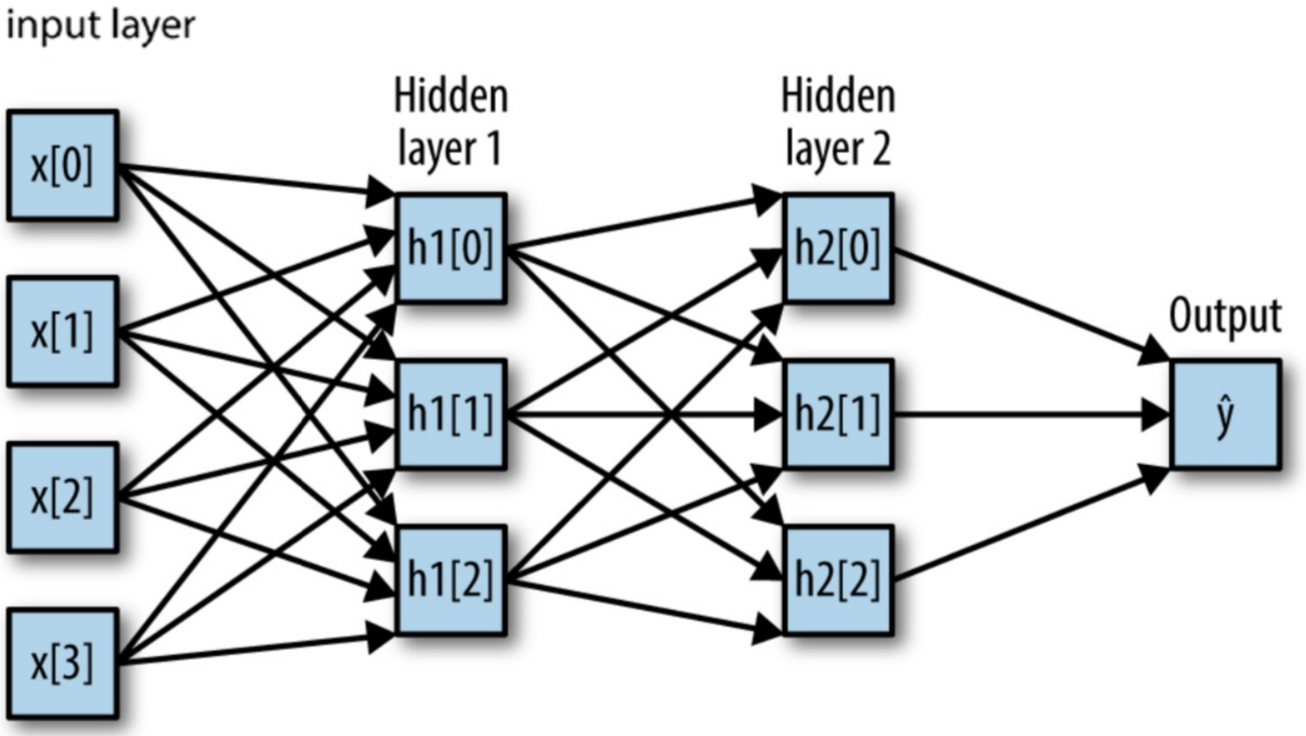
# Multilayer Perceptron



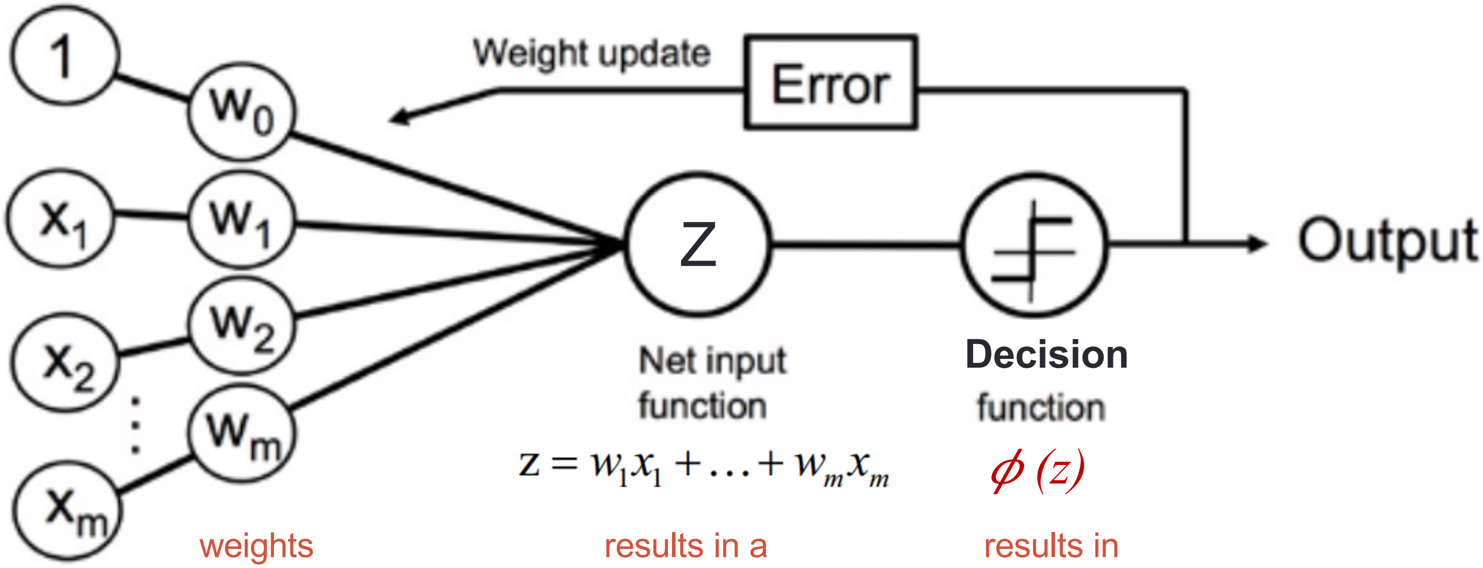
A generalization of linear model with multiple stages of processing to come to a decision (prediction or classification)

$$
\begin{aligned}
h_0 &= w_{00}x_0 + w_{10}x_1 + w_{20}x_2 + w_{30}x_3 + b_0 \\
h_1 &= w_{01}x_0 + w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1 \\
h_2 &= w_{02}x_0 + w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2 \\
\hat{y} &= v_0h_0 + v_1h_1 + v_2h_2 + b
\end{aligned}
$$

# Multilayer Perceptron (two hidden layers)



# Perceptron



$z = w_1x_1 + \ldots + w_mx_m$

$\phi\,(z)$

weights

results in a real number

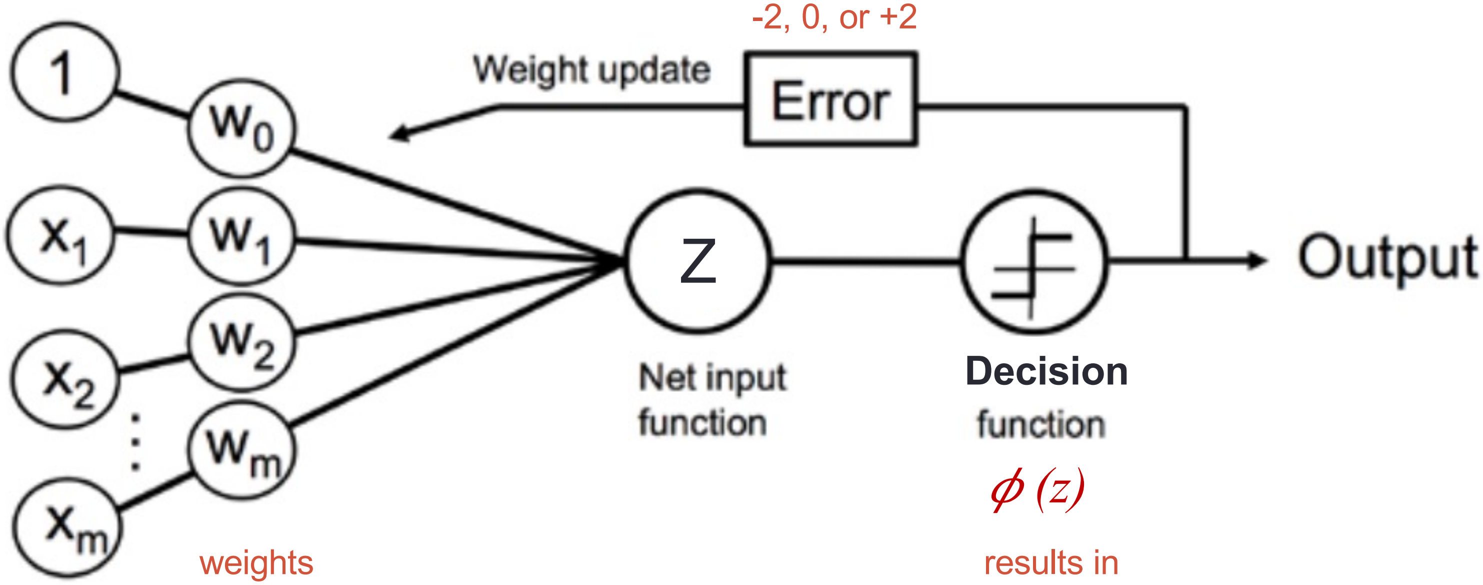
results in -1 or +1

inputs

Combines inputs with weights to obtain the net input *Z* which is passed on to predict the category by means of a decision function $\phi\,(z)$ (also called threshold function)

# Perceptron

results in
-2, 0, or +2



$\phi(z)$

weights

results in
-1 or +1

inputs

- Predicted categories (-1 or 1) are compared to true categories to compute the error and update the weights.
- The process is repeated multiple times (epochs) until convergence (error is small enough).

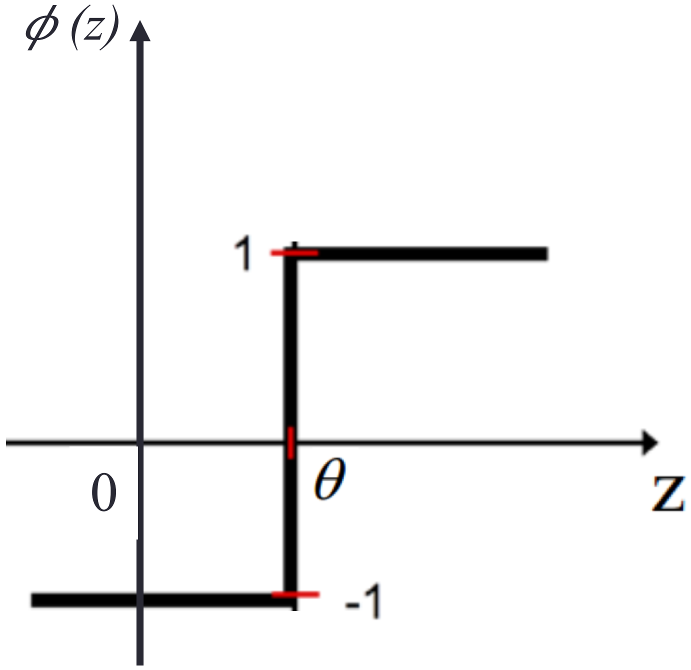
# Perceptron

net input function $\mathrm{z} = w_1 x_1 + \ldots + w_m x_m$

decision function $\phi(\mathrm{z})$

$$\phi(\mathrm{z}) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

$\theta$ is called the threshold



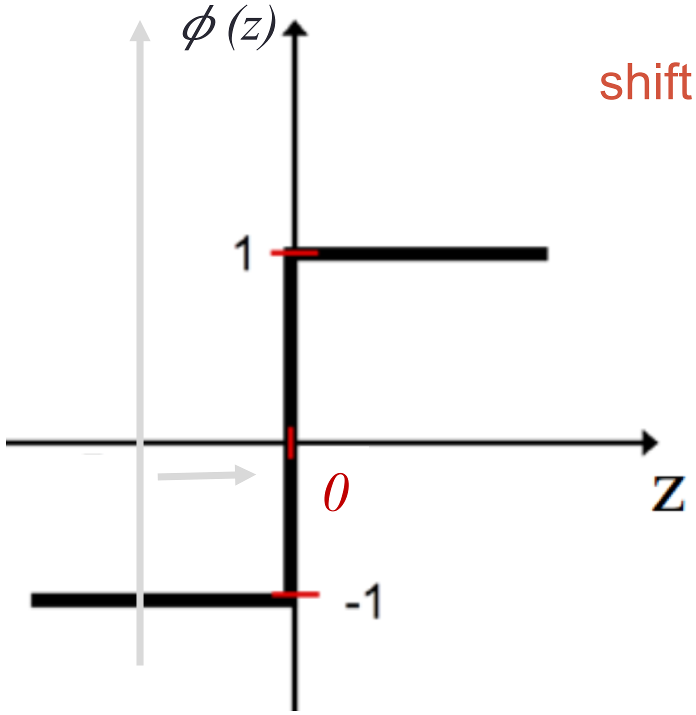
# Perceptron

net input function $$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m$$

$$w_0 = -\theta \text{ and } x_0 = 1$$

decision function

$$\phi(z) = \begin{cases} 1 \;\; if\ z \geq 0 \\ -1 \;\; otherwise \end{cases}$$



shift origin

# Perceptron

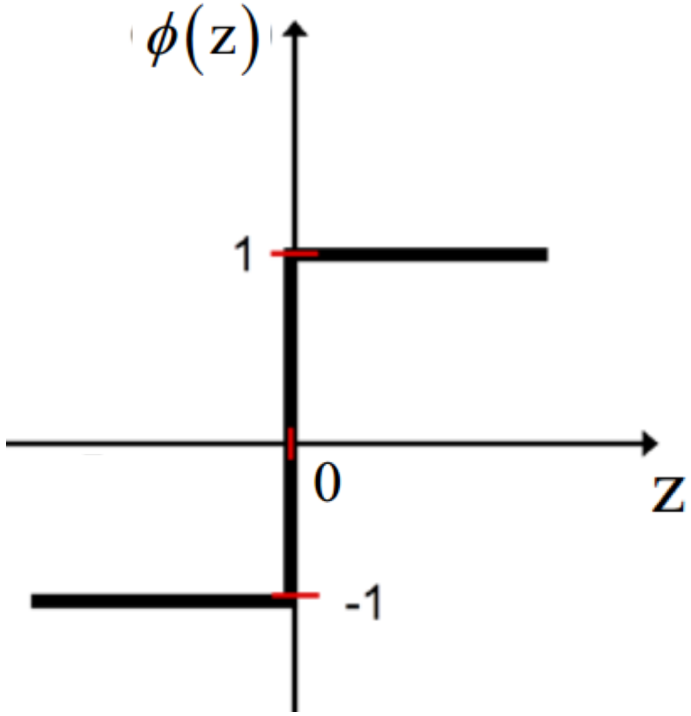
input function in matrix notation

net input function $$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

$$w_0 = -\theta \text{ and } x_0 = 1$$

decision function

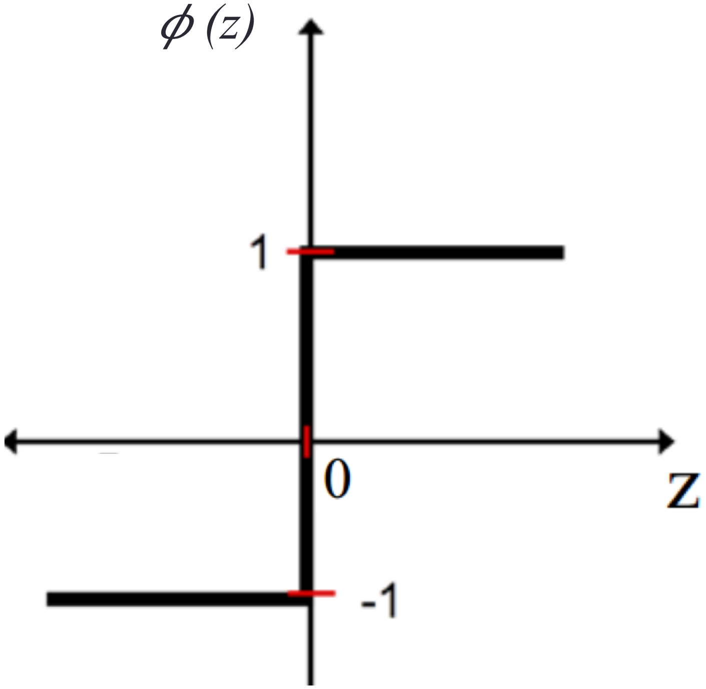
$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$



# Perceptron

threshold function $\phi(\mathrm{z})$

$$\phi(\mathrm{z}) = \begin{cases} 1 \;\; if\ z \geq 0 \\ -1 \;\; otherwise \end{cases}$$



input function

$$\mathrm{z} = \boldsymbol{w}^{T}\boldsymbol{x}$$

# Perceptron learning rule

- For each column, randomly initialize the weights $w_1, \ldots, w_p$
- For each row $i = 1, \ldots, n$

  find $\hat{y}_i = \phi(z_i)$

  find the error $(y_i - \hat{y}_i)$

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda(y_i - \hat{y}_i) x_{ij}$

  for each column $j = 1, \ldots, p$
- Repeat N times

# Perceptron learning rule

- For each column, randomly initialize the weights $w_1, \ldots, w_p$
- For each row $i = 1, \ldots, n$

  $\boxed{\text{find} \quad \hat{y}_i = \phi(z_i)}$

  prediction is +1 or -1

  find the error $(y_i - \hat{y}_i)$

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda(y_i - \hat{y}_i)\, x_{ij}$

  for each column $j = 1, \ldots, p$
- Repeat N times

# Perceptron learning rule

- For each column, randomly initialize the weights $w_1, \ldots, w_p$
- For each row $i = 1, \ldots, n$

  find $\hat{y}_i = \phi(z_i)$

  $\boxed{\text{find the error } (y_i - \hat{y}_i)}$

  error is -2, 0, or +2

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda(y_i - \hat{y}_i)\, x_{ij}$

  for each column $j = 1, \ldots, p$
- Repeat N times

# Perceptron learning rule

- For each column, randomly initialize the weights $w_1, \ldots, w_p$
- For each row $i = 1, \ldots, n$

  find $\hat{y}_i = \phi(z_i)$

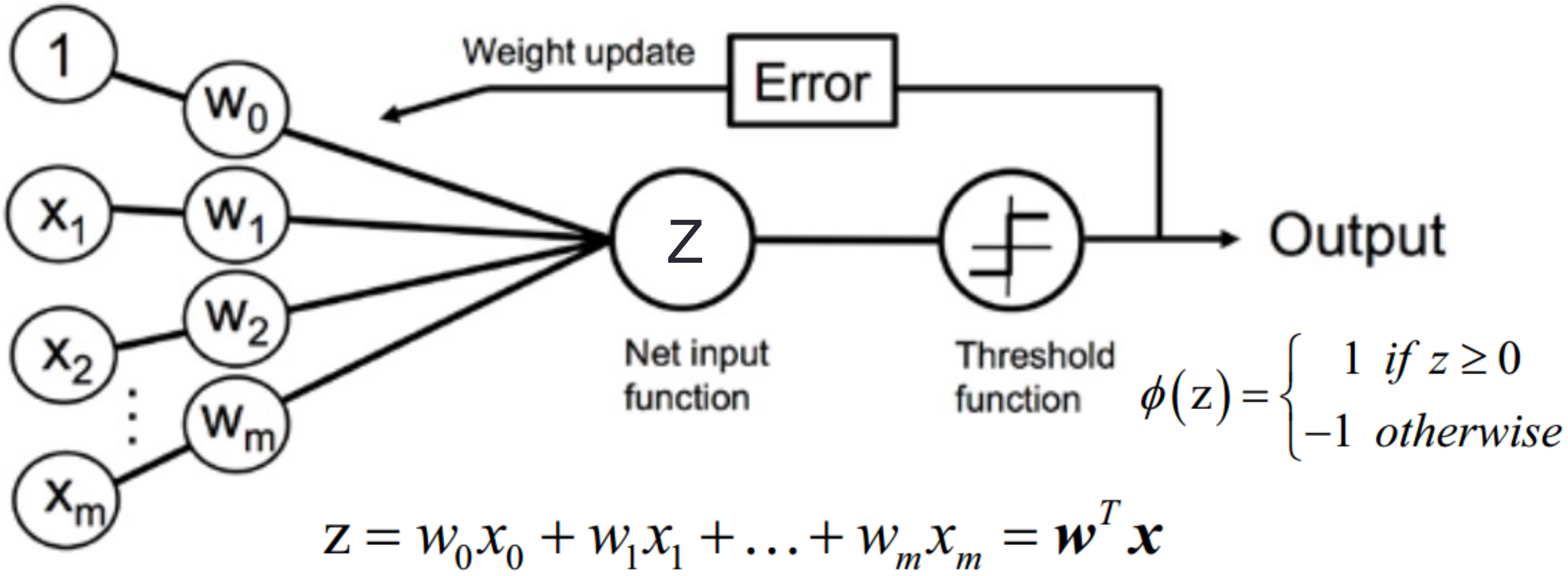
  find the error $(y_i - \hat{y}_i)$

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda(y_i - \hat{y}_i)\, x_{ij}$

  for each column $j = 1, \ldots, p$
- Repeat N times

# Perceptron learning rule



$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

# Adaline

# ADAptive LInear NEuron (Adaline)

- Similar to the Perceptron
- The Perceptron compares the true categories with predicted categories (+1 or -1)
- Adaline compares the true categories with the result of the adaline activation function (a real number)
- Activation function is denoted by $\phi(z)$

# ADAptive LInear NEuron (Adaline)

- Find $z = w_1 x_1 + \ldots + w_m x_m$
- net input *z* is transformed using Activation function

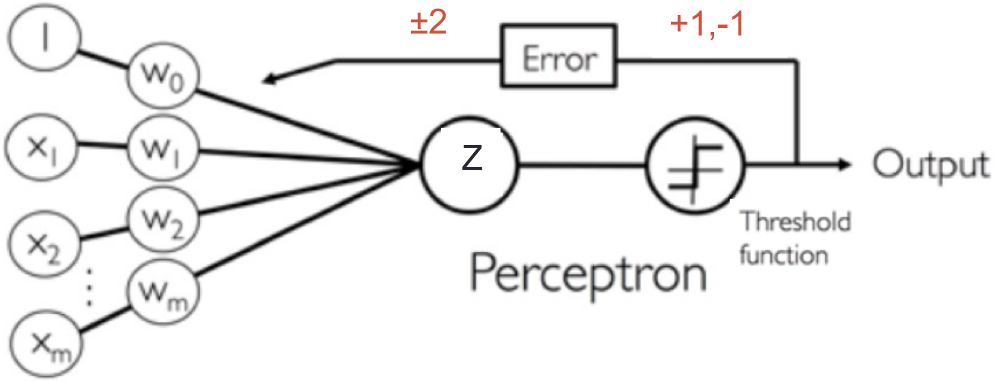
$$\phi(z) \;=\; z$$

- Update weights N times
- Prediction is given by decision function

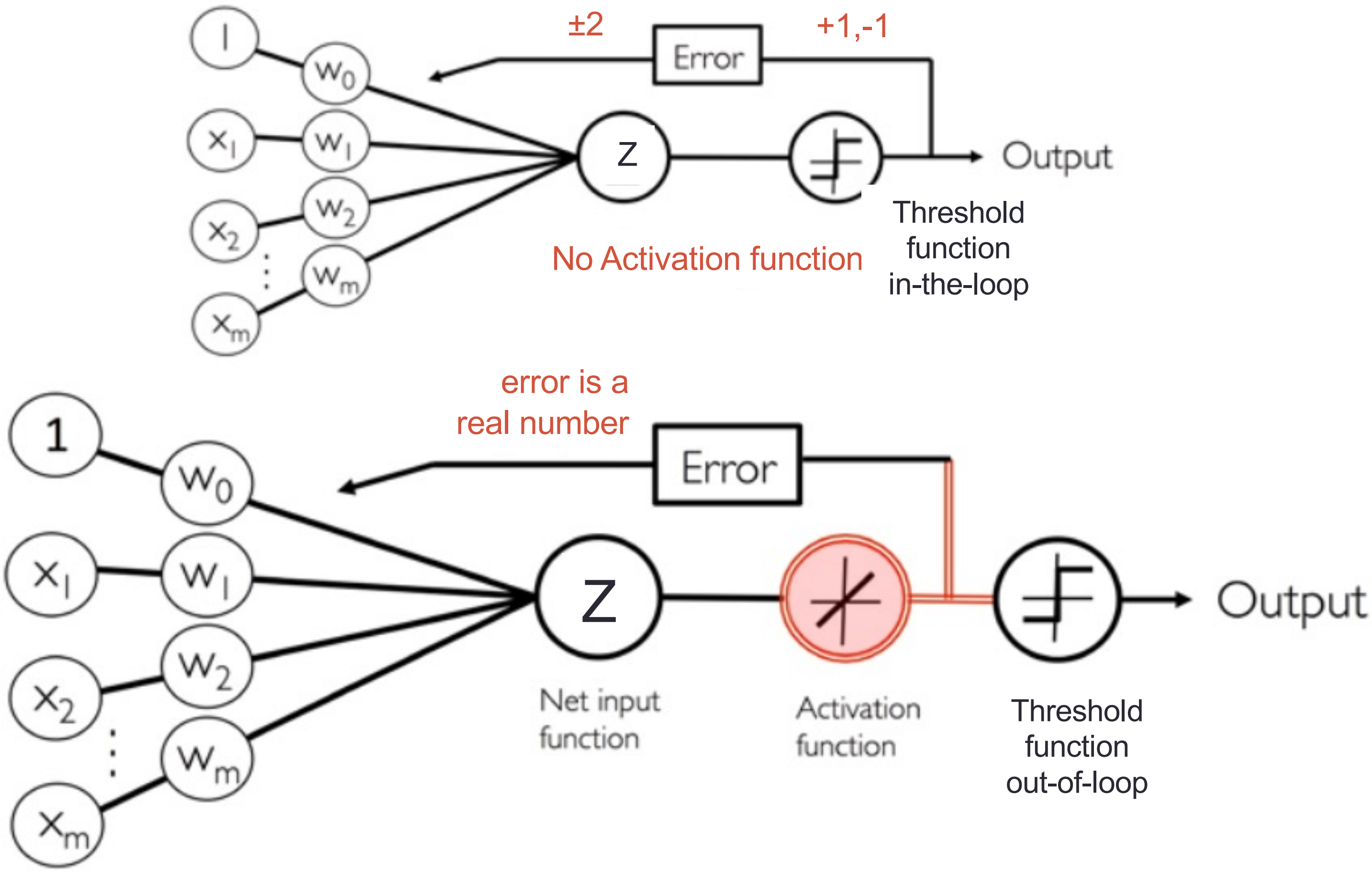
  using the updated weights

$$\hat{y} = \begin{cases} 1 & \text{if} \quad z > 0 \\ -1 & \text{if} \quad z \leq 0 \end{cases}$$

# Perceptron


±2
Error
+1,-1
I
$w_0$
$x_1$
$w_1$
Z
Output
$x_2$
$w_2$
Threshold function
Perceptron
$w_m$
$x_m$

# Adaline vs Perceptron



±2

+1,-1

No Activation function

Threshold function in-the-loop

error is a real number

Threshold function out-of-loop

Adaptive Linear Neuron (Adaline)

# Adaline learning rule

- Randomly initialize the weights $w_1, \ldots, w_p$
- For each row $i = 1, \ldots, n$

  find $z_i = w_1 x_{1i} + \cdots + w_p x_{pi}$

  find the error $( y_i - z_i )$

  error is a real number

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda \sum_{i=1}^{n} (y_i - \phi(z_i))\, x_{ij}$

  for each column $j = 1, \ldots, p$

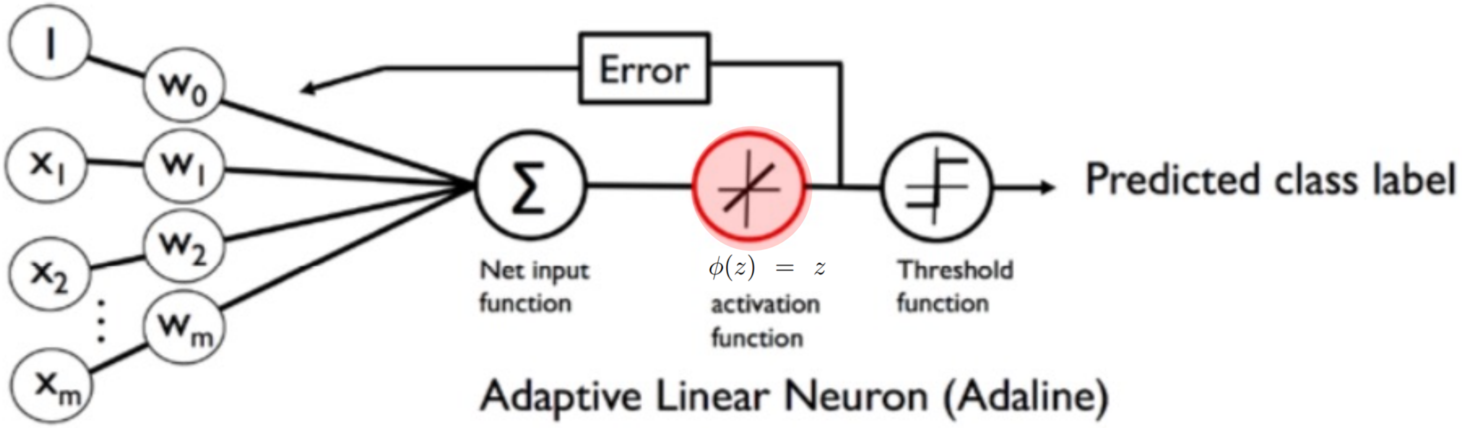
- Repeat N times, then predict with decision function

# Logistic Regression

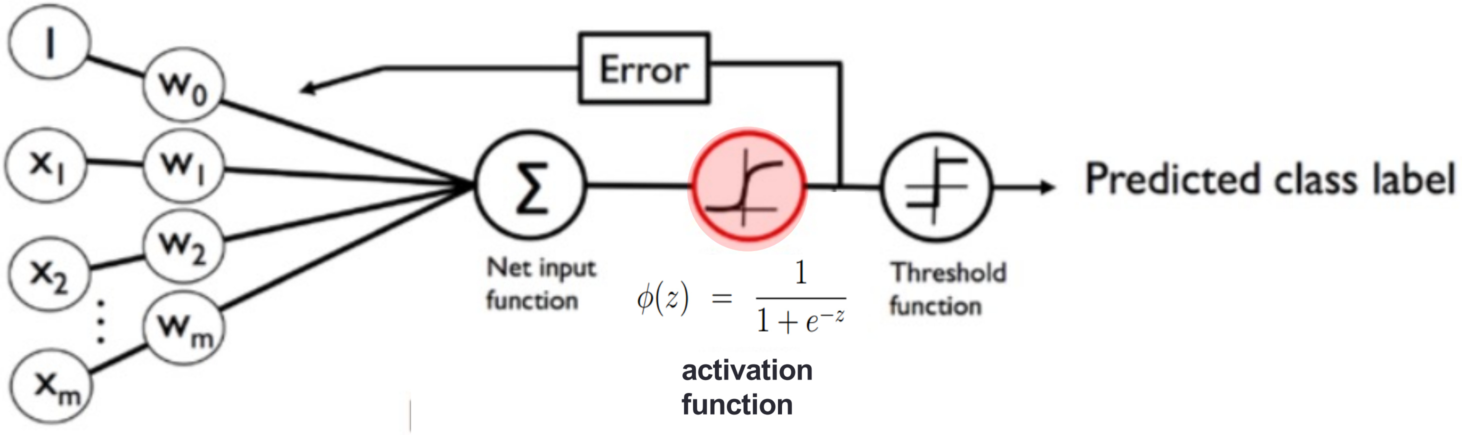
# Logistic Regression

- A special case of Adaline with different activation function and loss function

# Adaline vs Logistic Regression



Adaptive Linear Neuron (Adaline)



Logistic Regression

# Adaline

- activation function

$$\phi(z) = z \qquad \text{where} \qquad z = w_1\, x_1 + \cdots + w_p\, x_p$$

- loss function

$$\begin{aligned} J(w) &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_i - \phi(z_i))^2 \quad = \quad \sum_{i=1}^{n} (y_i - z_i)^2 \end{aligned}$$

# Logistic Regression

- activation function

$$\phi(z) = \frac{1}{1+e^{-z}} \qquad \text{where} \qquad z = w_1\, x_1 + \cdots + w_p\, x_p$$

- loss function

$$J(w) = -\sum_{i=1}^{n} [\, y_i \; \log \phi(z_i) + (1-y_i) \; \log(1-\phi(z_i)) \,]$$

# Logistic Regression

- Categories labeled as 0 and +1
- Activation function

$$\phi(z) = \frac{1}{1+e^{-z}}$$

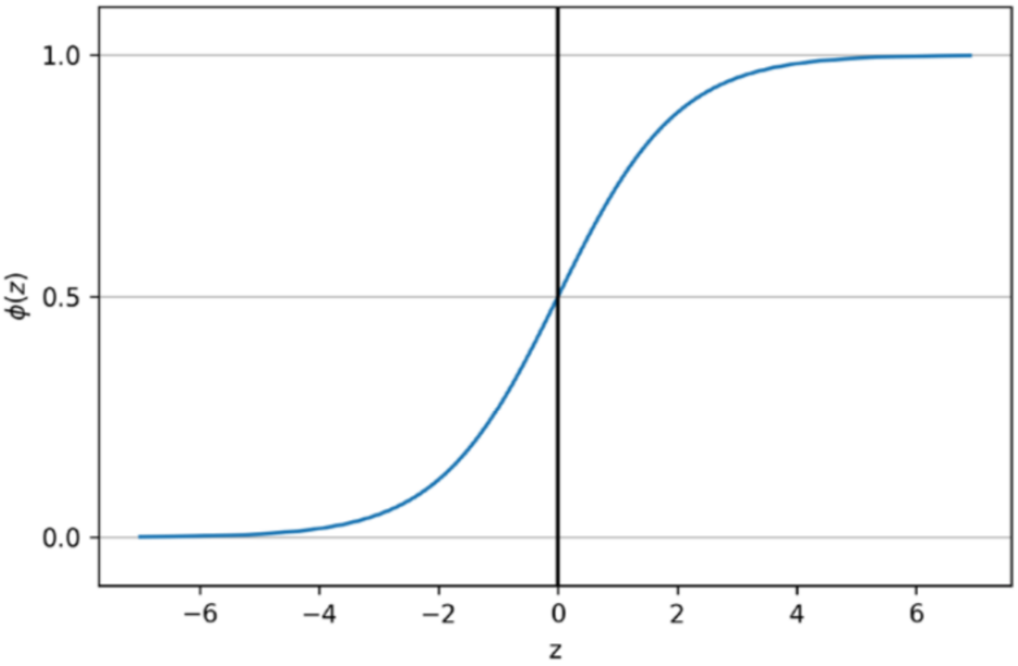
- Decision function (Threshold function)

$$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

# Logistic Regression

- Activation function

$$\phi(z) = \frac{1}{1+e^{-z}}$$



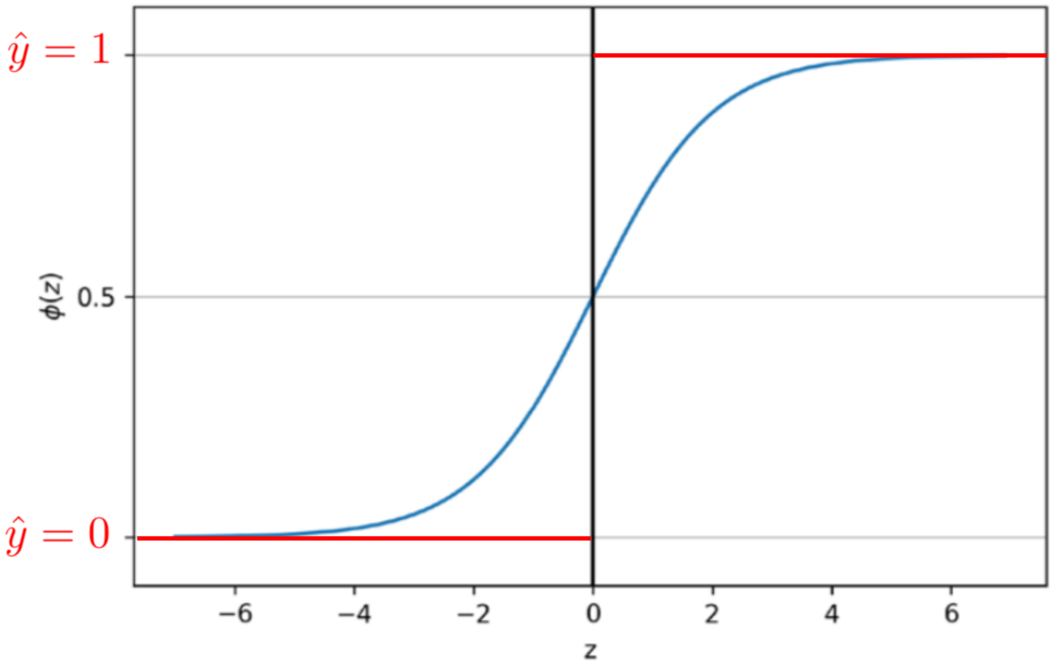
- Decision function

$$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

# Logistic Regression

- Activation function

$$\phi(z) = \frac{1}{1+e^{-z}}$$



- Decision function

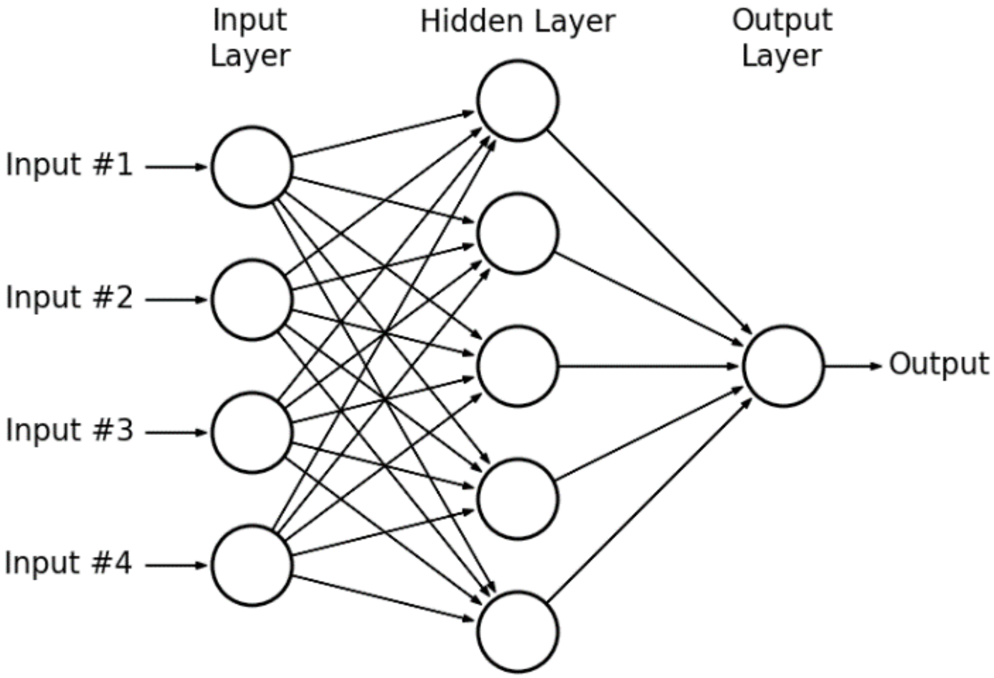
$$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{y} = \begin{cases} 1 & \text{if } z \geq 0.0 \\ 0 & \text{otherwise} \end{cases}$$

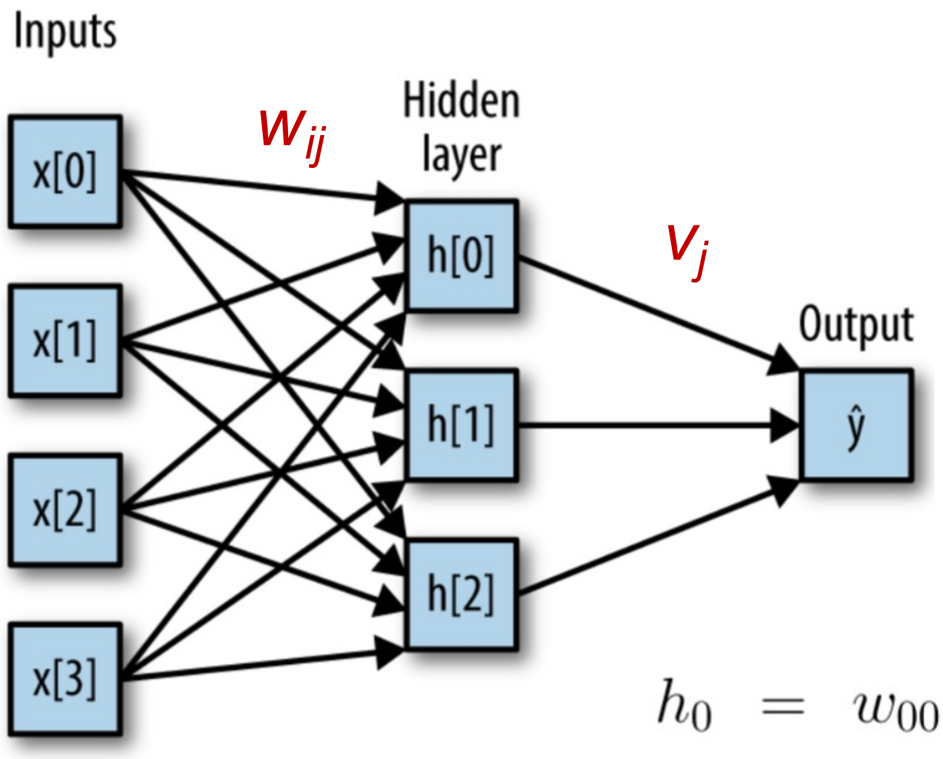
# Multilayer Perceptron

# Multilayer Perceptron

- A Perceptron with at least one intermediate layer



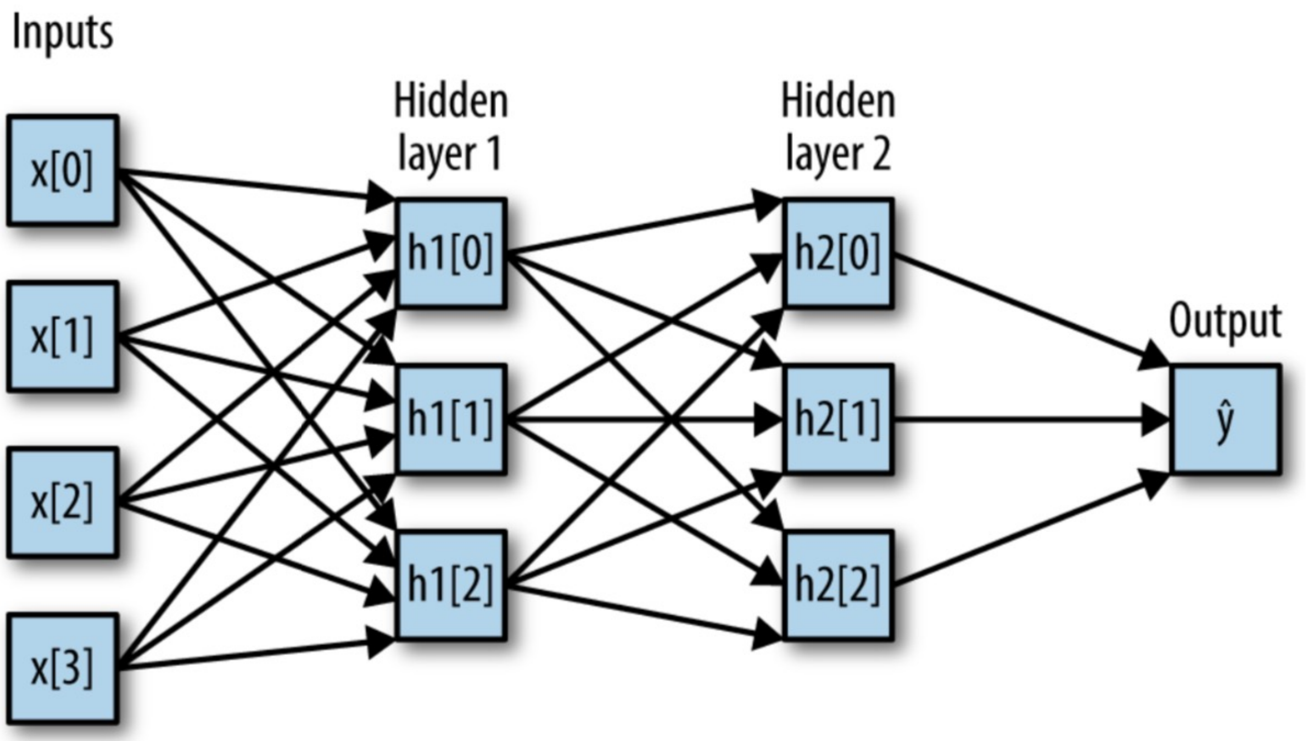
# Multilayer Perceptron (MLP)



A generalization of linear model with multiple stages of processing to come to a decision (prediction or classification)

$$
\begin{aligned}
h_0 &= w_{00}x_0 + w_{10}x_1 + w_{20}x_2 + w_{30}x_3 + b_0 \\
h_1 &= w_{01}x_0 + w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1 \\
h_2 &= w_{02}x_0 + w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2 \\
\hat{y} &= v_0h_0 + v_1h_1 + v_2h_2 + b
\end{aligned}
$$

# Multilayer Perceptron (MLP)



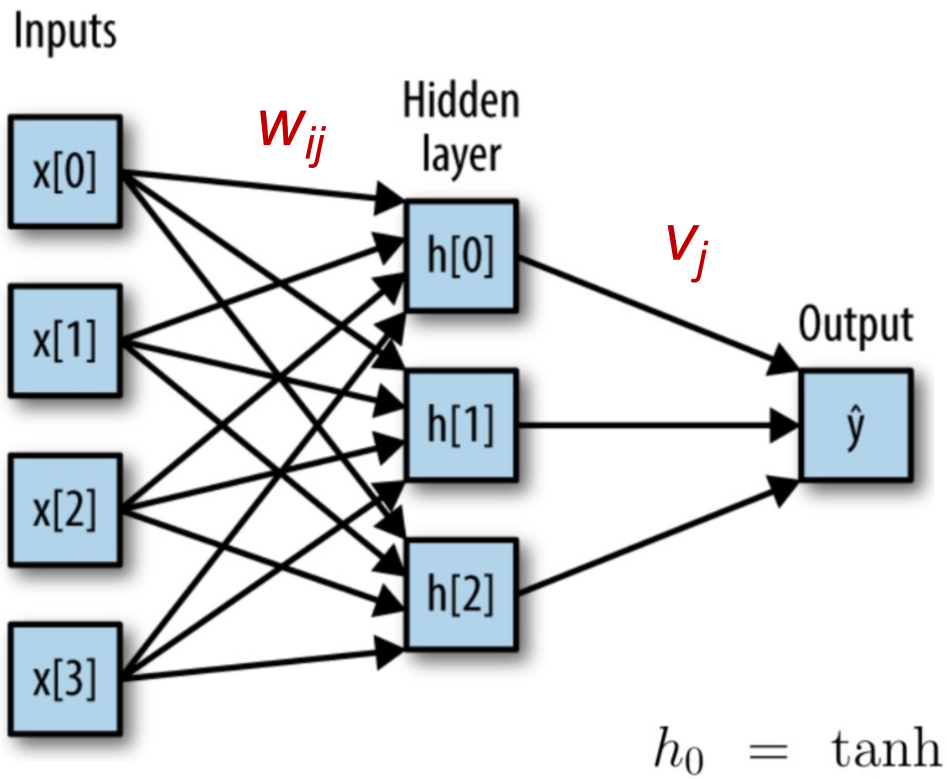
May have multiple hidden layers

# Activation Functions

- Activation functions are used to let the NN become a nonlinear model (for classification or regression)
- They help in the convergence of the learning algorithm
- Common Activation functions

| | |
|---|---|
| tanh | MLP, RNN |
| ReLU | MLP, CNN |
| softmax | (multi-class classification) |

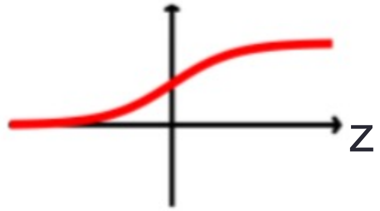
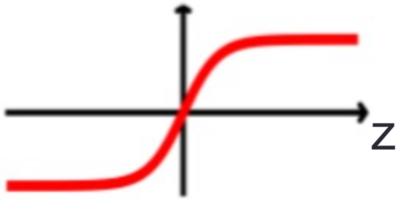
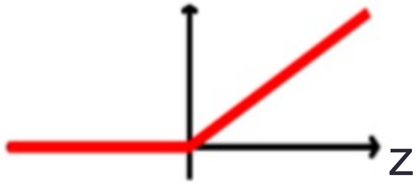
# MLP – tanh activation function



Activation functions are used to let the NN become a nonlinear model

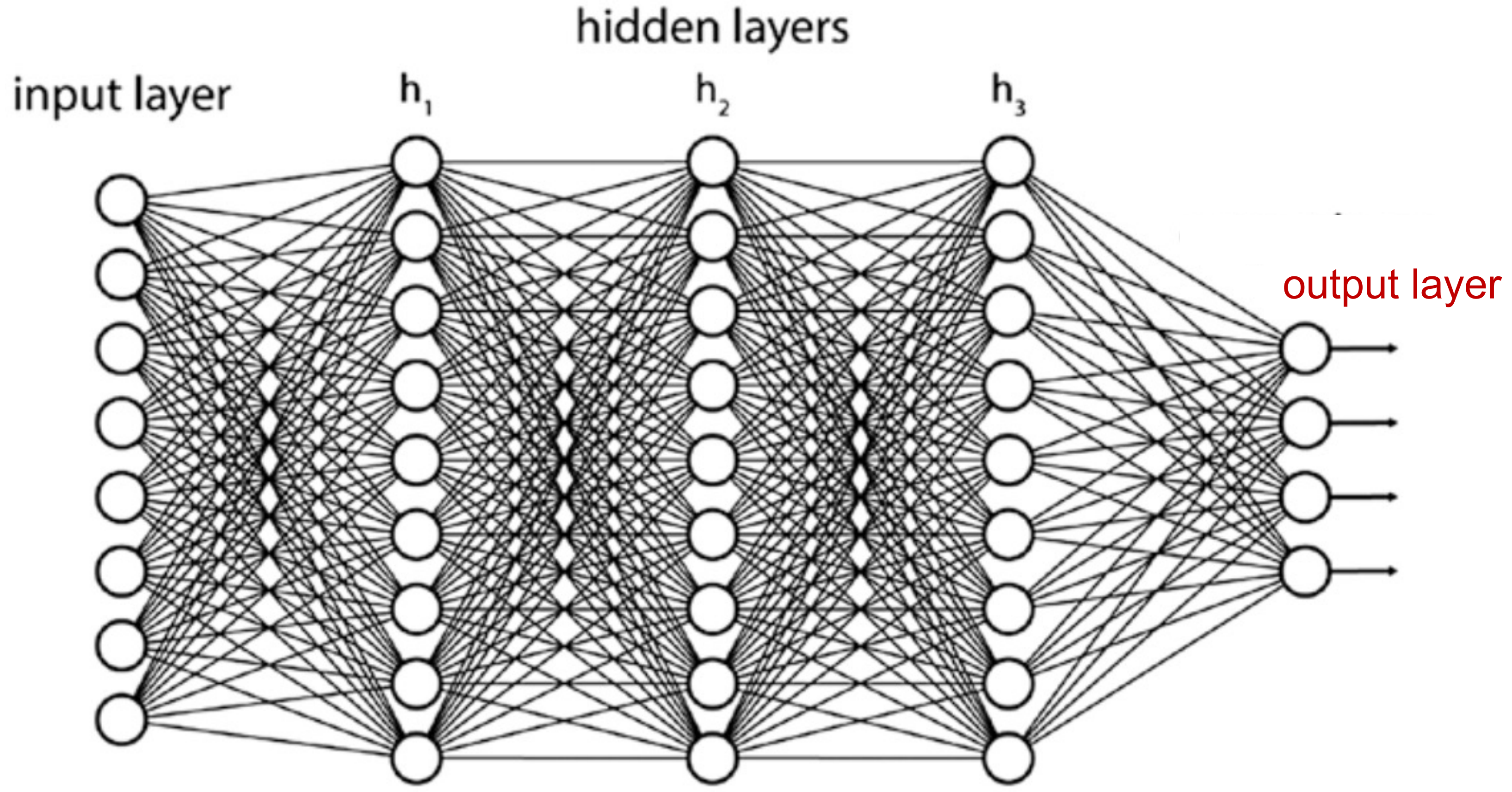
$$
\begin{aligned}
h_0 &= \underline{\tanh}(w_{00}x_0 + w_{10}x_1 + w_{20}x_2 + w_{30}x_3 + b_0) \\
h_1 &= \underline{\tanh}(w_{01}x_0 + w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1) \\
h_2 &= \underline{\tanh}(w_{02}x_0 + w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2) \\
\hat{y} &= v_0h_0 + v_1h_1 + v_2h_2 + b
\end{aligned}
$$

# Common Activation Functions

| | | |
|---|---|---|
| Logistic (sigmoid) | $\phi(z)= \dfrac{1}{1 + e^{-z}}$ |  |
| Hyperbolic Tangent (tanh) | $\phi(z)= \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$ |  |
| ReLU (Rectified Linear Unit) | $\phi(z)= \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$ |  |

# Deep Neural Network

- Model may have thousands of parameters (regression coefficients)



# Notes

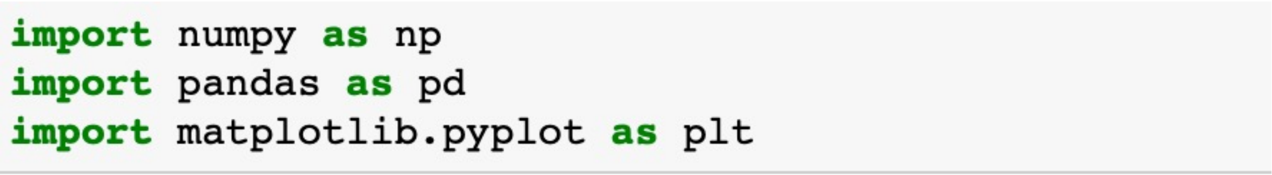
- Scale the data before building NN models
- NN layer is called Dense if *all* nodes are connected with nodes from neighbor layers
- NN is called Multilayer Perceptron (MLP) if all layers are dense

# Notes

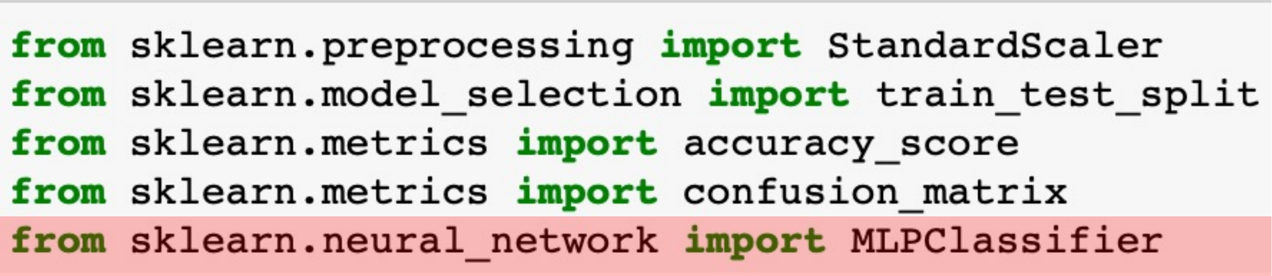
- For small datasets use a small number of hidden layers otherwise risk of overfitting
- Consider increasing the number of hidden layers with larger datasets
- NN hyperparameters:
  - number of hidden layers
  - number of nodes per layer
  - learning rate $\lambda$

# Example 1 – Multilayer Perceptron

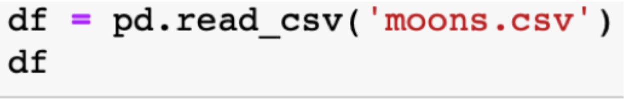
# Example 1 – Multilayer Perceptron



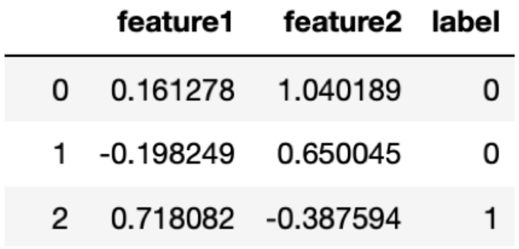
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```



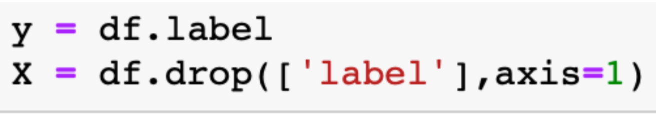
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.neural_network import MLPClassifier
```



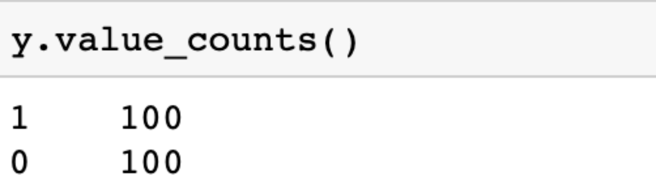
```
df = pd.read_csv('moons.csv')
df
```



|   | feature1 | feature2 | label |
|---|---|---|---|
| 0 | 0.161278 | 1.040189 | 0 |
| 1 | -0.198249 | 0.650045 | 0 |
| 2 | 0.718082 | -0.387594 | 1 |

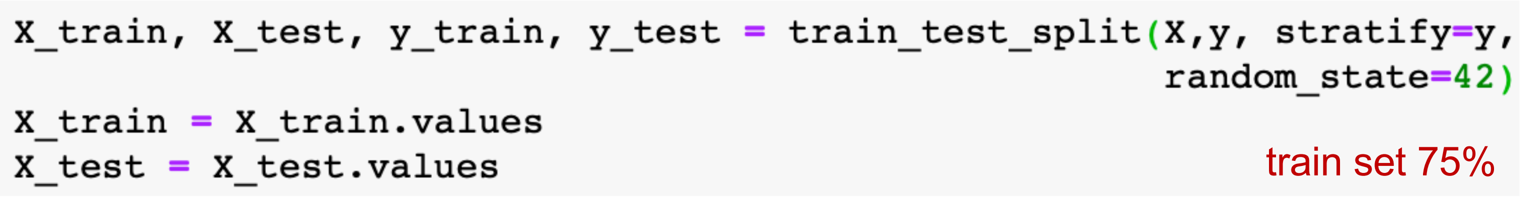


```
y = df.label
X = df.drop(['label'],axis=1)
```



```
y.value_counts()
```

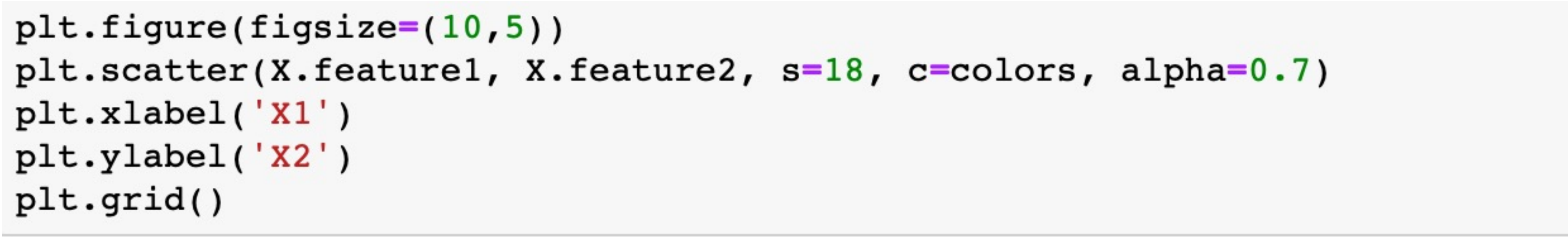
```
1    100
0    100
```



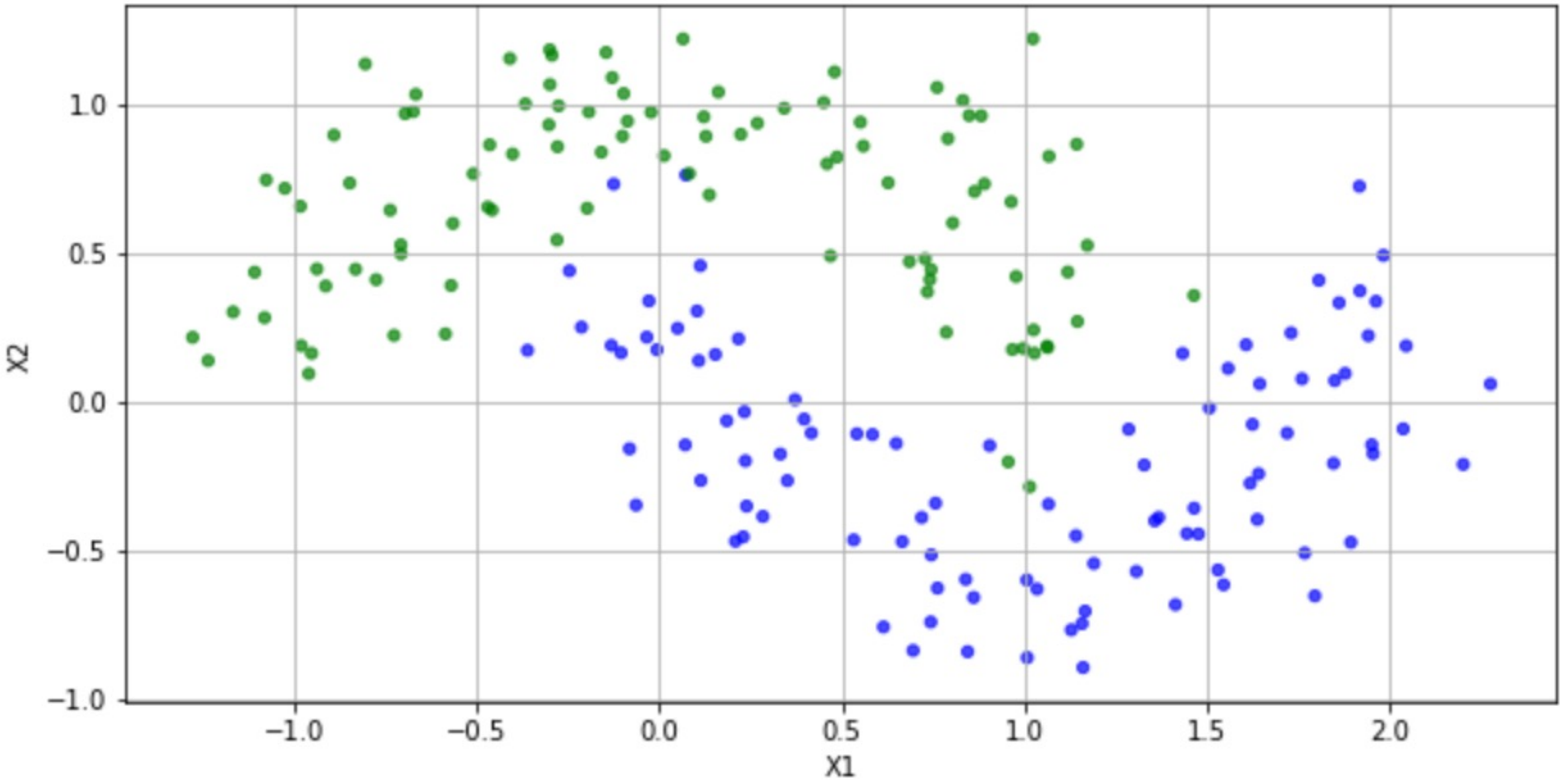
```
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y,
                                                    random_state=42)
X_train = X_train.values
X_test = X_test.values
```

train set 75%

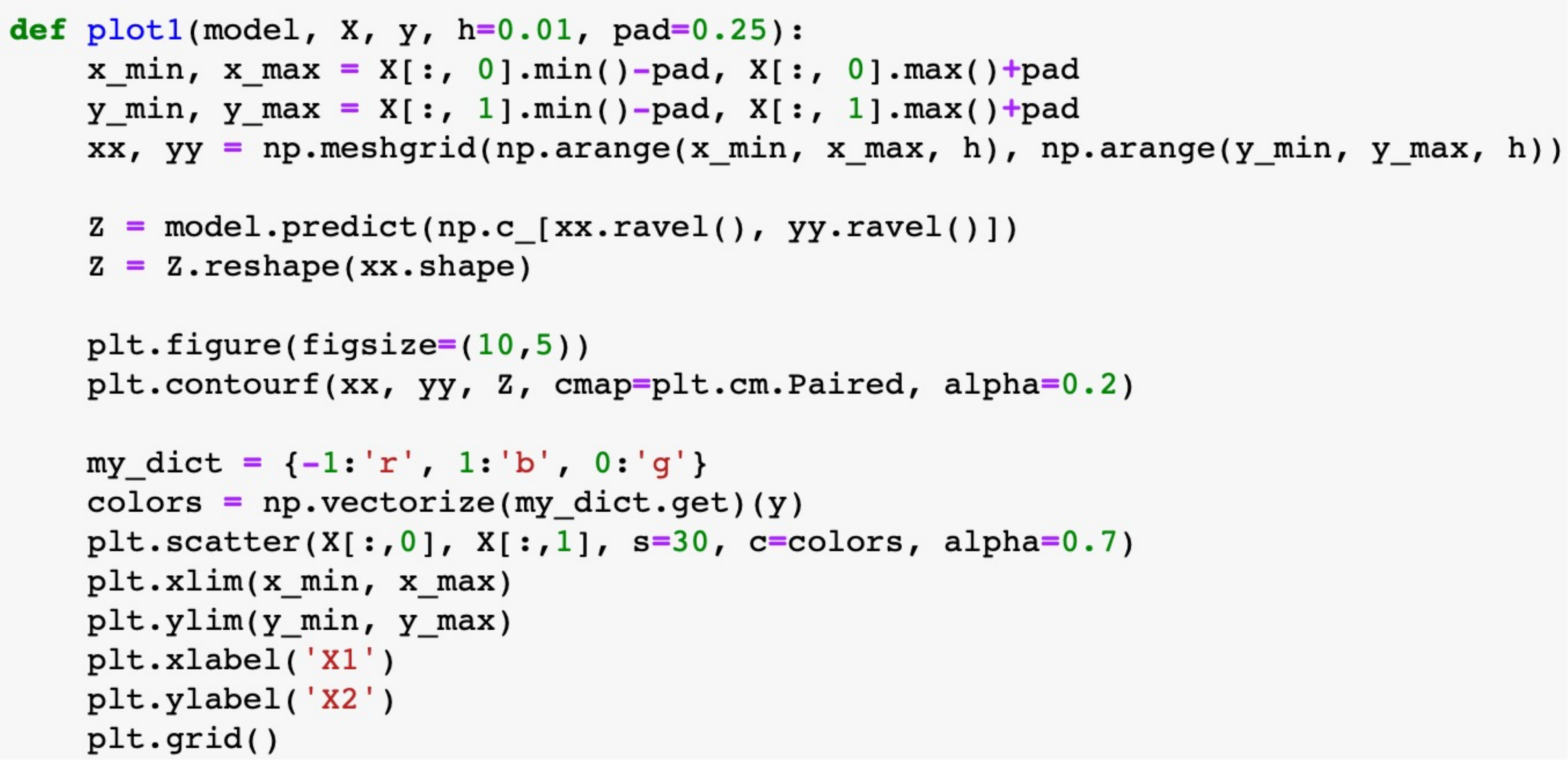
# Example 1 – Nonlinearly separable data



```
plt.figure(figsize=(10,5))
plt.scatter(X.feature1, X.feature2, s=18, c=colors, alpha=0.7)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```

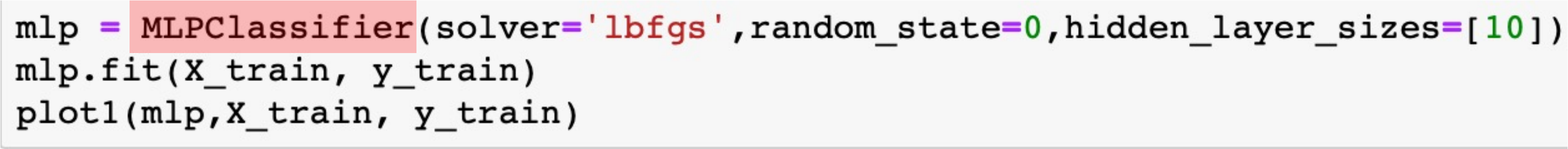


# Example 1 – Function to display boundary

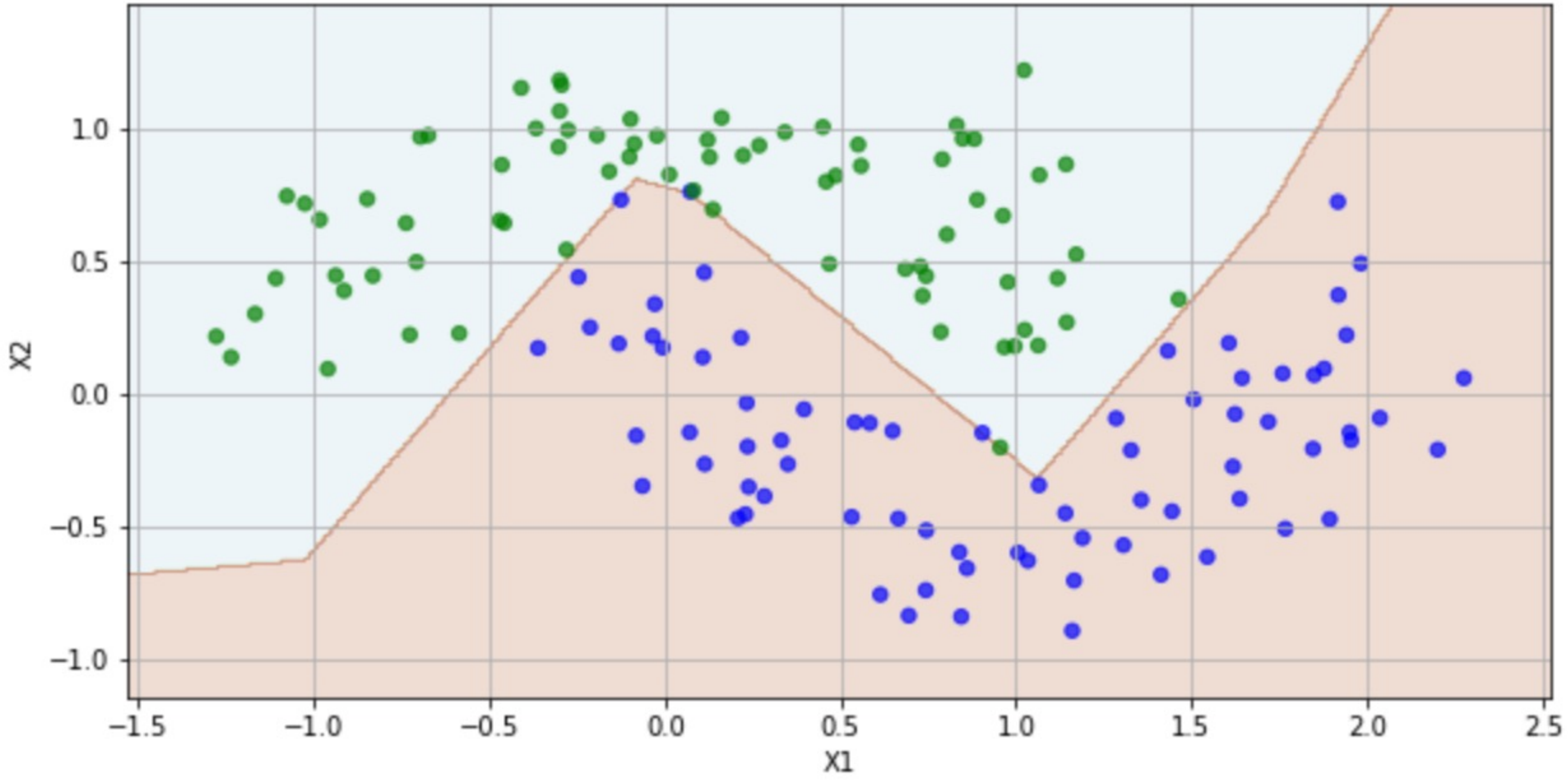


```
def plot1(model, X, y, h=0.01, pad=0.25):
    x_min, x_max = X[:, 0].min()-pad, X[:, 0].max()+pad
    y_min, y_max = X[:, 1].min()-pad, X[:, 1].max()+pad
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10,5))
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.2)

    my_dict = {-1:'r', 1:'b', 0:'g'}
    colors = np.vectorize(my_dict.get)(y)
    plt.scatter(X[:,0], X[:,1], s=30, c=colors, alpha=0.7)
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.grid()
```

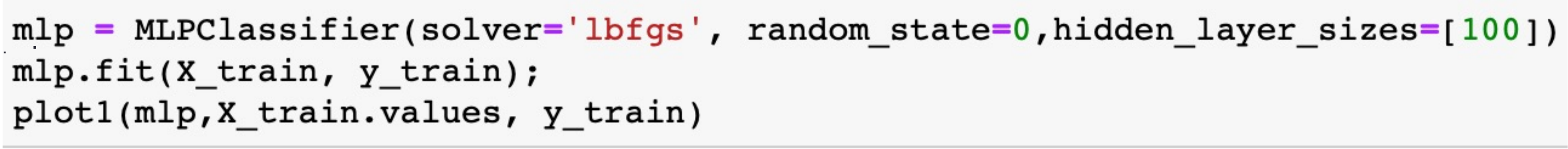
# Example 1:   1 hidden layer with 10 nodes



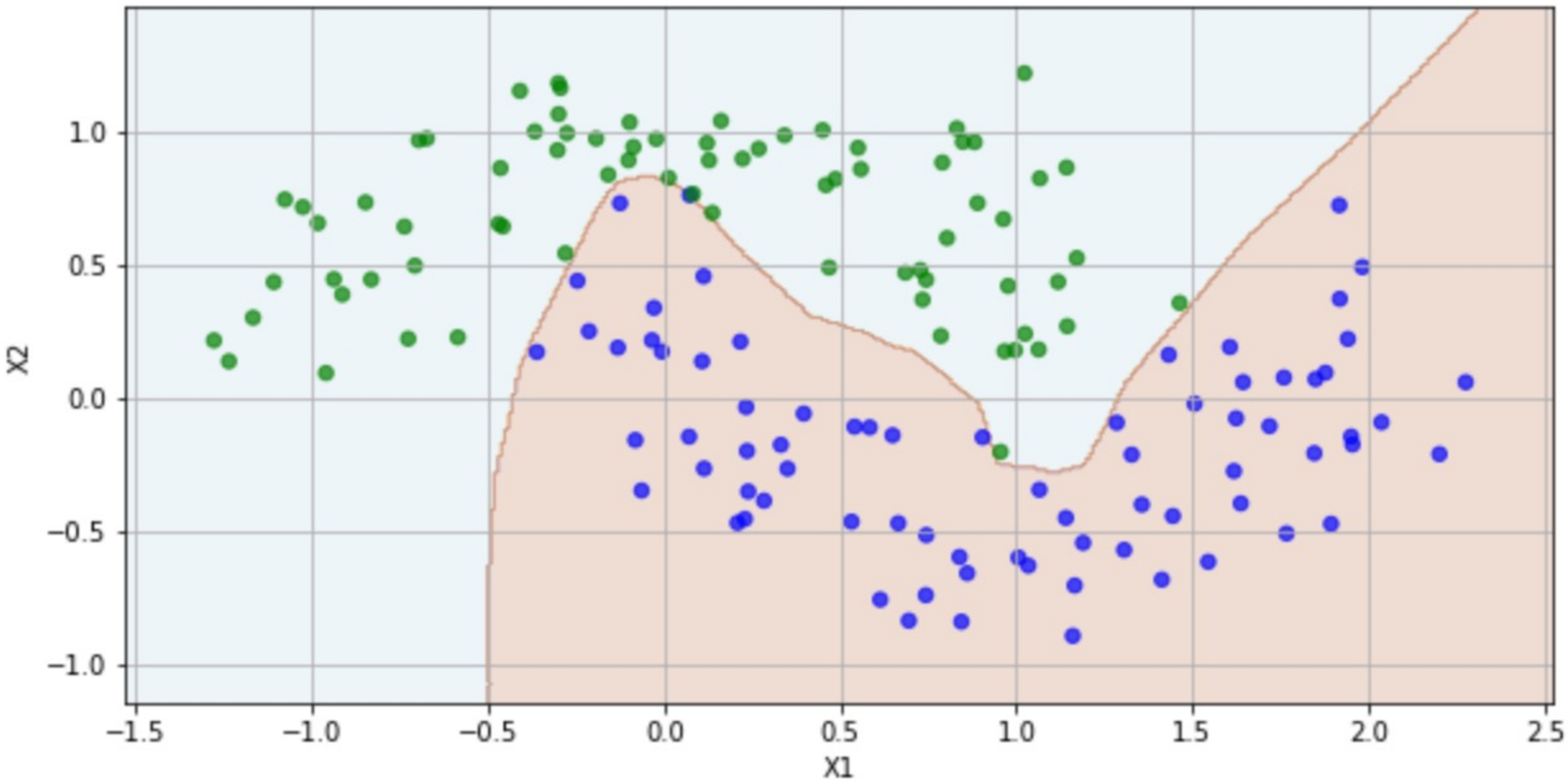
```
mlp = MLPClassifier(solver='lbfgs',random_state=0,hidden_layer_sizes=[10])
mlp.fit(X_train, y_train)
plot1(mlp,X_train, y_train)
```



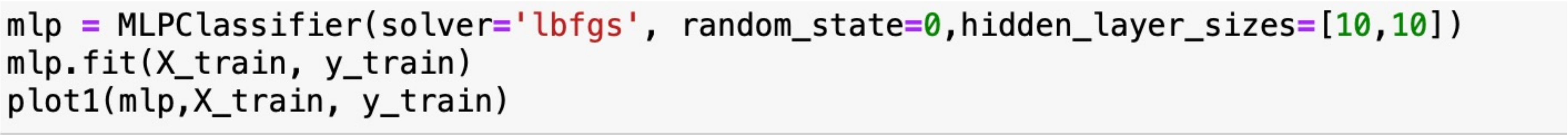
# Example 1: 1 hidden layer with 100 nodes



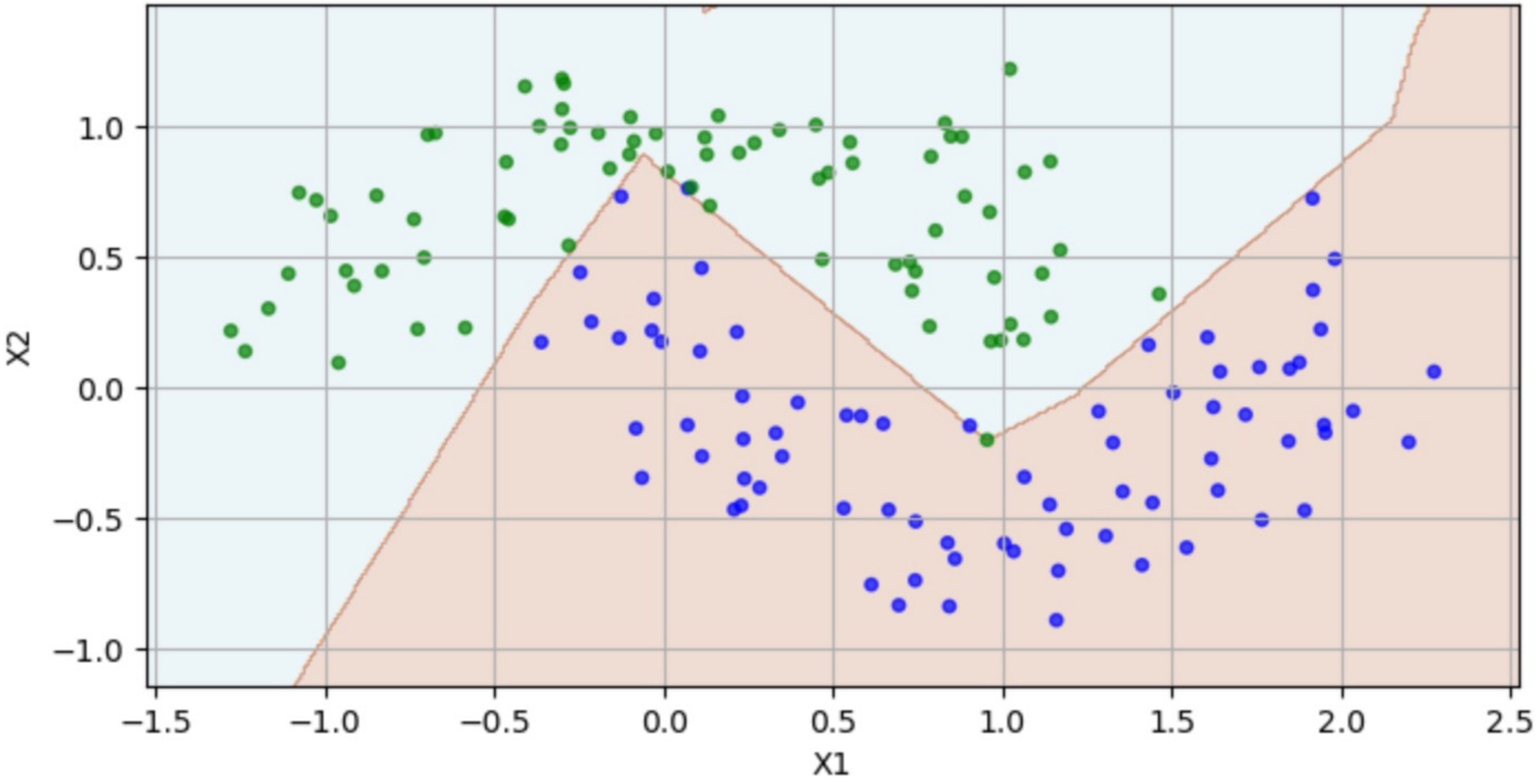
```
mlp = MLPClassifier(solver='lbfgs', random_state=0,hidden_layer_sizes=[100])
mlp.fit(X_train, y_train);
plot1(mlp,X_train.values, y_train)
```



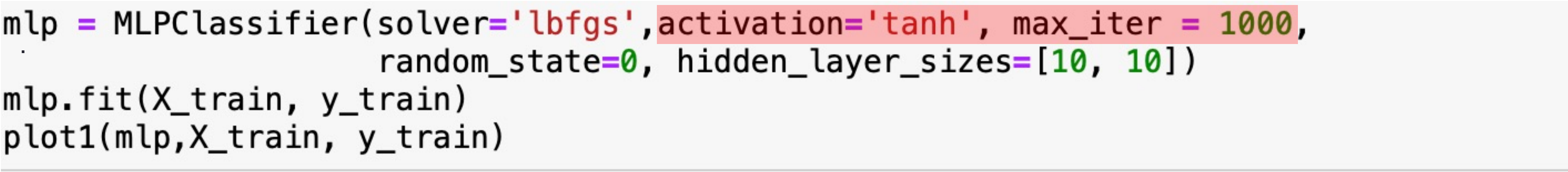
# Example 1:   2 hidden layers with 10 nodes each



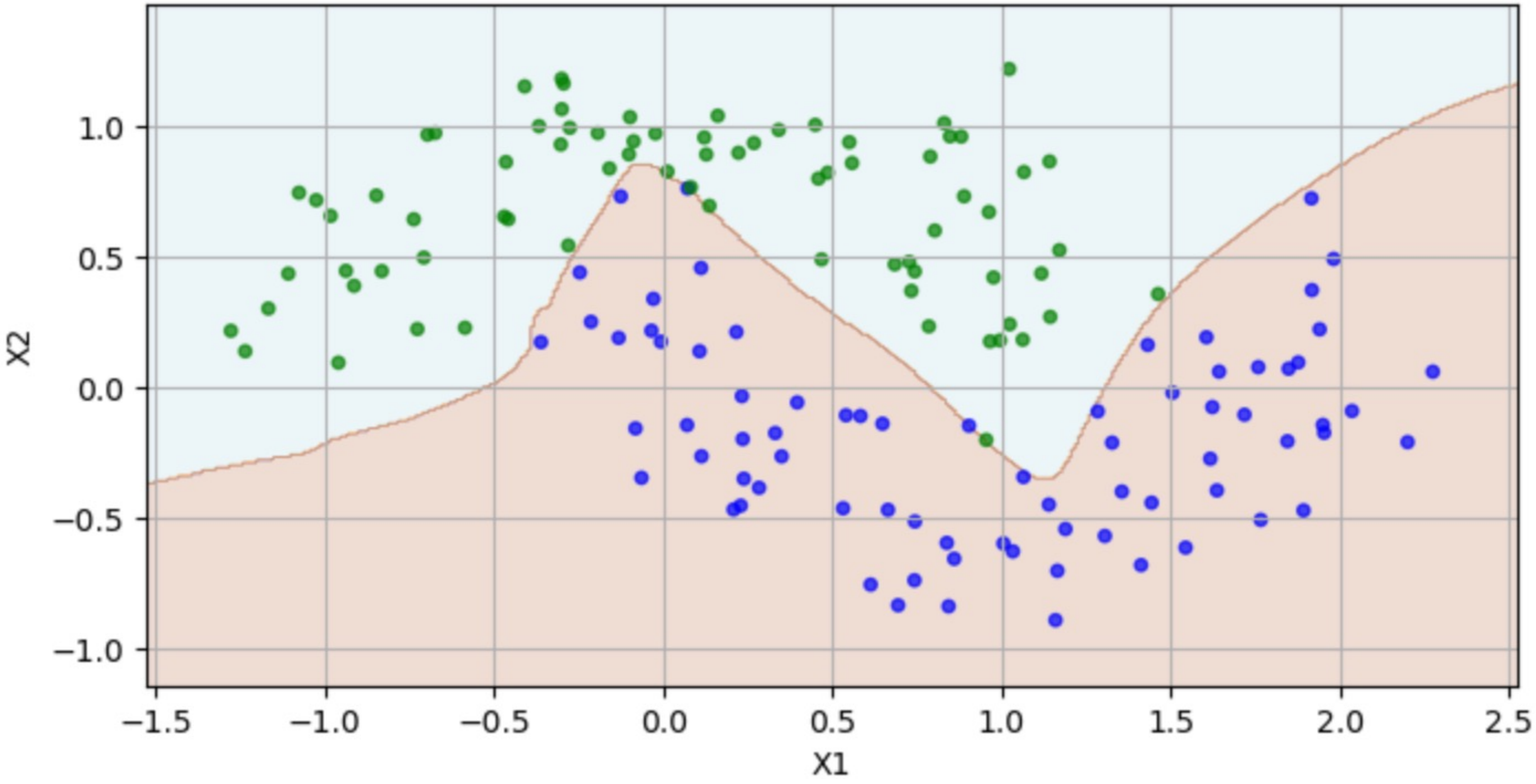
```
mlp = MLPClassifier(solver='lbfgs', random_state=0,hidden_layer_sizes=[10,10])
mlp.fit(X_train, y_train)
plot1(mlp,X_train, y_train)
```



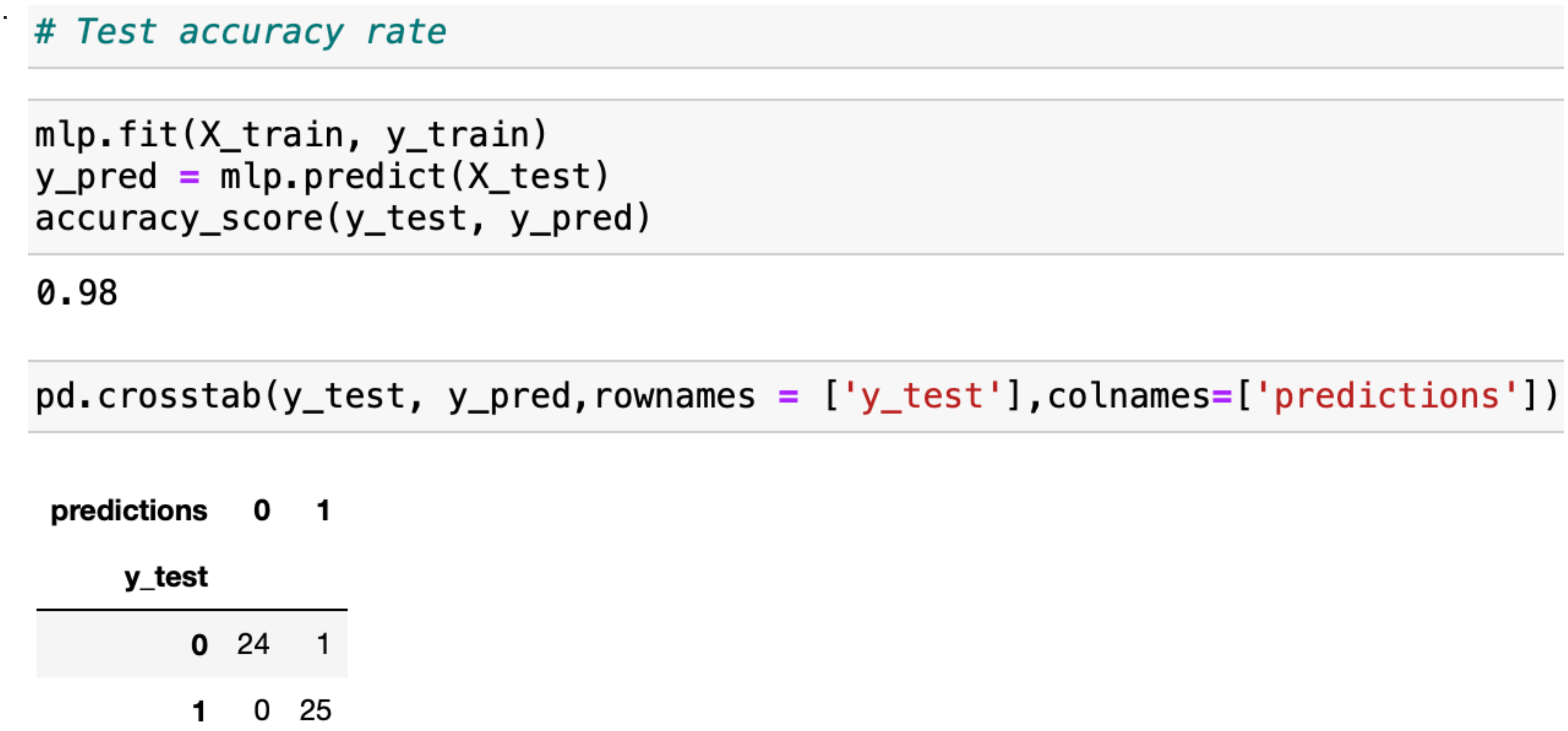
# Example 1: Two 10-node hidden layers with **tanh**



```
mlp = MLPClassifier(solver='lbfgs',activation='tanh', max_iter = 1000,
                    random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
plot1(mlp,X_train, y_train)
```



# Example 1: Two 10-node hidden layers with **tanh**



```
# Test accuracy rate
```

```
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.98
```

```
pd.crosstab(y_test, y_pred,rownames = ['y_test'],colnames=['predictions'])
```

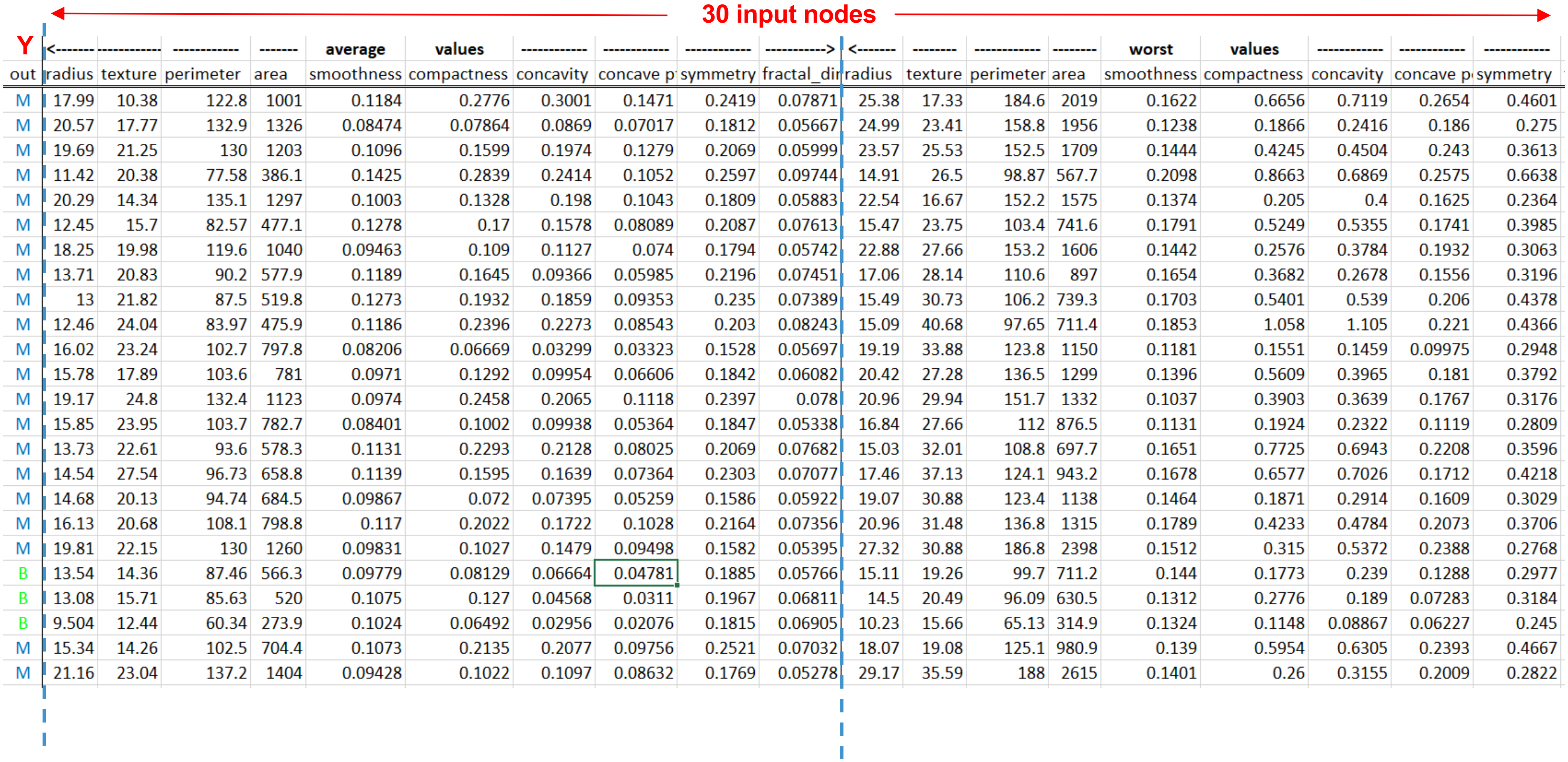
| predictions | 0 | 1 |
|---|---|---|
| y_test | | |
| 0 | 24 | 1 |
| 1 | 0 | 25 |

test set 25%

# Example 2 – MLP Cancer dataset

Analytics

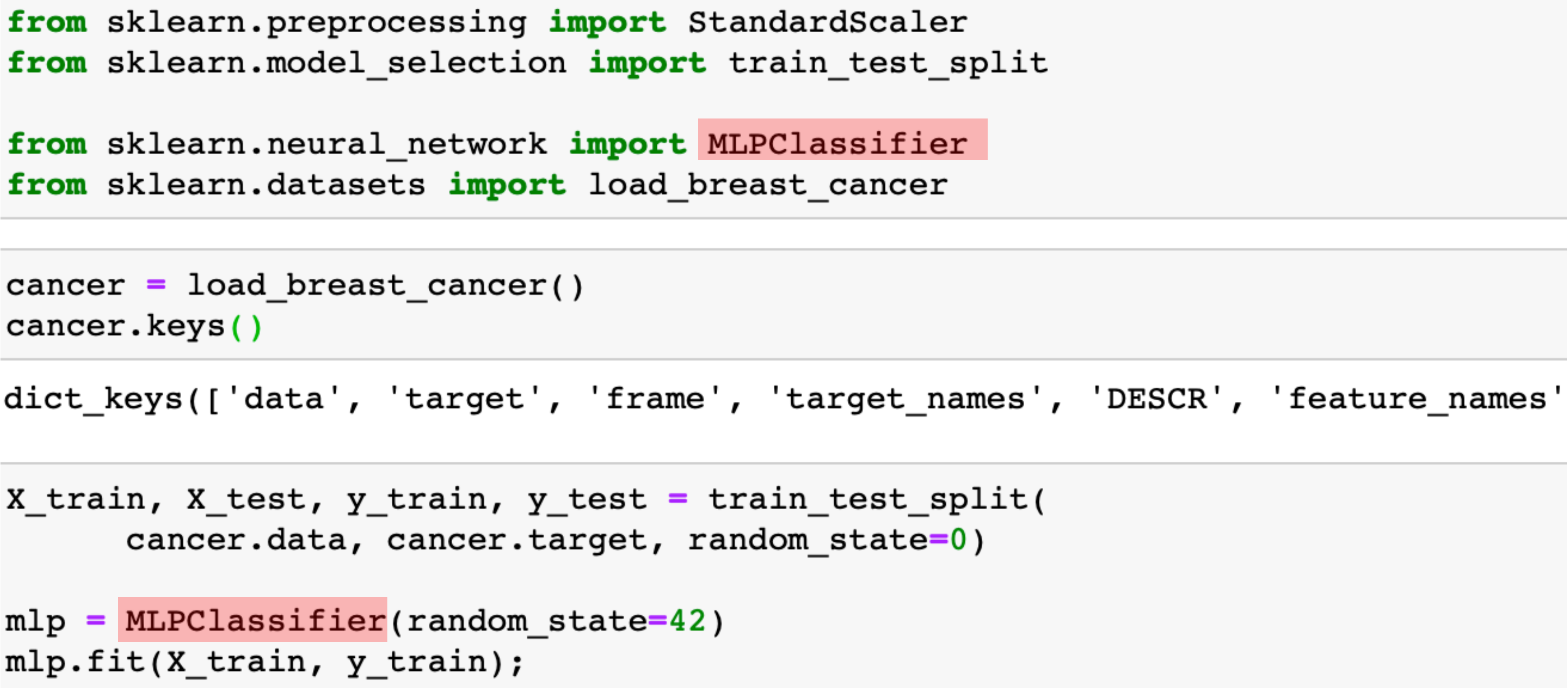
# Cancer Data



30 input nodes

| Y | <------- | ----------- | ------------ | ------- | average | values | ------------ | ------------ | ------------ | -----------> | <------- | -------- | ------------ | -------- | worst | values | ------------ | ------------ | ------------ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| out | radius | texture | perimeter | area | smoothness | compactness | concavity | concave p | symmetry | fractal_dir | radius | texture | perimeter | area | smoothness | compactness | concavity | concave p | symmetry |
| M | 17.99 | 10.38 | 122.8 | 1001 | 0.1184 | 0.2776 | 0.3001 | 0.1471 | 0.2419 | 0.07871 | 25.38 | 17.33 | 184.6 | 2019 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 |
| M | 20.57 | 17.77 | 132.9 | 1326 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 | 0.05667 | 24.99 | 23.41 | 158.8 | 1956 | 0.1238 | 0.1866 | 0.2416 | 0.186 | 0.275 |
| M | 19.69 | 21.25 | 130 | 1203 | 0.1096 | 0.1599 | 0.1974 | 0.1279 | 0.2069 | 0.05999 | 23.57 | 25.53 | 152.5 | 1709 | 0.1444 | 0.4245 | 0.4504 | 0.243 | 0.3613 |
| M | 11.42 | 20.38 | 77.58 | 386.1 | 0.1425 | 0.2839 | 0.2414 | 0.1052 | 0.2597 | 0.09744 | 14.91 | 26.5 | 98.87 | 567.7 | 0.2098 | 0.8663 | 0.6869 | 0.2575 | 0.6638 |
| M | 20.29 | 14.34 | 135.1 | 1297 | 0.1003 | 0.1328 | 0.198 | 0.1043 | 0.1809 | 0.05883 | 22.54 | 16.67 | 152.2 | 1575 | 0.1374 | 0.205 | 0.4 | 0.1625 | 0.2364 |
| M | 12.45 | 15.7 | 82.57 | 477.1 | 0.1278 | 0.17 | 0.1578 | 0.08089 | 0.2087 | 0.07613 | 15.47 | 23.75 | 103.4 | 741.6 | 0.1791 | 0.5249 | 0.5355 | 0.1741 | 0.3985 |
| M | 18.25 | 19.98 | 119.6 | 1040 | 0.09463 | 0.109 | 0.1127 | 0.074 | 0.1794 | 0.05742 | 22.88 | 27.66 | 153.2 | 1606 | 0.1442 | 0.2576 | 0.3784 | 0.1932 | 0.3063 |
| M | 13.71 | 20.83 | 90.2 | 577.9 | 0.1189 | 0.1645 | 0.09366 | 0.05985 | 0.2196 | 0.07451 | 17.06 | 28.14 | 110.6 | 897 | 0.1654 | 0.3682 | 0.2678 | 0.1556 | 0.3196 |
| M | 13 | 21.82 | 87.5 | 519.8 | 0.1273 | 0.1932 | 0.1859 | 0.09353 | 0.235 | 0.07389 | 15.49 | 30.73 | 106.2 | 739.3 | 0.1703 | 0.5401 | 0.539 | 0.206 | 0.4378 |
| M | 12.46 | 24.04 | 83.97 | 475.9 | 0.1186 | 0.2396 | 0.2273 | 0.08543 | 0.203 | 0.08243 | 15.09 | 40.68 | 97.65 | 711.4 | 0.1853 | 1.058 | 1.105 | 0.221 | 0.4366 |
| M | 16.02 | 23.24 | 102.7 | 797.8 | 0.08206 | 0.06669 | 0.03299 | 0.03323 | 0.1528 | 0.05697 | 19.19 | 33.88 | 123.8 | 1150 | 0.1181 | 0.1551 | 0.1459 | 0.09975 | 0.2948 |
| M | 15.78 | 17.89 | 103.6 | 781 | 0.0971 | 0.1292 | 0.09954 | 0.06606 | 0.1842 | 0.06082 | 20.42 | 27.28 | 136.5 | 1299 | 0.1396 | 0.5609 | 0.3965 | 0.181 | 0.3792 |
| M | 19.17 | 24.8 | 132.4 | 1123 | 0.0974 | 0.2458 | 0.2065 | 0.1118 | 0.2397 | 0.078 | 20.96 | 29.94 | 151.7 | 1332 | 0.1037 | 0.3903 | 0.3639 | 0.1767 | 0.3176 |
| M | 15.85 | 23.95 | 103.7 | 782.7 | 0.08401 | 0.1002 | 0.09938 | 0.05364 | 0.1847 | 0.05338 | 16.84 | 27.66 | 112 | 876.5 | 0.1131 | 0.1924 | 0.2322 | 0.1119 | 0.2809 |
| M | 13.73 | 22.61 | 93.6 | 578.3 | 0.1131 | 0.2293 | 0.2128 | 0.08025 | 0.2069 | 0.07682 | 15.03 | 32.01 | 108.8 | 697.7 | 0.1651 | 0.7725 | 0.6943 | 0.2208 | 0.3596 |
| M | 14.54 | 27.54 | 96.73 | 658.8 | 0.1139 | 0.1595 | 0.1639 | 0.07364 | 0.2303 | 0.07077 | 17.46 | 37.13 | 124.1 | 943.2 | 0.1678 | 0.6577 | 0.7026 | 0.1712 | 0.4218 |
| M | 14.68 | 20.13 | 94.74 | 684.5 | 0.09867 | 0.072 | 0.07395 | 0.05259 | 0.1586 | 0.05922 | 19.07 | 30.88 | 123.4 | 1138 | 0.1464 | 0.1871 | 0.2914 | 0.1609 | 0.3029 |
| M | 16.13 | 20.68 | 108.1 | 798.8 | 0.117 | 0.2022 | 0.1722 | 0.1028 | 0.2164 | 0.07356 | 20.96 | 31.48 | 136.8 | 1315 | 0.1789 | 0.4233 | 0.4784 | 0.2073 | 0.3706 |
| M | 19.81 | 22.15 | 130 | 1260 | 0.09831 | 0.1027 | 0.1479 | 0.09498 | 0.1582 | 0.05395 | 27.32 | 30.88 | 186.8 | 2398 | 0.1512 | 0.315 | 0.5372 | 0.2388 | 0.2768 |
| B | 13.54 | 14.36 | 87.46 | 566.3 | 0.09779 | 0.08129 | 0.06664 | 0.04781 | 0.1885 | 0.05766 | 15.11 | 19.26 | 99.7 | 711.2 | 0.144 | 0.1773 | 0.239 | 0.1288 | 0.2977 |
| B | 13.08 | 15.71 | 85.63 | 520 | 0.1075 | 0.127 | 0.04568 | 0.0311 | 0.1967 | 0.06811 | 14.5 | 20.49 | 96.09 | 630.5 | 0.1312 | 0.2776 | 0.189 | 0.07283 | 0.3184 |
| B | 9.504 | 12.44 | 60.34 | 273.9 | 0.1024 | 0.06492 | 0.02956 | 0.02076 | 0.1815 | 0.06905 | 10.23 | 15.66 | 65.13 | 314.9 | 0.1324 | 0.1148 | 0.08867 | 0.06227 | 0.245 |
| M | 15.34 | 14.26 | 102.5 | 704.4 | 0.1073 | 0.2135 | 0.2077 | 0.09756 | 0.2521 | 0.07032 | 18.07 | 19.08 | 125.1 | 980.9 | 0.139 | 0.5954 | 0.6305 | 0.2393 | 0.4667 |
| M | 21.16 | 23.04 | 137.2 | 1404 | 0.09428 | 0.1022 | 0.1097 | 0.08632 | 0.1769 | 0.05278 | 29.17 | 35.59 | 188 | 2615 | 0.1401 | 0.26 | 0.3155 | 0.2009 | 0.2822 |

# Example 2 – MLP on Cancer data



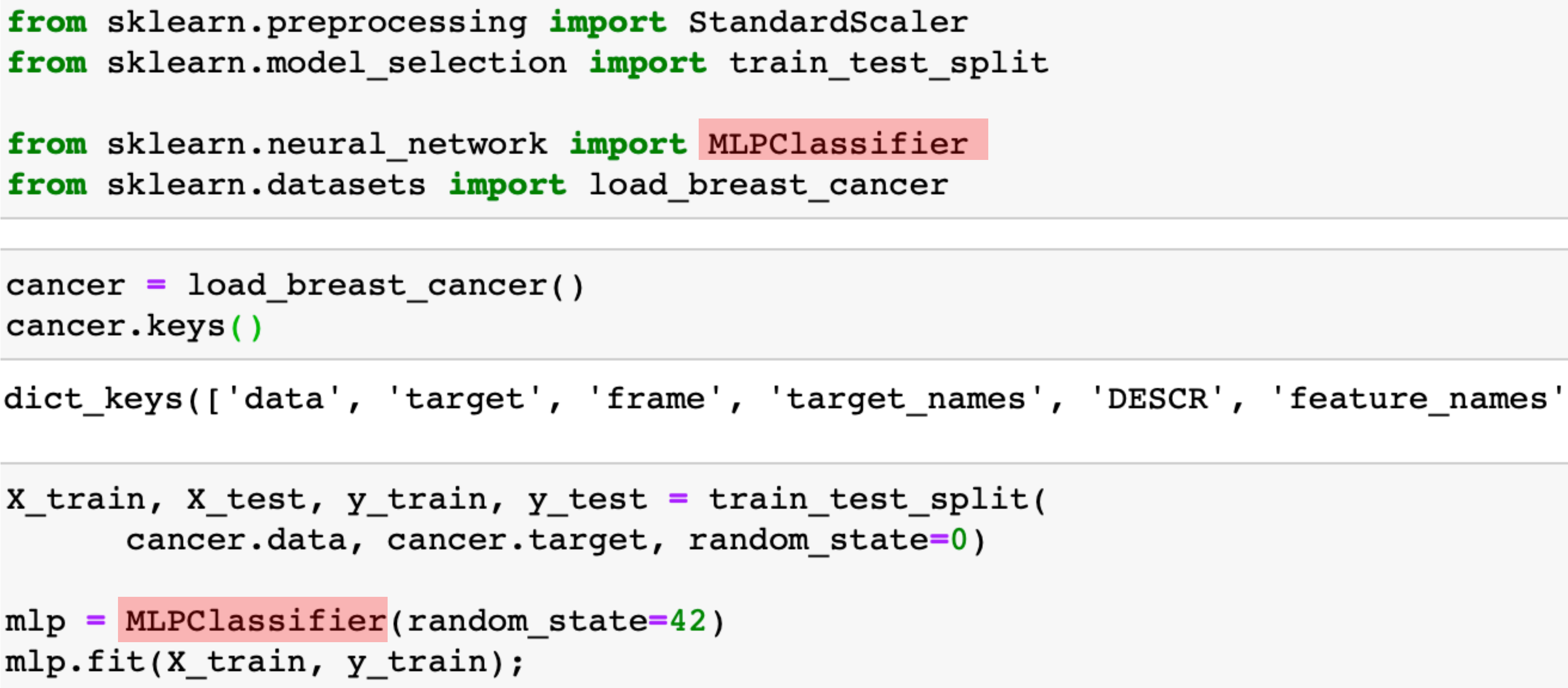
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'
```

```
X_train, X_test, y_train, y_test = train_test_split(
      cancer.data, cancer.target, random_state=0)

mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train);
```

# Example 2 – MLP on Cancer data

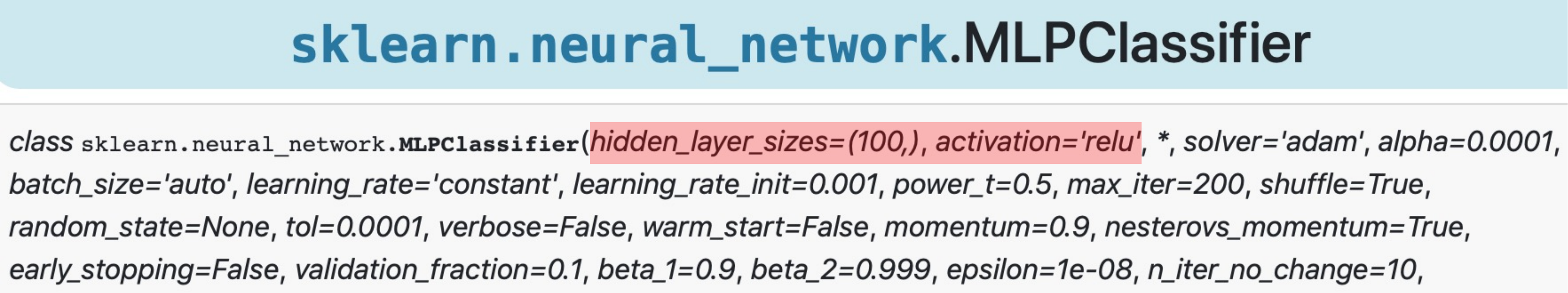


```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'
```

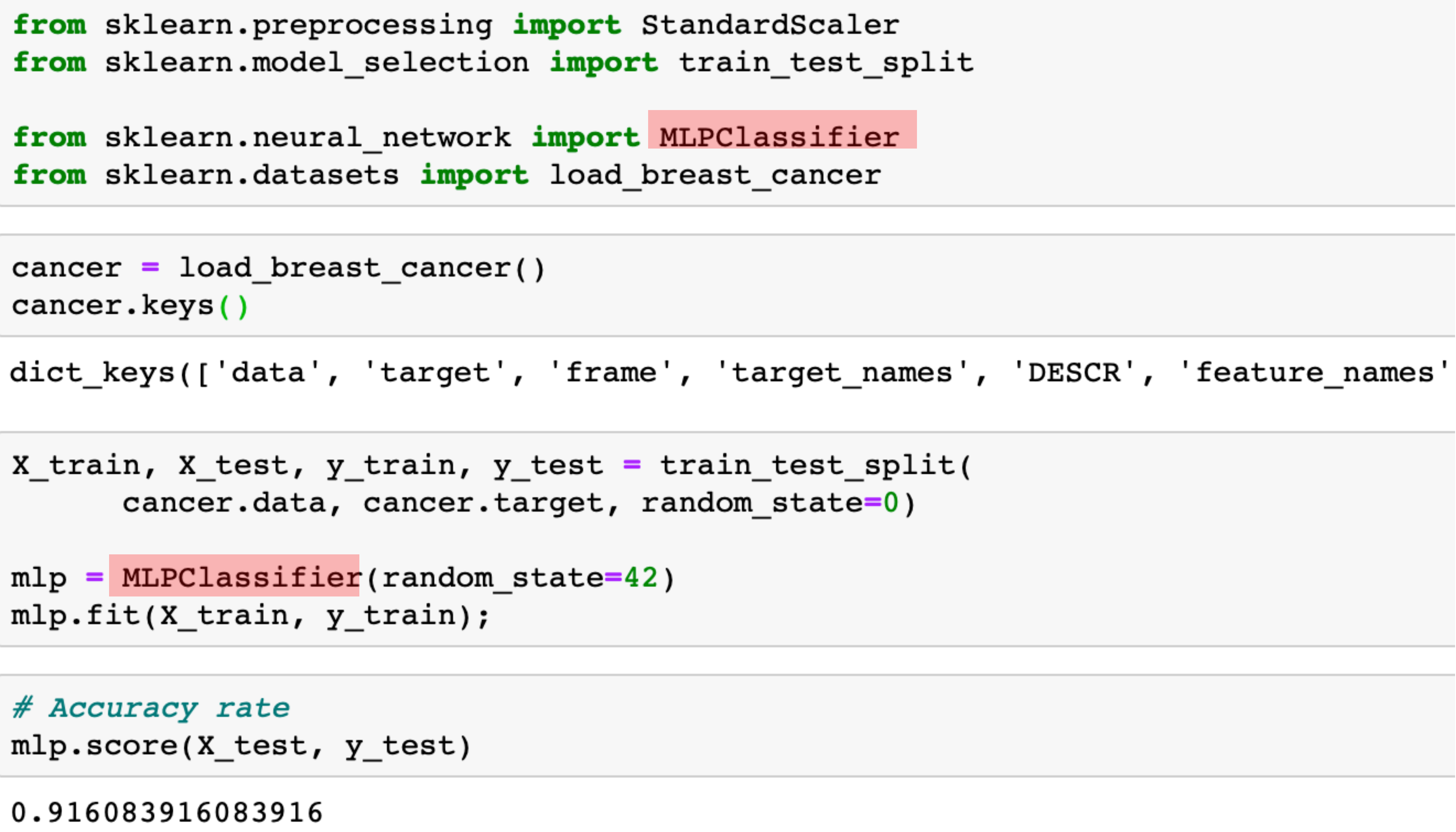
```
X_train, X_test, y_train, y_test = train_test_split(
      cancer.data, cancer.target, random_state=0)

mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train);
```



## sklearn.neural_network.MLPClassifier

*class* `sklearn.neural_network.MLPClassifier`(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10,*

# Example 2 – MLP on Cancer data



```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
cancer.keys()
```

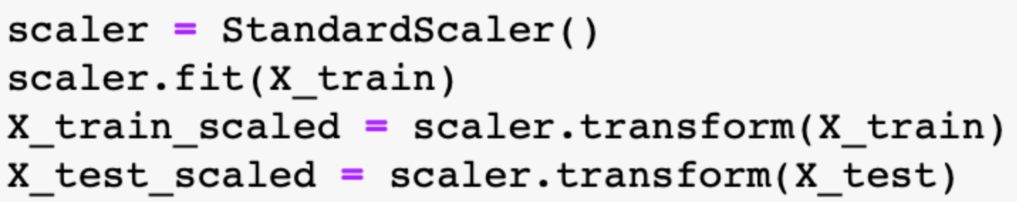
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'
```

```
X_train, X_test, y_train, y_test = train_test_split(
      cancer.data, cancer.target, random_state=0)

mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train);
```

```
# Accuracy rate
mlp.score(X_test, y_test)
```

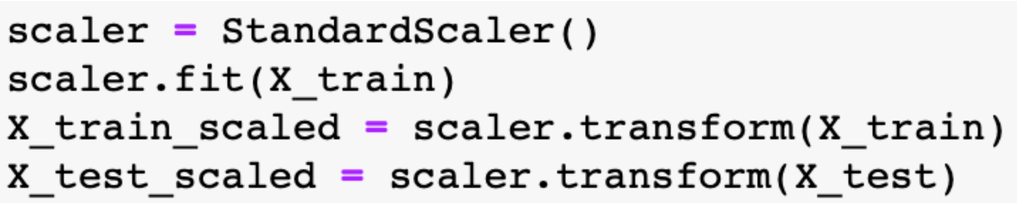
```
0.916083916083916
```

# Example 2 – Standardize the data

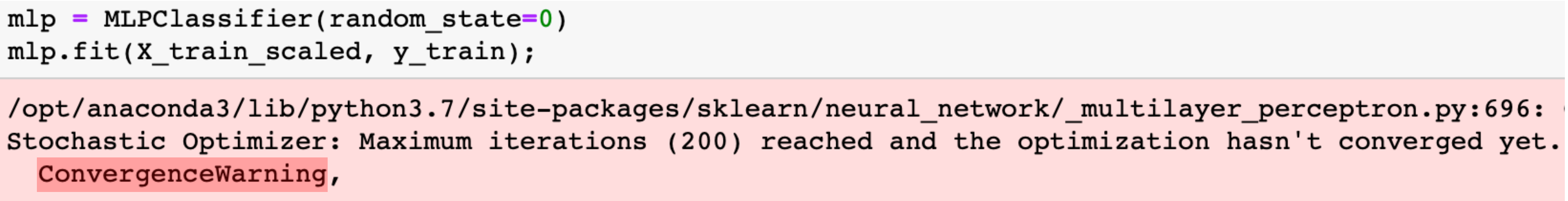


```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

# Example 2 – Standardize the data

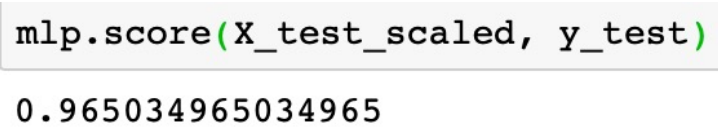


```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```



```
mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train);
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perceptron.py:696:
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  ConvergenceWarning,
```

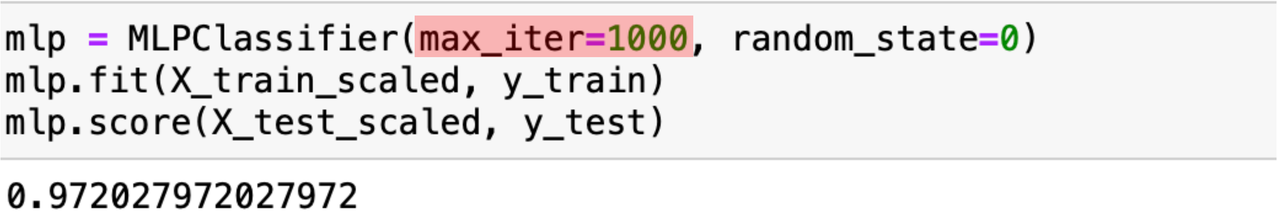


```
mlp.score(X_test_scaled, y_test)
```

```
0.965034965034965
```

# Example 2

### Increase max_iter

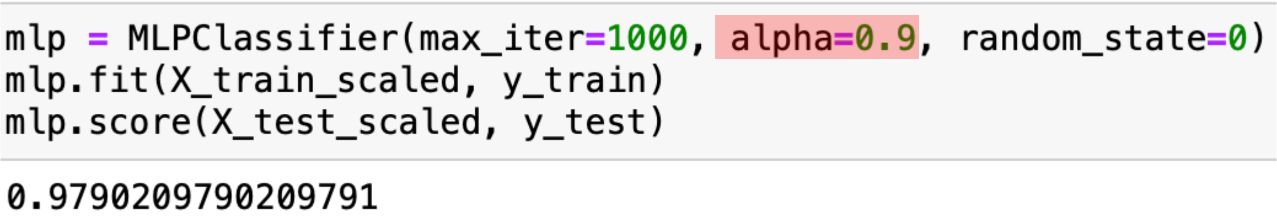


```
mlp = MLPClassifier(max_iter=1000, random_state=0)
mlp.fit(X_train_scaled, y_train)
mlp.score(X_test_scaled, y_test)
```

```
0.972027972027972
```

No Convergence Warning

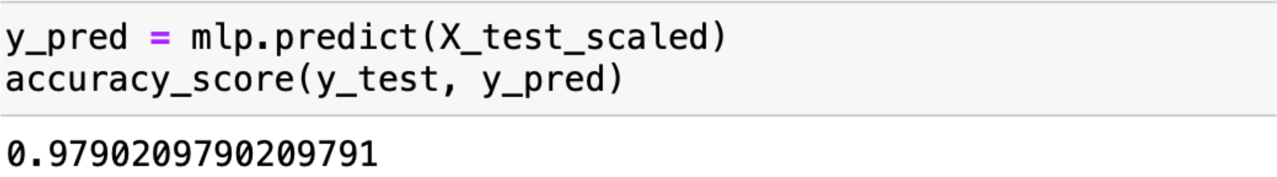
### Regularization on MLP



```
mlp = MLPClassifier(max_iter=1000, alpha=0.9, random_state=0)
mlp.fit(X_train_scaled, y_train)
mlp.score(X_test_scaled, y_test)
```

```
0.9790209790209791
```

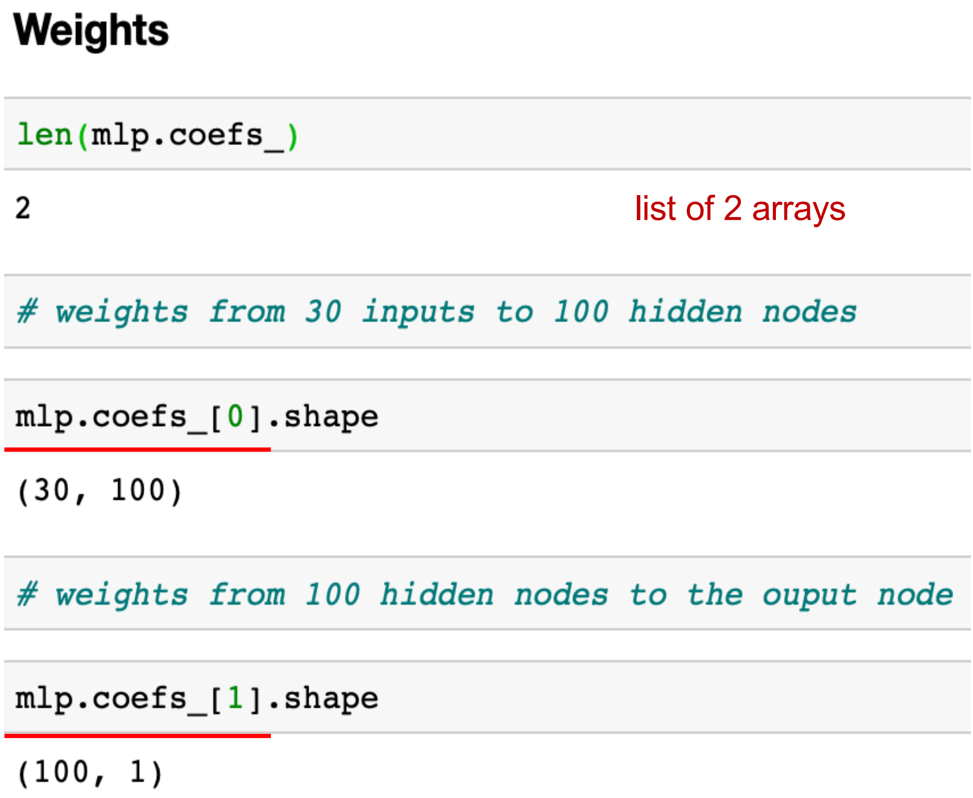
search for the best alpha



```
y_pred = mlp.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

```
0.9790209790209791
```

# Example 2 - weights



## Weights

```
len(mlp.coefs_)
```

```
2
```

list of 2 arrays

```
# weights from 30 inputs to 100 hidden nodes
```

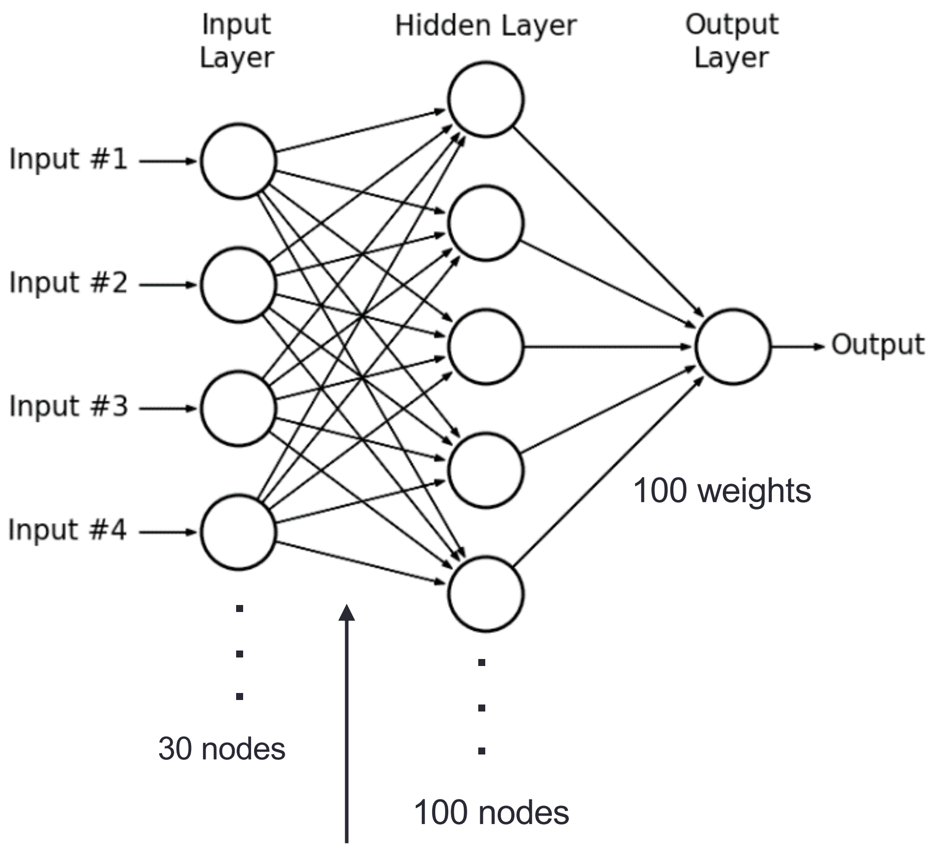
```
mlp.coefs_[0].shape
```

```
(30, 100)
```

```
# weights from 100 hidden nodes to the ouput node
```

```
mlp.coefs_[1].shape
```

```
(100, 1)
```



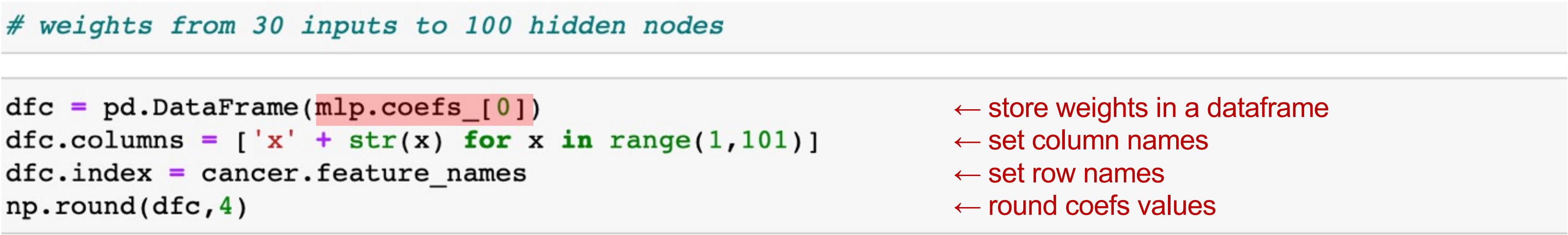
100 weights

30 nodes

100 nodes

3000 weights

# Example 2 - weights

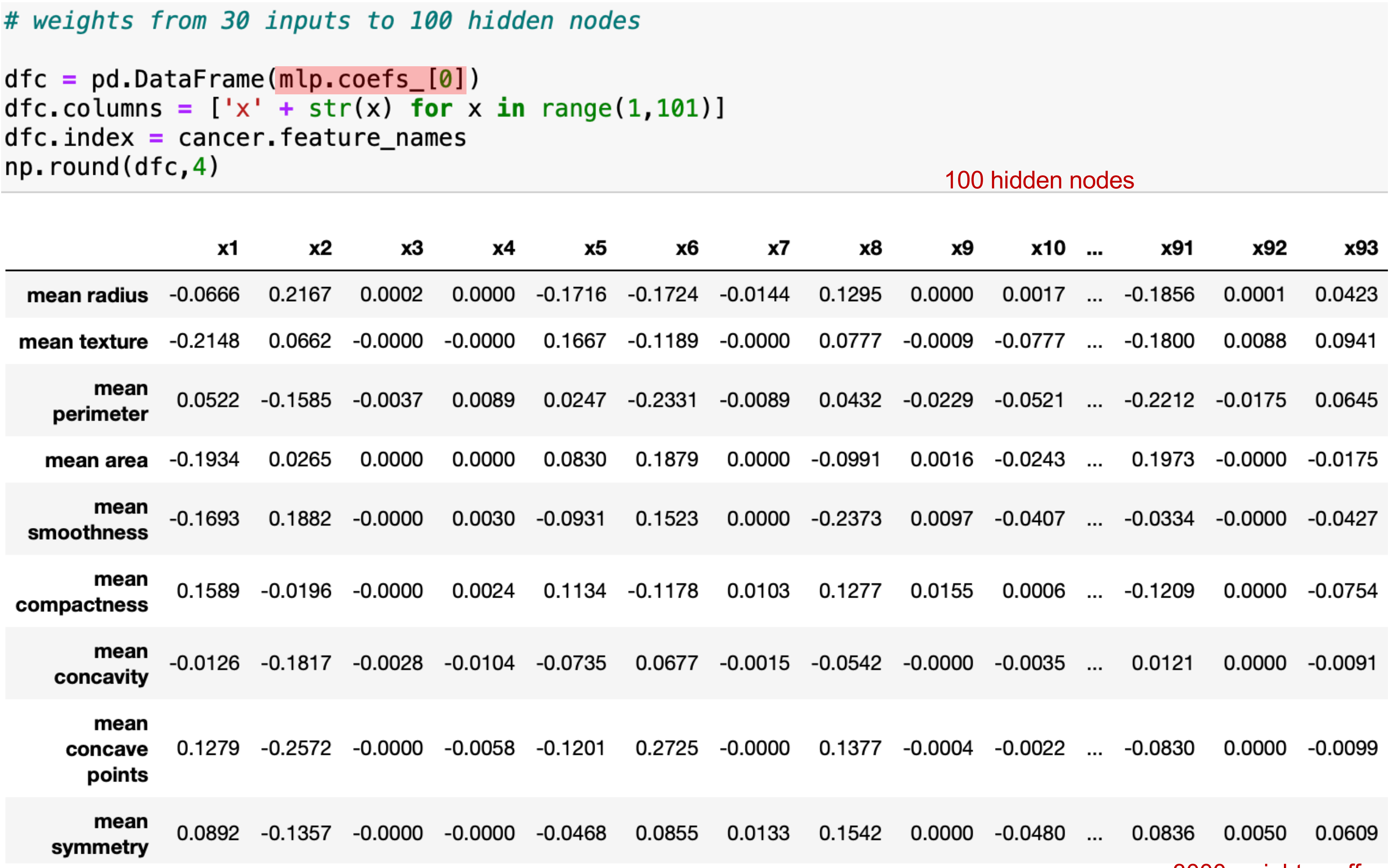


```
# weights from 30 inputs to 100 hidden nodes
```

```
dfc = pd.DataFrame(mlp.coefs_[0])
dfc.columns = ['x' + str(x) for x in range(1,101)]
dfc.index = cancer.feature_names
np.round(dfc,4)
```

← store weights in a dataframe
← set column names
← set row names
← round coefs values

# Example 2 - weights



```
# weights from 30 inputs to 100 hidden nodes

dfc = pd.DataFrame(mlp.coefs_[0])
dfc.columns = ['x' + str(x) for x in range(1,101)]
dfc.index = cancer.feature_names
np.round(dfc,4)
```

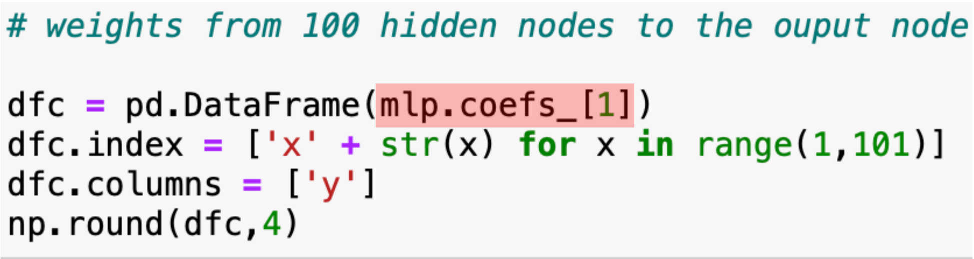
100 hidden nodes

| | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | ... | x91 | x92 | x93 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **mean radius** | -0.0666 | 0.2167 | 0.0002 | 0.0000 | -0.1716 | -0.1724 | -0.0144 | 0.1295 | 0.0000 | 0.0017 | ... | -0.1856 | 0.0001 | 0.0423 |
| **mean texture** | -0.2148 | 0.0662 | -0.0000 | -0.0000 | 0.1667 | -0.1189 | -0.0000 | 0.0777 | -0.0009 | -0.0777 | ... | -0.1800 | 0.0088 | 0.0941 |
| **mean perimeter** | 0.0522 | -0.1585 | -0.0037 | 0.0089 | 0.0247 | -0.2331 | -0.0089 | 0.0432 | -0.0229 | -0.0521 | ... | -0.2212 | -0.0175 | 0.0645 |
| **mean area** | -0.1934 | 0.0265 | 0.0000 | 0.0000 | 0.0830 | 0.1879 | 0.0000 | -0.0991 | 0.0016 | -0.0243 | ... | 0.1973 | -0.0000 | -0.0175 |
| **mean smoothness** | -0.1693 | 0.1882 | -0.0000 | 0.0030 | -0.0931 | 0.1523 | 0.0000 | -0.2373 | 0.0097 | -0.0407 | ... | -0.0334 | -0.0000 | -0.0427 |
| **mean compactness** | 0.1589 | -0.0196 | -0.0000 | 0.0024 | 0.1134 | -0.1178 | 0.0103 | 0.1277 | 0.0155 | 0.0006 | ... | -0.1209 | 0.0000 | -0.0754 |
| **mean concavity** | -0.0126 | -0.1817 | -0.0028 | -0.0104 | -0.0735 | 0.0677 | -0.0015 | -0.0542 | -0.0000 | -0.0035 | ... | 0.0121 | 0.0000 | -0.0091 |
| **mean concave points** | 0.1279 | -0.2572 | -0.0000 | -0.0058 | -0.1201 | 0.2725 | -0.0000 | 0.1377 | -0.0004 | -0.0022 | ... | -0.0830 | 0.0000 | -0.0099 |
| **mean symmetry** | 0.0892 | -0.1357 | -0.0000 | -0.0000 | -0.0468 | 0.0855 | 0.0133 | 0.1542 | 0.0000 | -0.0480 | ... | 0.0836 | 0.0050 | 0.0609 |

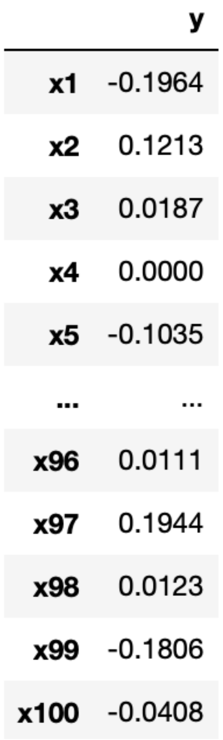
30 input nodes

3000 weight coeffs

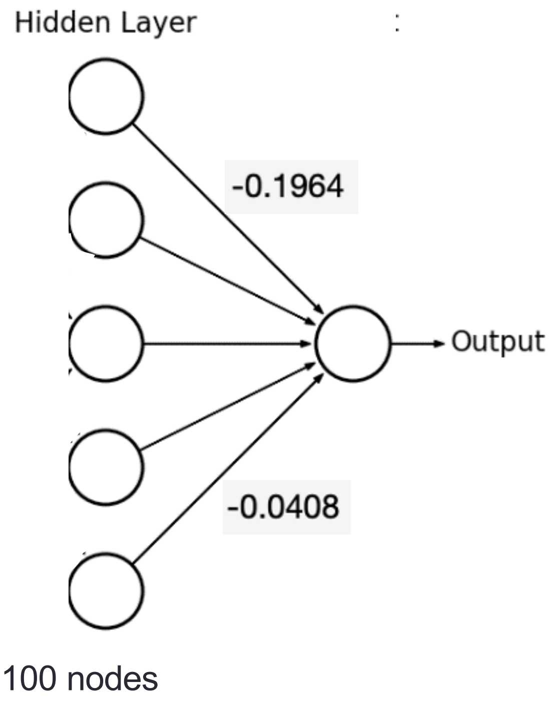
# Example 2 – weights to Output node



```
# weights from 100 hidden nodes to the ouput node

dfc = pd.DataFrame(mlp.coefs_[1])
dfc.index = ['x' + str(x) for x in range(1,101)]
dfc.columns = ['y']
np.round(dfc,4)
```



| | y |
|---|---|
| **x1** | -0.1964 |
| **x2** | 0.1213 |
| **x3** | 0.0187 |
| **x4** | 0.0000 |
| **x5** | -0.1035 |
| **...** | ... |
| **x96** | 0.0111 |
| **x97** | 0.1944 |
| **x98** | 0.0123 |
| **x99** | -0.1806 |
| **x100** | -0.0408 |


Hidden Layer
-0.1964
Output
-0.0408
100 nodes