Import biblioteki **TensorFlow** (https://www.tensorflow.org/) z której będziemy korzystali w **uczeniu maszynowym**:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

Dwa gangi

Zbiór danych:

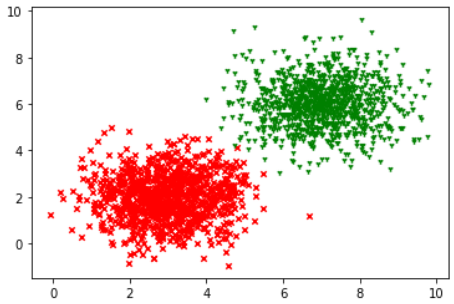
```
[0]*10+[1]*10
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
x_label1 = np.random.normal(3, 1, 1000)
y_label1 = np.random.normal(2, 1, 1000)
x_label2 = np.random.normal(7, 1, 1000)
y_label2 = np.random.normal(6, 1, 1000)

xs = np.append(x_label1, x_label2)
ys = np.append(y_label1, y_label2)
labels = np.asarray([0.]*len(x_label1)+[1.]*len(x_label2))
labels
```



```
plt.scatter(x_label1, y_label1, c='r', marker='x', s=20)
plt.scatter(x_label2, y_label2, c='g', marker='1', s=20)
plt.show()
```




```
x_label1
```

```
2.68307868,  4.66273859,  3.36915147,  1.98293959,  2.62546685,
4.38539606,  4.69224183,  2.70834409,  2.63050344,  4.29729179,
2.11068124,  3.31990076,  2.73190657,  4.42589322,  2.72154997,
1.88285419,  2.89986418,  0.17934489,  2.95267495,  4.98433754,
2.90472819,  3.31329914,  2.98437242,  2.5931381 ,  3.35257714,
2.90127548,  3.96984306,  3.95331981,  4.74830588,  3.43173024,
4.3212225 ,  3.31536219,  3.64267684,  1.6769249 ,  4.70460446,
2.22904672,  4.23417066,  4.6323563 ,  2.63640651,  3.50687867,
4.20654705,  3.19363574,  3.50712794,  2.69277101,  4.60447728,
2.73262265,  2.14817253,  3.47049923,  3.03099951,  4.60620622,
4.38278362,  3.23772916,  3.15260483,  1.37510451,  1.50559682,
3.54658476,  2.73032468,  3.07066198,  1.73078664,  2.71534584,
2.83098612,  2.71005905,  2.66244132,  4.64124423,  1.63807643,
4.5498686 ,  1.43707989,  2.46149256,  1.69856217,  4.13959843,
3.48856384,  1.66367636,  3.69322501,  2.7989094 ,  3.17405849,
2.58013837,  3.92031694,  3.27884964,  3.40461   ,  4.70748007,
3.32179367,  2.86280426,  4.09735689,  3.56811499,  4.62437907,
3.40736751,  2.44840248,  3.12513496, -0.07375846,  2.01613616,
3.8303686 ,  1.74364273,  0.85276857,  4.85970435,  2.33714171,
2.96131982,  3.25069684,  2.90714052,  4.54742469,  4.07463251,
4.01939775,  3.18159087,  3.18116798,  1.92418221,  4.06937852,
3.58502439,  0.76774536,  0.52259387,  3.1006132 ,  1.23437259,
3.55543964,  3.27628947,  2.98520558,  1.76266393,  3.60021679,
2.22100027,  1.83004998,  2.5429399 ,  3.15729264,  2.90020762,
2.81370189,  2.40006658,  2.49962699,  4.69822861,  3.49281562,
4.27300193,  4.93379125,  2.3882269 ,  2.92293099,  2.67670661,
2.83823787,  1.76455179,  1.59921429,  0.60812177,  1.99921687,
4.00467006,  1.43840405,  3.23292122,  2.10218422,  2.7097794 ,
4.55193037,  2.72390946,  2.6946714 ,  4.04147539,  3.07284839,
1.06476285,  2.52817839,  2.77333626,  2.98842434,  1.78602818,
3.02536624,  1.29963508,  3.46177439,  1.03129639,  2.20804341,
3.86582095,  1.36675552,  2.99531477,  3.22944971,  4.19387593,
4.06451662,  2.16846767,  3.46402366,  3.28468374,  3.36670877,
1.54549211,  2.23190878,  3.51359142,  1.42613142,  3.69896487,
3.35878163,  3.0947897 ,  3.27284904,  3.74822455,  2.07367887,
2.38344506,  2.07880178,  3.49308984,  2.73586549,  3.03744338,
4.63865202,  1.96616214,  3.15910445,  0.74188525,  3.04701975,
```



```
3.97140497,  4.08502155,  3.3137523 ,  2.10554432,  3.09525294,
1.44129969,  3.82412892,  3.15582287,  2.82231236,  4.29993777,
2.40399629,  1.8853618 ,  5.47153464,  2.66554613,  3.1490699 ,
2.91657195,  2.62321185,  3.19174617,  3.05483917,  2.45486673,
1.27580971,  4.65512679,  2.20071859,  3.18293678,  0.48474966,
1.20937417,  3.28080456,  2.93684535,  3.43832951,  2.2430044 ,
4.41541676,  4.68239371,  3.6195759 ,  3.56977441,  1.06340113,
2.77204101,  3.7541242 ,  2.68120843,  2.04617664,  2.02061831,
2.45594325,  3.80997177,  2.10385677,  5.27360727,  4.63158968,
3.73004285,  4.08221056,  2.3485268 ,  2.97900196,  2.76974282,
3.25635983,  1.97318706,  4.02327473,  4.39769915,  2.08396979,
2.87123474,  2.08132664,  2.52983145,  2.95689346,  3.35196994,
1.04971167,  1.72262994,  3.32835746,  2.44423178,  4.27446416,
3.04001015,  2.43647428,  2.51869679,  3.10030902,  3.36692991])
```

Definiujemy model:

```
model = Sequential()
```

Dodajemy **jedną warstwę** (Dense) z **jednym neuronem** (units=1) z **biasem** (use_bias=True) i **liniową funkcją aktywacji** (activation="linear"):

```
model.add(Dense(units = 1, use_bias=True, input_dim=2, activation = "sigmoid"))
```

Definiujemy **optymalizator** i **błąd** (entropia krzyżowa). **Współczynnik uczenia = 0.1**

```
#opt = tf.keras.optimizers.Adam(learning_rate=0.1)
opt = tf.keras.optimizers.SGD(learning_rate=0.2)


model.compile(loss='binary_crossentropy',optimizer=opt)
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_1 (Dense)             (None, 1)                 3         

=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

Przygotowanie danych:

```
xs=xs.reshape(-1,1)
ys=ys.reshape(-1,1)
data_points=np.concatenate([xs,ys],axis=1)
data_points
```



```
[5.97215790, 1.44306685],
[4.30283564, 1.42523501],
...,
[7.61607157, 7.0249111 ],
[7.26977568, 5.194599  ],
[6.77230854, 4.13097325]])
```

Proces **uczenia**:

```
epochs = 100
h = model.fit(data_points,labels, verbose=1, epochs=epochs,validation_split=0.2)
```

```
50/50 [==============================] - 0s 2ms/step - loss: 0.0227 - val_loss: 0.0227
Epoch 79/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0225 - val_loss: 0.0220
Epoch 80/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0223 - val_loss: 0.0216
Epoch 81/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0223 - val_loss: 0.0205
Epoch 82/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0221 - val_loss: 0.0187
Epoch 83/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0219 - val_loss: 0.0221
Epoch 84/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0217 - val_loss: 0.0202
Epoch 85/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0215 - val_loss: 0.0246
Epoch 86/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0214 - val_loss: 0.0224
Epoch 87/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0214 - val_loss: 0.0226
Epoch 88/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0209 - val_loss: 0.0180
Epoch 89/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0211 - val_loss: 0.0253
Epoch 90/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0207 - val_loss: 0.0202
Epoch 91/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0205 - val_loss: 0.0255
Epoch 92/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0205 - val_loss: 0.0175
Epoch 93/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0203 - val_loss: 0.0184
Epoch 94/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0202 - val_loss: 0.0218
Epoch 95/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0201 - val_loss: 0.0217
Epoch 96/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0200 - val_loss: 0.0196
```



```
50/50 [==============================] - 0s 2ms/step - loss: 0.0196 - val_loss: 0.0258
Epoch 99/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0197 - val_loss: 0.0195
Epoch 100/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0194 - val_loss: 0.0226
```

```
Loss = h.history['loss']
Loss
```

```
       0.026617188006639 48,
       0.026431255042552948,
       0.026116378605365753,
       0.025844631716609,
       0.02549923025071621,
       0.02523864060640335,
       0.025040175765752792,
       0.02489537186920643,
       0.02473808079957962,
       0.024468781426548958,
       0.0238710418343544,
       0.024013342335820198,
       0.023657433688640594,
       0.023582186549901962,
       0.023412542417645454,
       0.023208389058709145,
       0.023003580048680305,
       0.022825945168733597,
       0.022693239152431488,
       0.02245958149433136,
       0.02227034419775009,
       0.022262707352638245,
       0.022060463204979897,
       0.021900251507759094,
       0.021668661385774612,
       0.021532952785491943,
       0.02139592356979847,
       0.02137177623808384,
       0.020941562950611115,
       0.021102702245116234,
       0.020746057853102684,
       0.020524682477116585,
       0.020520441234111786,
       0.020281152799725533,
       0.02019719034433365,
       0.020052066072821617,
       0.020046480000019073,
```



```
       0.01942899264395237]
```

Sprawdźmy jakie są **wartości wag**:

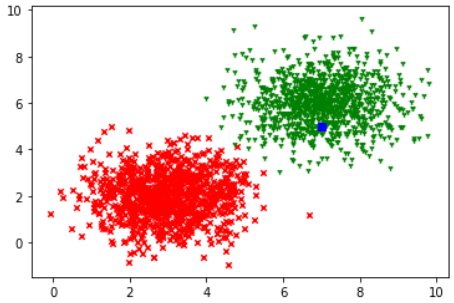
```
weights = model.get_weights()

print(weights[0])
print(weights[1])    #bias
```

```
    [[1.3021898]
     [1.5386865]]
    [-12.82827]
```

```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```

Sprawdzamy działanie modelu dla punktu o współrzędnych **x** i **y**:

```
x=7.0
y=5.0
plt.scatter(x_label1, y_label1, c='r', marker='x', s=20)
plt.scatter(x_label2, y_label2, c='g', marker='1', s=20)
plt.scatter(x,y,c='b', marker='s')
plt.show()
```




```
model.predict([[x,y]])
```



```
array([[0.9816659]], dtype=float32)
```

✓ 0 s ukończono o 15:57

Nie udało się automatycznie zapisać pliku. Został on zaktualizowany zdalnie lub na innej karcie. Pokaż porównanie