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
```

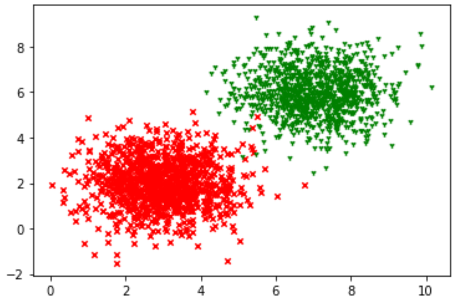
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
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
```

```
array([0., 0., 0., ..., 1., 1., 1.])
```

```
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
```

```
       2.52743031, 3.17723035, 3.00153551, 2.73917826, 1.96451522,
       4.36098214, 2.86495491, 2.40235077, 5.14683774, 1.37198408,
       3.2178151 , 5.58753596, 2.47188638, 3.83013983, 2.67525327,
       3.65574575, 3.70525976, 2.57336829, 4.49608305, 2.82454399,
       3.78092117, 2.93652839, 1.64395489, 3.11566513, 1.261612  ,
       3.54229872, 2.4753733 , 3.11982878, 1.03742137, 4.74848365,
       3.77774231, 1.78683156, 3.32611294, 4.48370006, 4.47664778,
       0.54815056, 3.2590739 , 2.79712207, 1.98236237, 1.9368531 ,
       3.05405244, 3.3126317 , 4.31447892, 3.78308775, 3.82455903,
       3.02709461, 3.42351827, 3.07153746, 3.49306413, 2.44046066,
       3.53548602, 5.50943478, 2.25308377, 4.80236835, 2.68225832,
       1.61367159, 2.14596803, 1.80034444, 3.17462426, 2.52334495,
       4.81850833, 3.39578242, 2.83140127, 1.86925321, 2.84300091,
       3.27764455, 3.83436808, 4.05080324, 2.4645345 , 4.29462806,
       2.98269872, 3.66022462, 2.54188187, 3.74458556, 2.2929538 ,
       3.67509851, 1.83562906, 3.22416369, 6.77321507, 2.35153146,
       3.76006384, 3.00864763, 2.08378824, 2.44847592, 3.0704999 ,
       3.59087763, 2.77686523, 2.99772775, 3.84512077, 2.31214938,
       4.214887  , 3.04382508, 2.0843742 , 2.08051969, 2.78265569,
       2.14363738, 3.94154907, 3.07690683, 1.86020723, 4.36939729,
       1.62883093, 1.73699028, 1.98530006, 3.46088698, 2.0888241 ,
       2.77629027, 3.59105088, 3.46393118, 2.99829967, 1.39027451,
       5.40072231, 3.61290302, 1.91172798, 4.1934188 , 1.14195586,
       2.32631352, 2.82522696, 3.50137121, 2.60405373, 4.21033992,
       3.50047727, 4.5556547 , 2.22087576, 2.92212922, 1.6812057 ,
       3.42340473, 2.56103066, 1.21324806, 3.25015984, 1.53185982,
       2.93373286, 3.19731302, 2.64878223, 2.06666106, 4.26419172,
       2.77927365, 3.78849716, 2.68138835, 3.49790415, 4.26449132,
       3.13019963, 4.64637945, 2.20124353, 2.92066688, 4.70203584,
       2.92146079, 2.90588111, 3.38655102, 3.38607622, 2.62035866,
       3.39307531, 3.76156901, 2.69905071, 2.78123475, 2.87627387,
       3.03887868, 3.80304475, 1.59299388, 2.72701164, 2.19217474,
       1.91166478, 3.94343358, 3.88919752, 2.41825063, 1.34047562,
       2.28136456, 2.04152004, 2.23771793, 1.65016746, 3.02689615,
       2.85799487, 2.62022887, 2.37653615, 3.68574345, 1.6229937 ,
       2.34112741, 3.10358907, 2.95606769, 3.25080902, 2.62707538,
       3.12136378, 3.66410145, 3.39333876, 3.45057606, 2.3631955 ,
       2.34525198, 1.89379372, 2.87212479, 3.65722747, 2.95246425,
       4.1603826 , 3.45982503, 3.42745016, 3.08334646, 3.38899123,
       4.6938705 , 5.04012875, 3.39874415, 3.77963663, 2.11962162,
       3.08636255, 1.53774397, 2.7873578 , 2.60252082, 3.69410055,
       3.91520228, 2.80758032, 3.49392584, 1.90242889, 2.75282638,
       2.33650028, 2.28054685, 2.85373651, 3.79411893, 2.31882358,
       0.33566681, 3.79628544, 2.52979986, 2.76238884, 1.68315059,
       4.69127792, 4.69408791, 3.68561701, 1.9349074 , 4.19962104,
       2.84696832, 0.98746965, 3.09898792, 3.03506228, 2.64631455,
       2.57175263, 1.86865728, 2.32815783, 4.73748266, 3.29360576,
       3.07326768, 2.71172262, 3.80774556, 2.62438469, 2.69452793,
       4.69179714, 2.47495356, 4.09600947, 4.6058617 , 2.38477617,
       4.2593486 , 2.08723834, 3.73274513, 2.50705999, 5.37825189,
       3.7177498 , 3.23642455, 4.35155769, 3.08312675, 2.52975544,
       4.29658116, 1.77829144, 3.06202364, 2.85341111, 2.4299152 ,
       0.38614052, 2.22690369, 2.09388546, 2.11203689, 5.23514622,
       4.8906394 , 0.81781098, 1.79499394, 2.9801654 , 0.94714323])
```

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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 3)                 9

 dense_5 (Dense)             (None, 1)                 4

=================================================================
Total params: 13
Trainable params: 13
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
```

```
array([[3.07155032, 2.78934244],
       [1.67765835, 0.59381758],
       [2.4212774 , 0.1332204 ],
       ...,
       [7.6669042 , 7.69793382],
       [7.54677438, 6.70901347],
       [7.67694209, 6.85666224]])
```

Proces **uczenia**:

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

```
Epoch 76/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0130 - val_loss: 0.0044
Epoch 77/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0150 - val_loss: 0.0128
Epoch 78/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0100 - val_loss: 0.0084
Epoch 79/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0098 - val_loss: 0.0036
Epoch 80/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0136 - val_loss: 0.0304
Epoch 81/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0125 - val_loss: 0.0030
Epoch 82/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0115 - val_loss: 0.0396
Epoch 83/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0129 - val_loss: 0.0149
Epoch 84/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0132 - val_loss: 0.0186
Epoch 85/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0130 - val_loss: 0.0092
Epoch 86/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0116 - val_loss: 0.0543
Epoch 87/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0120 - val_loss: 0.0543
Epoch 88/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0165 - val_loss: 0.0146
Epoch 89/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0121 - val_loss: 0.0100
Epoch 90/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0106 - val_loss: 0.0050
Epoch 91/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0148 - val_loss: 0.0051
Epoch 92/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0144 - val_loss: 0.0532
Epoch 93/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0138 - val_loss: 0.0170
Epoch 94/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0089 - val_loss: 0.0158
Epoch 95/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0224 - val_loss: 0.0076
Epoch 96/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0118 - val_loss: 0.0205
Epoch 97/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0147 - val_loss: 0.0148
Epoch 98/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0128 - val_loss: 0.0030
Epoch 99/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0174 - val_loss: 0.0753
Epoch 100/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0140 - val_loss: 0.0175
```

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

```
       0.014665761962532997,
       0.012546411715447903,
       0.01432284340262413,
       0.01784893311560154,
       0.019479958340525627,
       0.017260048538446426,
       0.010706406086683273,
       0.008852238766849041,
       0.011579795740544796,
       0.014203536324203014,
       0.011646116152405739,
       0.01031790766865015,
       0.015467925928533077,
       0.02750273235142231,
       0.01100177876651287,
       0.012858688831329346,
       0.012469500303268433,
       0.011326916515827179,
       0.016902901232242584,
       0.015573987737298012,
       0.010669786483049393,
       0.013028305023908615,
       0.014993441291153431,
       0.00999538041651249,
       0.009809046052396297,
       0.013637780211865902,
       0.012474835850298405,
       0.011495785787701607,
       0.01290920376777649,
       0.0131621602922678,
       0.012967715039849281,
       0.011610596440732479,
       0.01204559113830328,
       0.016497014090418816,
       0.012069068849086761,
       0.010618417523801327,
       0.014848730526864529,
       0.014429502189159393,
       0.013782104477286339,
       0.008900203742086887,
       0.02237713895738125,
       0.011819989420473576,
       0.014726122841238976,
       0.012798476964235306,
       0.017359983175992966,
       0.014048492535948753]
```

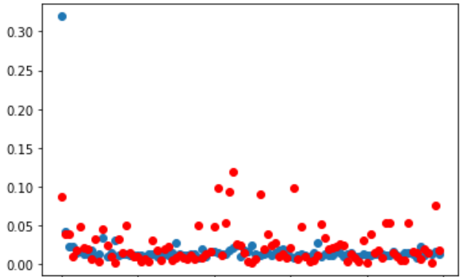
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

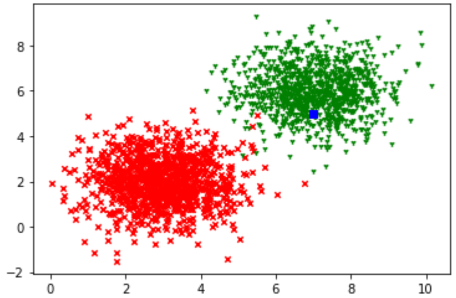
```
[[-1.7424707  2.5061858 -1.760079 ]
 [-1.675316   5.261621  -1.8686308]]
[15.291124   1.2085257 16.8492   ]
```

```
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]])
```

```
1/1 [==============================] - 0s 55ms/step
array([[0.9989512]], dtype=float32)
```

0 s ukończono o 16:01