

# Study about Neural Networks - Network

## 1. Introduction and problem design

This jupyter notebook intends to be as simpler as possible for people understand what neural networks really is.

This is the step two for students lerning networks.

This notebook have 2 neurons in the main layer and 1 neuron in the hidden layer with a custom activate function to plot the categorization.

## 2. Import of libraries ¶

In [1]:

```
import numpy as np
from sklearn import datasets as d
from sklearn import model_selection as ms

import keras
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import backend

import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
def custom_activation(x):
    """
    This function returns the linear function with a clip
    to select the data in two categories (0,1).
    """
    return keras.backend.clip(keras.activations.linear(x),0,1.1)
```

## 3. Generation of dataset using numpy and sklearn

In [3]:

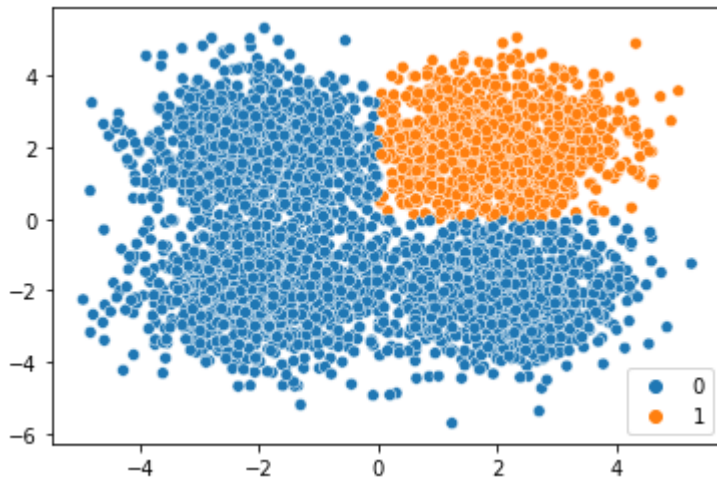
```

space = 0.2
X = np.append(
    d.make_circles(n_samples=1000, noise=0.8, factor=.9)[0] + np.array([2,2
    ]),
    d.make_circles(n_samples=1000, noise=0.8, factor=.9)[0] + np.array([-2,2
    ]),
    axis = 0)
X = np.append(X,
    d.make_circles(n_samples=1000, noise=0.8, factor=.9)[0] + np.array([2,-2
    ]),
    axis=0)
X = np.append(X,
    d.make_circles(n_samples=1000, noise=0.8, factor=.9)[0] + np.array([-2,-
    2]),
    axis=0)
y = []
for point in X:
    if point[0] > 0 and point[1] > 0:
        y.append(1)
    else:
        y.append(0)
y = np.array(y)


```

In [4]:

```
sns.scatterplot(X[:,0], X[:,1], hue=y);
```



### 3. Neural network modelling

 Illustration of Network

In [5]:

```

# Splitting the dataset in training and validation
x_train, x_val, y_train, y_val = ms.train_test_split(X, y, test_size=0.2, random
_state=1)

```

In [6]:

```
# Creating the simple linear neuron
model = Sequential()
model.add(Dense(2, activation='tanh', input_dim=2))
model.add(Dense(1, activation=custom_activation))

# Hinge loss function was best model for this problem, uncomment the others to test
model.compile(loss='Hinge')
# model.compile(loss='mean_squared_error')
# model.compile(loss='BinaryCrossentropy')
# model.compile(loss='MeanSquaredLogarithmicError')
# model.compile(loss='MeanAbsolutePercentageError')

model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=100)
```

```
Epoch 1/100
100/100 [=====] - 0s 1ms/step - loss: 0.96
71 - val_loss: 0.9470
Epoch 2/100
100/100 [=====] - 0s 530us/step - loss: 0.
9423 - val_loss: 0.9140
Epoch 3/100
100/100 [=====] - 0s 475us/step - loss: 0.
9100 - val_loss: 0.8763
Epoch 4/100
100/100 [=====] - 0s 490us/step - loss: 0.
8745 - val_loss: 0.8451
Epoch 5/100
100/100 [=====] - 0s 495us/step - loss: 0.
8450 - val_loss: 0.8208
Epoch 6/100
100/100 [=====] - 0s 531us/step - loss: 0.
8226 - val_loss: 0.8042
Epoch 7/100
100/100 [=====] - 0s 502us/step - loss: 0.
8059 - val_loss: 0.7925
Epoch 8/100
100/100 [=====] - 0s 511us/step - loss: 0.
7945 - val_loss: 0.7848
Epoch 9/100
100/100 [=====] - 0s 503us/step - loss: 0.
7865 - val_loss: 0.7795
Epoch 10/100
100/100 [=====] - 0s 494us/step - loss: 0.
7805 - val_loss: 0.7753
Epoch 11/100
100/100 [=====] - 0s 519us/step - loss: 0.
7760 - val_loss: 0.7721
Epoch 12/100
100/100 [=====] - 0s 513us/step - loss: 0.
7725 - val_loss: 0.7696
Epoch 13/100
100/100 [=====] - 0s 478us/step - loss: 0.
7702 - val_loss: 0.7678
Epoch 14/100
100/100 [=====] - 0s 481us/step - loss: 0.
7684 - val_loss: 0.7663
Epoch 15/100
100/100 [=====] - 0s 518us/step - loss: 0.
7667 - val_loss: 0.7653
Epoch 16/100
100/100 [=====] - 0s 523us/step - loss: 0.
7654 - val_loss: 0.7644
Epoch 17/100
100/100 [=====] - 0s 518us/step - loss: 0.
7643 - val_loss: 0.7633
Epoch 18/100
100/100 [=====] - 0s 480us/step - loss: 0.
7633 - val_loss: 0.7627
Epoch 19/100
100/100 [=====] - 0s 505us/step - loss: 0.
7626 - val_loss: 0.7620
Epoch 20/100
100/100 [=====] - 0s 484us/step - loss: 0.
7619 - val_loss: 0.7611
Epoch 21/100
```

```
100/100 [=====] - 0s 508us/step - loss: 0.7613 - val_loss: 0.7608
Epoch 22/100
100/100 [=====] - 0s 500us/step - loss: 0.7607 - val_loss: 0.7604
Epoch 23/100
100/100 [=====] - 0s 471us/step - loss: 0.7602 - val_loss: 0.7598
Epoch 24/100
100/100 [=====] - 0s 490us/step - loss: 0.7598 - val_loss: 0.7596
Epoch 25/100
100/100 [=====] - 0s 525us/step - loss: 0.7594 - val_loss: 0.7593
Epoch 26/100
100/100 [=====] - 0s 487us/step - loss: 0.7590 - val_loss: 0.7591
Epoch 27/100
100/100 [=====] - 0s 489us/step - loss: 0.7587 - val_loss: 0.7589
Epoch 28/100
100/100 [=====] - 0s 517us/step - loss: 0.7584 - val_loss: 0.7587
Epoch 29/100
100/100 [=====] - 0s 512us/step - loss: 0.7581 - val_loss: 0.7583
Epoch 30/100
100/100 [=====] - 0s 494us/step - loss: 0.7579 - val_loss: 0.7580
Epoch 31/100
100/100 [=====] - 0s 464us/step - loss: 0.7576 - val_loss: 0.7577
Epoch 32/100
100/100 [=====] - 0s 475us/step - loss: 0.7574 - val_loss: 0.7574
Epoch 33/100
100/100 [=====] - 0s 481us/step - loss: 0.7572 - val_loss: 0.7572
Epoch 34/100
100/100 [=====] - 0s 488us/step - loss: 0.7569 - val_loss: 0.7571
Epoch 35/100
100/100 [=====] - 0s 484us/step - loss: 0.7567 - val_loss: 0.7570
Epoch 36/100
100/100 [=====] - 0s 498us/step - loss: 0.7565 - val_loss: 0.7570
Epoch 37/100
100/100 [=====] - 0s 511us/step - loss: 0.7563 - val_loss: 0.7569
Epoch 38/100
100/100 [=====] - 0s 462us/step - loss: 0.7561 - val_loss: 0.7567
Epoch 39/100
100/100 [=====] - 0s 485us/step - loss: 0.7559 - val_loss: 0.7564
Epoch 40/100
100/100 [=====] - 0s 501us/step - loss: 0.7558 - val_loss: 0.7564
Epoch 41/100
100/100 [=====] - 0s 468us/step - loss: 0.7558 - val_loss: 0.7564
```

```
7556 - val_loss: 0.7563
Epoch 42/100
100/100 [=====] - 0s 497us/step - loss: 0.
7554 - val_loss: 0.7561
Epoch 43/100
100/100 [=====] - 0s 487us/step - loss: 0.
7553 - val_loss: 0.7560
Epoch 44/100
100/100 [=====] - 0s 501us/step - loss: 0.
7552 - val_loss: 0.7558
Epoch 45/100
100/100 [=====] - 0s 498us/step - loss: 0.
7550 - val_loss: 0.7556
Epoch 46/100
100/100 [=====] - 0s 455us/step - loss: 0.
7549 - val_loss: 0.7556
Epoch 47/100
100/100 [=====] - 0s 516us/step - loss: 0.
7548 - val_loss: 0.7555
Epoch 48/100
100/100 [=====] - 0s 479us/step - loss: 0.
7547 - val_loss: 0.7551
Epoch 49/100
100/100 [=====] - 0s 483us/step - loss: 0.
7546 - val_loss: 0.7549
Epoch 50/100
100/100 [=====] - 0s 519us/step - loss: 0.
7545 - val_loss: 0.7547
Epoch 51/100
100/100 [=====] - 0s 496us/step - loss: 0.
7544 - val_loss: 0.7547
Epoch 52/100
100/100 [=====] - 0s 502us/step - loss: 0.
7543 - val_loss: 0.7546
Epoch 53/100
100/100 [=====] - 0s 489us/step - loss: 0.
7543 - val_loss: 0.7545
Epoch 54/100
100/100 [=====] - 0s 492us/step - loss: 0.
7542 - val_loss: 0.7545
Epoch 55/100
100/100 [=====] - 0s 509us/step - loss: 0.
7541 - val_loss: 0.7543
Epoch 56/100
100/100 [=====] - 0s 508us/step - loss: 0.
7540 - val_loss: 0.7543
Epoch 57/100
100/100 [=====] - 0s 515us/step - loss: 0.
7540 - val_loss: 0.7542
Epoch 58/100
100/100 [=====] - 0s 494us/step - loss: 0.
7539 - val_loss: 0.7542
Epoch 59/100
100/100 [=====] - 0s 506us/step - loss: 0.
7539 - val_loss: 0.7541
Epoch 60/100
100/100 [=====] - 0s 496us/step - loss: 0.
7538 - val_loss: 0.7538
Epoch 61/100
100/100 [=====] - 0s 521us/step - loss: 0.
7538 - val_loss: 0.7537
```

```
Epoch 62/100
100/100 [=====] - 0s 454us/step - loss: 0.
7537 - val_loss: 0.7536
Epoch 63/100
100/100 [=====] - 0s 448us/step - loss: 0.
7537 - val_loss: 0.7536
Epoch 64/100
100/100 [=====] - 0s 505us/step - loss: 0.
7536 - val_loss: 0.7535
Epoch 65/100
100/100 [=====] - 0s 484us/step - loss: 0.
7536 - val_loss: 0.7534
Epoch 66/100
100/100 [=====] - 0s 469us/step - loss: 0.
7535 - val_loss: 0.7534
Epoch 67/100
100/100 [=====] - 0s 501us/step - loss: 0.
7535 - val_loss: 0.7533
Epoch 68/100
100/100 [=====] - 0s 483us/step - loss: 0.
7535 - val_loss: 0.7532
Epoch 69/100
100/100 [=====] - 0s 520us/step - loss: 0.
7534 - val_loss: 0.7530
Epoch 70/100
100/100 [=====] - 0s 515us/step - loss: 0.
7534 - val_loss: 0.7530
Epoch 71/100
100/100 [=====] - 0s 486us/step - loss: 0.
7533 - val_loss: 0.7529
Epoch 72/100
100/100 [=====] - 0s 481us/step - loss: 0.
7533 - val_loss: 0.7528
Epoch 73/100
100/100 [=====] - 0s 480us/step - loss: 0.
7533 - val_loss: 0.7527
Epoch 74/100
100/100 [=====] - 0s 489us/step - loss: 0.
7533 - val_loss: 0.7528
Epoch 75/100
100/100 [=====] - 0s 458us/step - loss: 0.
7533 - val_loss: 0.7527
Epoch 76/100
100/100 [=====] - 0s 522us/step - loss: 0.
7532 - val_loss: 0.7527
Epoch 77/100
100/100 [=====] - 0s 522us/step - loss: 0.
7532 - val_loss: 0.7526
Epoch 78/100
100/100 [=====] - 0s 511us/step - loss: 0.
7532 - val_loss: 0.7524
Epoch 79/100
100/100 [=====] - 0s 498us/step - loss: 0.
7532 - val_loss: 0.7523
Epoch 80/100
100/100 [=====] - 0s 498us/step - loss: 0.
7531 - val_loss: 0.7522
Epoch 81/100
100/100 [=====] - 0s 492us/step - loss: 0.
7531 - val_loss: 0.7521
Epoch 82/100
```

```
100/100 [=====] - 0s 487us/step - loss: 0.7531 - val_loss: 0.7520
Epoch 83/100
100/100 [=====] - 0s 519us/step - loss: 0.7531 - val_loss: 0.7519
Epoch 84/100
100/100 [=====] - 0s 509us/step - loss: 0.7530 - val_loss: 0.7520
Epoch 85/100
100/100 [=====] - 0s 485us/step - loss: 0.7530 - val_loss: 0.7520
Epoch 86/100
100/100 [=====] - 0s 477us/step - loss: 0.7530 - val_loss: 0.7519
Epoch 87/100
100/100 [=====] - 0s 509us/step - loss: 0.7530 - val_loss: 0.7518
Epoch 88/100
100/100 [=====] - 0s 485us/step - loss: 0.7529 - val_loss: 0.7518
Epoch 89/100
100/100 [=====] - 0s 504us/step - loss: 0.7529 - val_loss: 0.7516
Epoch 90/100
100/100 [=====] - 0s 470us/step - loss: 0.7529 - val_loss: 0.7516
Epoch 91/100
100/100 [=====] - 0s 489us/step - loss: 0.7529 - val_loss: 0.7515
Epoch 92/100
100/100 [=====] - 0s 505us/step - loss: 0.7529 - val_loss: 0.7516
Epoch 93/100
100/100 [=====] - 0s 514us/step - loss: 0.7528 - val_loss: 0.7516
Epoch 94/100
100/100 [=====] - 0s 493us/step - loss: 0.7528 - val_loss: 0.7516
Epoch 95/100
100/100 [=====] - 0s 459us/step - loss: 0.7528 - val_loss: 0.7516
Epoch 96/100
100/100 [=====] - 0s 522us/step - loss: 0.7528 - val_loss: 0.7516
Epoch 97/100
100/100 [=====] - 0s 497us/step - loss: 0.7528 - val_loss: 0.7516
Epoch 98/100
100/100 [=====] - 0s 513us/step - loss: 0.7528 - val_loss: 0.7515
Epoch 99/100
100/100 [=====] - 0s 483us/step - loss: 0.7527 - val_loss: 0.7515
Epoch 100/100
100/100 [=====] - 0s 506us/step - loss: 0.7527 - val_loss: 0.7515
```

Out[6]:

<tensorflow.python.keras.callbacks.History at 0x7f7af063c4c0>



## 4. Exploring the weights of the neuron

In [7]:

```
#Test points
pto_test_x = 3
pto_test_y = 5
```

In [8]:

```
# Prediction of a point with the help of model
model.predict(np.array([[pto_test_x,pto_test_y]]))
```

Out[8]:

```
array([[1.1]], dtype=float32)
```

In [9]:

```
# Calculating the prediction with the weight of the linear neuron

a = np.tanh(
    pto_test_x * model.layers[0].get_weights()[0][0][0] +\
    pto_test_y * model.layers[0].get_weights()[0][1][0] +\
    model.layers[0].get_weights()[1][0]
)

b = np.tanh(
    pto_test_x * model.layers[0].get_weights()[0][0][1] +\
    pto_test_y * model.layers[0].get_weights()[0][1][1] +\
    model.layers[0].get_weights()[1][1]
)

c = np.clip(
    a * model.layers[1].get_weights()[0][0][0] +\
    b * model.layers[1].get_weights()[0][1][0] +\
    model.layers[1].get_weights()[1][0],
    0,1.1)
c
```

Out[9]:

```
1.1
```

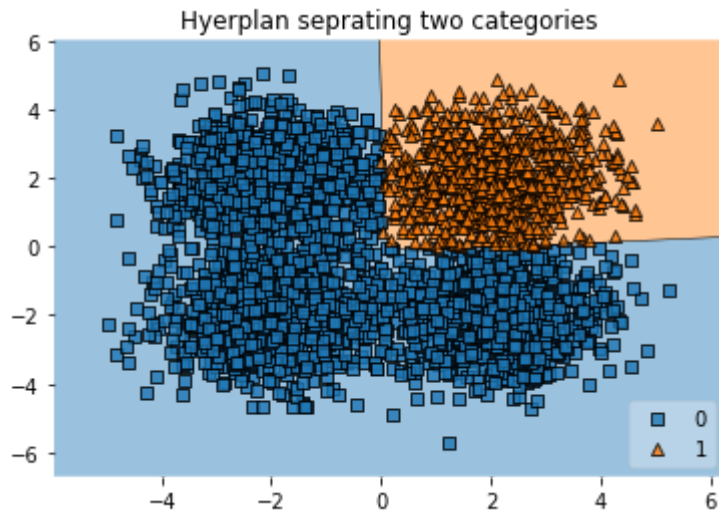
## 5. Plotting the line of the linear neuron

In [10]:

```
plot_decision_regions(x_train, y_train, clf=model, legend=4)
plt.title('Hyerplan seprating two categories')
```

Out[10]:

Text(0.5, 1.0, 'Hyerplan seprating two categories')



## 6. Conclusion

This notebook implemented a more complex neural network to help students.

Three neurons were needed in order to separate the data in two categories.