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
```

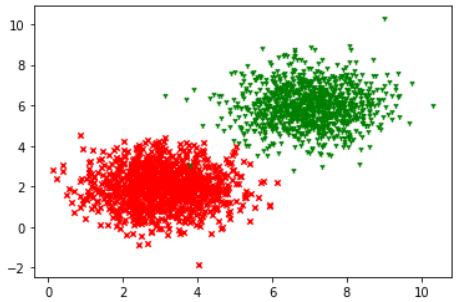
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
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
```

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

```
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
```

```
4.95152671, 2.40821936, 3.51079647, 1.96872238, 5.75956663,
4.53517287, 4.77770474, 3.41662156, 3.41728895, 2.15779038,
2.41865166, 3.6831147 , 1.98482885, 1.43419942, 3.01812046,
3.66770989, 2.97869613, 2.30656168, 4.12275169, 0.56066787,
0.67244036, 2.18912334, 4.54258359, 2.3617773 , 2.50423735,
3.9281843 , 4.31827964, 2.9222936 , 3.33287082, 2.13473671,
2.62669564, 2.02817061, 2.67712723, 3.59858596, 3.73791171,
3.81455659, 3.86882924, 2.22924474, 2.5082117 , 2.76997283,
3.10904214, 3.32400813, 3.01981044, 2.85531866, 3.70929031,
6.12878677, 4.65799285, 2.15282142, 3.95678094, 1.35304783,
4.79139754, 3.12229169, 3.19429355, 2.5986723 , 2.15296881,
3.49715881, 1.37333645, 2.19337582, 1.76768332, 2.18423302,
3.11164807, 3.18391077, 2.80409489, 2.77838651, 4.33561539,
1.57546306, 2.64319114, 2.34713732, 1.81671495, 2.66573849,
5.1290117 , 1.72260062, 2.9115866 , 2.28655203, 4.40278178,
3.15240078, 2.9102068 , 3.90325671, 3.01635406, 2.76628566,
2.69004472, 1.98829948, 2.72301974, 1.66705455, 3.16505564,
3.64105684, 3.73568156, 4.10676832, 1.09087972, 4.27816478,
4.7289813 , 4.60353768, 3.3168271 , 4.15717026, 3.44661978,
2.27002143, 1.63625995, 3.58567169, 3.02589852, 2.73721771,
2.45465004, 3.59321535, 3.62410355, 3.43389678, 2.33701224,
3.98805233, 3.06917366, 1.14016512, 3.88070876, 3.73534311,
2.7770648 , 4.96505716, 3.34480422, 3.70975608, 4.0235651 ,
0.90542053, 3.63648473, 3.82631205, 1.05780602, 2.43228945,
3.77601675, 3.83893403, 3.36512599, 3.50002278, 3.62799659,
4.02849401, 3.89883921, 3.69361274, 3.24108284, 4.2863234 ,
4.46433464, 2.59725307, 4.21499313, 2.94826971, 1.38165871,
3.09823941, 2.38470244, 4.54780988, 2.78487094, 3.35349723,
3.02625907, 2.15437944, 1.87796708, 3.29142614, 1.74663638,
3.38648327, 1.27550336, 2.88665644, 3.64141671, 4.4844536 ,
3.62114456, 4.1721245 , 4.28819642, 2.10517459, 3.54197436,
4.46446202, 3.9015839 , 2.4485798 , 2.72769614, 1.69335876,
1.55352287, 2.45832177, 1.49953876, 4.2162915 , 3.55516684,
2.25810543, 0.91946784, 2.12294756, 2.33392389, 2.29269809,
2.83254371, 2.6856634 , 3.32801813, 3.65765585, 4.38395713,
4.23904304, 2.9218446 , 2.69875885, 3.14240576, 5.44556061,
5.1539476 , 4.17500351, 2.95958089, 2.69800979, 3.79161154,
```



```
1.82919354, 2.82255109, 2.66384788, 1.81149835, 4.57659221,
1.01744853, 2.792855  , 3.6124966 , 2.55487007, 3.32992518,
3.77619849, 3.29494843, 4.33929467, 3.56766464, 3.17117069,
2.67170641, 3.65704795, 4.40615631, 3.20836534, 3.10226461,
3.87543917, 2.20956548, 2.82610015, 3.87929185, 1.86578015,
3.4803039 , 3.79978953, 3.06767452, 2.74779509, 3.25350248,
0.93784327, 1.80174359, 1.90534586, 5.11055602, 2.82007891,
2.80053744, 5.02206221, 2.44964918, 4.8725375 , 2.34769506,
4.61003953, 3.09310146, 1.31012589, 3.60728716, 2.80805041,
1.6188026 , 2.57330517, 3.64303806, 2.63045598, 4.79084283,
3.5747279 , 1.74789419, 1.16056031, 3.18029079, 2.35442838,
2.44788003, 2.84159785, 3.92568424, 4.69724664, 3.71710243,
3.13956629, 4.23022639, 3.41101018, 2.53088974, 3.61623056,
1.99720892, 2.92847637, 3.84991539, 4.04204492, 4.01977085])
```

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 = 3, use_bias=True, input_dim=2, activation = "sigmoid"))
model.add(Dense(units = 1, use bias=True, 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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_2 (Dense)             (None, 3)                 9         

 dense_3 (Dense)             (None, 1)                 4         

=================================================================
Total params: 13
Trainable params: 13
Non-trainable params: 0
_________________________________________________________________
```

Przygotowanie danych:

```
xs=xs.reshape(-1,1)
```



```
data_points
```

```
array([[1.19276022, 3.79214336],
       [0.68060322, 0.71113941],
       [2.25956428, 1.27320275],
       ...,
       [7.25829277, 7.28787535],
       [6.78714613, 6.88925493],
       [5.80465651, 5.11747271]])
```

Proces **uczenia**:

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

```
Epoch 77/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0125 - val_loss: 0.0089
Epoch 78/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0127 - val_loss: 0.0163
Epoch 79/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0127 - val_loss: 0.0090
Epoch 80/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0125 - val_loss: 0.0166
Epoch 81/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0126 - val_loss: 0.0109
Epoch 82/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0122 - val_loss: 0.0159
Epoch 83/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0126 - val_loss: 0.0104
Epoch 84/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0120 - val_loss: 0.0132
Epoch 85/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0122 - val_loss: 0.0212
Epoch 86/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0117 - val_loss: 0.0090
Epoch 87/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0121 - val_loss: 0.0095
Epoch 88/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0122 - val_loss: 0.0151
Epoch 89/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0114 - val_loss: 0.0120
Epoch 90/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0117 - val_loss: 0.0122
Epoch 91/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0114 - val_loss: 0.0077
Epoch 92/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0118 - val_loss: 0.0154
Epoch 93/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0112 - val_loss: 0.0123
Epoch 94/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0119 - val_loss: 0.0160
Epoch 95/100
```



```
Epoch 97/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0109 - val_loss: 0.0168
Epoch 98/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0116 - val_loss: 0.0101
Epoch 99/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0114 - val_loss: 0.0127
Epoch 100/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0108 - val_loss: 0.0156
```

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

```
       0.015434540808200836,
       0.01503483671694994,
       0.015518811531364918,
       0.01490419264882803,
       0.0146859809756279,
       0.015032423660159111,
       0.015054848976433277,
       0.014906716533005238,
       0.014214523136615753,
       0.014282374642789364,
       0.014198966324329376,
       0.013451698236167431,
       0.013693264685571194,
       0.013481770642101765,
       0.01289574708789587,
       0.013047036714851856,
       0.013333308781385422,
       0.012957963161170483,
       0.013497598469257355,
       0.013051323592662811,
       0.01253355760127306,
       0.012691779062151909,
       0.012694892473518848,
       0.012450719252228737,
       0.012648934498429298,
       0.012213226407766342,
       0.012619555927813053,
       0.012036140076816082,
       0.012211564928293228,
       0.011707433499395847,
       0.01207730919122696,
       0.01219948846846819,
       0.011438356712460518,
       0.011686568148434162,
       0.011375256814062595,
       0.011806829832494259,
       0.011178749613463879,
```

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

```
       0.010926077142357826,
       0.011568348854780197,
       0.011356642469763756,
       0.010768054984509945]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

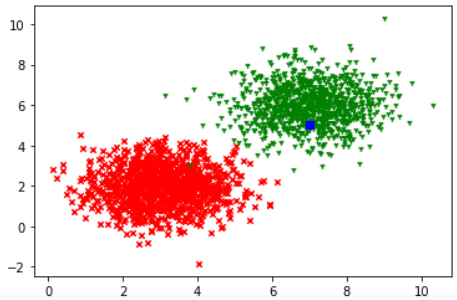
```
    [[-0.6585891  -0.29898286  0.4724988 ]
     [-0.70752895 -0.49309435  0.56131923]]
    [ 6.0124297  2.8032994 -4.2576523]
```

```
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**:

0 20 40 60 80 100

```
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]])
```

```
1/1 [==============================] - 0s 56ms/step
array([[0.9924081]], dtype=float32)
```

✓ 0 s ukończono o 15:59

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