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:

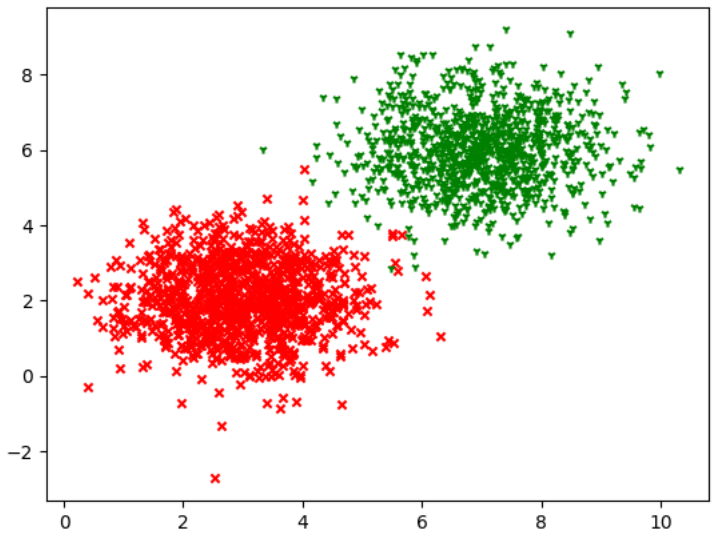
```
[0]*10+[1]*10
```

```
[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.,1.]]*len(x_label1)+[[1.,0.]]*len(x_label2))
labels
```

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




```
x_label1
```

```
3.87886923, 3.86215713, 3.78974762, 2.73719413, 3.13089114,
4.07671312, 2.71468241, 2.23360323, 3.63851583, 3.68034329,
2.04623076, 4.59463796, 3.86400724, 5.23642544, 3.36821133,
2.8552132 , 1.52293104, 2.48259079, 3.34945691, 2.8507916 ,
1.59114875, 3.51453219, 2.77199231, 3.29018717, 3.89907215,
2.23131389, 2.99308827, 2.60663754, 2.45822235, 1.51707369,
1.4600306 , 3.71696232, 3.42229088, 3.5066215 , 2.22353605,
3.84048511, 3.4551797 , 4.94187816, 2.45779482, 2.90091683,
1.61556413, 4.85334547, 1.42780502, 3.52322192, 4.20187272,
3.28675803, 1.85462733, 3.55493032, 2.87360385, 3.05451162,
1.79042789, 3.48289039, 2.7982702 , 1.30603178, 3.19990752,
3.72729122, 2.53159905, 6.12996699, 3.25421852, 5.429128  ,
0.82824977, 3.21712102, 1.95044149, 4.32362033, 2.89840644,
4.2582576 , 4.49703198, 4.24685434, 5.13507639, 1.39039144,
2.93276789, 3.01966454, 5.50334281, 2.42411897, 3.36577491,
2.13231258, 3.05201565, 3.67086862, 4.43540369, 5.58864158,
2.87608961, 3.63726434, 4.59021132, 2.36711955, 3.45126739,
4.84044628, 3.67766773, 3.9441564 , 2.90093583, 4.4765614 ,
2.47426638, 2.77659874, 4.41215273, 3.74263678, 3.94081188,
4.22061497, 3.23159642, 4.37080023, 2.14000748, 3.6492005 ,
4.59091546, 3.79038732, 1.86251265, 4.68374449, 3.77041866,
2.70346625, 0.89394567, 4.8835898 , 3.13381693, 3.84414017,
3.22957559, 3.89008652, 4.02931323, 3.55565132, 3.21377489,
4.17093645, 5.16911184, 4.75351774, 1.83087135, 3.11376335,
2.45591691, 4.50606539, 3.45904994, 3.01984548, 2.10250601,
4.07732938, 2.4602521 , 4.14556211, 2.50709155, 3.58208884,
3.41216956, 2.73096589, 2.00644878, 3.68027027, 3.59324188,
2.95503545, 3.61006472, 2.40179237, 2.53176958, 4.48328972,
1.16673587, 2.48719734, 3.80700549, 1.68990923, 3.09472397,
3.82021496, 2.17773859, 4.65583339, 3.5305347 , 2.91101156,
0.82329487, 3.85796237, 3.45517163, 3.80527095, 3.58759223,
3.85461629, 2.50918581, 3.66237354, 3.34853729, 3.35469545,
2.31965286, 3.56618443, 3.41368493, 1.93124819, 4.00474797,
3.65039349, 2.40627647, 2.97714686, 3.03506765, 3.0011046 ,
3.61610176, 2.29010309, 1.68183333, 2.69211045, 3.57623553,
3.96259631, 3.13520271, 3.45149866, 3.15555842, 3.01728707,
4.19514612, 4.42170028, 1.3837745 , 2.09794636, 2.49880642,
2.74600983, 4.44571015, 4.06800734, 3.584281  , 0.8398605 ,
6.30885576, 2.13054495, 2.82435992, 3.28445823, 2.98998178,
2.95512032, 3.50726404, 3.56531124, 1.54844588, 1.10069834,
2.30275786, 1.60888788, 3.08935652, 1.91701237, 2.3357318 ,
1.55520047, 2.15749901, 3.10391568, 2.17061499, 3.45477974,
2.90369802, 2.52122182, 3.59189685, 4.00978929, 3.83080237,
3.18487219, 4.64537933, 1.7159808 , 3.63448956, 3.02341381,
3.65839927, 1.32276084, 2.61158   , 4.88937132, 2.46232984,
2.20072446, 2.52056971, 3.38320116, 1.83845798, 3.4646569 ])
```

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)
```

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

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

```
Epoch 72/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0596 - val_loss: 0.0598
Epoch 73/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0590 - val_loss: 0.0638
Epoch 74/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0583 - val_loss: 0.0687
Epoch 75/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0579 - val_loss: 0.0712
Epoch 76/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0573 - val_loss: 0.0574
Epoch 77/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0568 - val_loss: 0.0659
Epoch 78/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0562 - val_loss: 0.0614
Epoch 79/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0557 - val_loss: 0.0644
Epoch 80/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0553 - val_loss: 0.0564
Epoch 81/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0548 - val_loss: 0.0608
Epoch 82/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0543 - val_loss: 0.0572
Epoch 83/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0539 - val_loss: 0.0586
Epoch 84/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0534 - val_loss: 0.0549
Epoch 85/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0530 - val_loss: 0.0571
Epoch 86/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0526 - val_loss: 0.0602
Epoch 87/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0521 - val_loss: 0.0536
Epoch 88/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0516 - val_loss: 0.0498
Epoch 89/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0514 - val_loss: 0.0564
Epoch 90/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0510 - val_loss: 0.0587
Epoch 91/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0506 - val_loss: 0.0612
Epoch 92/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0500 - val_loss: 0.0484
Epoch 93/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0499 - val_loss: 0.0536
Epoch 94/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0494 - val_loss: 0.0513
Epoch 95/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0491 - val_loss: 0.0528
Epoch 96/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0487 - val_loss: 0.0552
Epoch 97/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0484 - val_loss: 0.0480
Epoch 98/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0480 - val_loss: 0.0597
Epoch 99/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0477 - val_loss: 0.0495
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0474 - val_loss: 0.0554
```

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

```
        0.07855037599802017,
        0.07730505615472794,
        0.07634053379297256,
        0.07496247440576553,
        0.07428767532110214,
        0.07322156429290771,
        0.07219693809747696,
        0.071380116045475,
        0.07044018059968948,
        0.06962387263774872,
        0.0686832144856453,
        0.06797654926776886,
        0.06703279167413712,
        0.0662742331624031,
        0.06560489535331726,
        0.0647490993142128,
        0.0640266016125679,
        0.0632423460483551,
        0.06279438734054565,
        0.06215343996882439,
        0.06136888638138771,
        0.06074590981006622,
        0.060138050466775894,
        0.059602368623018265,
        0.05898134782910347,
        0.058319639414548874,
        0.05788402259349823,
        0.05725295841693878,
        0.05677634850144386,
        0.0562136285007,
        0.055725522339344025,
        0.05526250973343849,
        0.054812364280223846,
        0.05430470034480095,
        0.05391914024949074,
        0.05340442806482315,
        0.05302376300096512,
        0.052638884633779526,
        0.05210435017943382,
        0.05163104459643364,
        0.05139570310711861,
        0.050956111401319504,
        0.05057628080248833,
        0.05004867538809776,
        0.0498836524784565,
        0.0494043342769146,
        0.04907752200961113,
        0.048704762011766434,
        0.04837605357170105,
        0.047954630106687546,
        0.0477200411260128,
        0.04742835462093353]
```

Sprawdźmy jakie są **wartości wag**:

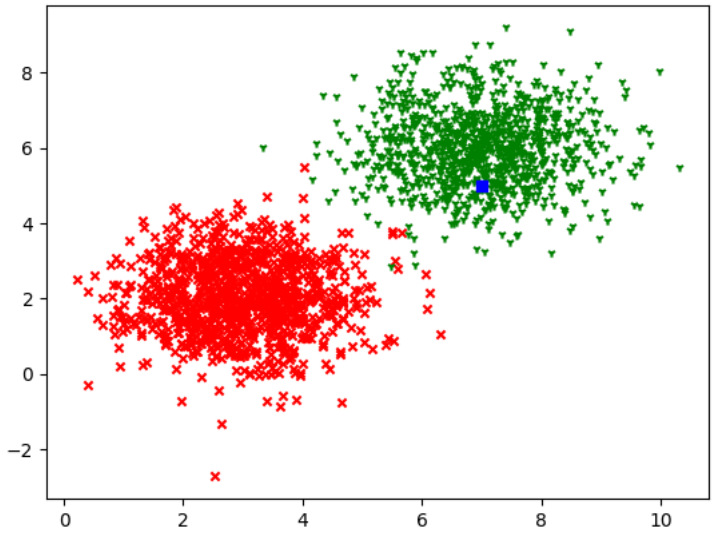
```
weights = model.get_weights()

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

```
[[ 0.8162037  -0.81870586]
 [ 1.1267896  -1.1290156 ]]
[-8.6111145  8.633041 ]
```

```
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 71ms/step
array([[0.9958448 , 0.00415518]], dtype=float32)
```

Learning rate 0.01

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.01)
```

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
```

```
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

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

```
Epoch 72/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2914 - val_loss: 0.2201
Epoch 73/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2890 - val_loss: 0.2278
Epoch 74/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2868 - val_loss: 0.2203
Epoch 75/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2847 - val_loss: 0.2275
Epoch 76/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2825 - val_loss: 0.2278
Epoch 77/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2804 - val_loss: 0.2244
Epoch 78/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2783 - val_loss: 0.2285
Epoch 79/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2764 - val_loss: 0.2176
Epoch 80/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2743 - val_loss: 0.2199
Epoch 81/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2723 - val_loss: 0.2136
Epoch 82/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2703 - val_loss: 0.2201
Epoch 83/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2684 - val_loss: 0.2131
Epoch 84/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2665 - val_loss: 0.2113
Epoch 85/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2645 - val_loss: 0.2165
Epoch 86/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2627 - val_loss: 0.2067
Epoch 87/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2609 - val_loss: 0.2003
Epoch 88/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2592 - val_loss: 0.2008
Epoch 89/100
50/50 [==============================] - 0s 2ms/step - loss: 0.2574 - val_loss: 0.2006
Epoch 90/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2556 - val_loss: 0.2032
Epoch 91/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2539 - val_loss: 0.2047
Epoch 92/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2522 - val_loss: 0.1988
Epoch 93/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2506 - val_loss: 0.1990
Epoch 94/100
50/50 [==============================] - 0s 4ms/step - loss: 0.2489 - val_loss: 0.2004
Epoch 95/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2473 - val_loss: 0.2038
Epoch 96/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2457 - val_loss: 0.1998
Epoch 97/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2441 - val_loss: 0.1964
Epoch 98/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2425 - val_loss: 0.1972
Epoch 99/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2410 - val_loss: 0.1966
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.2395 - val_loss: 0.1944
```

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

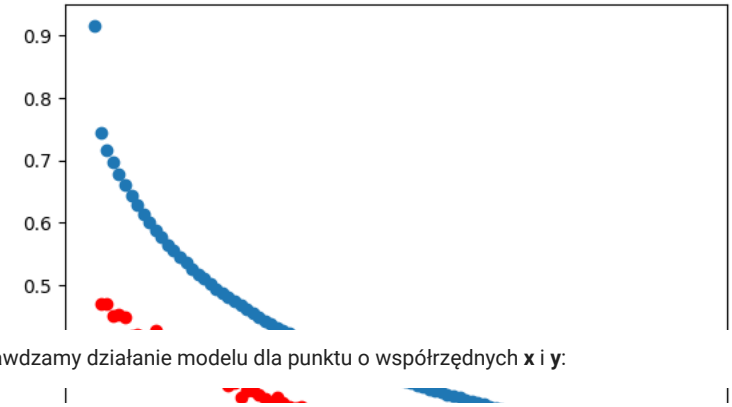
```
      0.37762096524238586,
      0.3737662434577942,
      0.3700096905231476,
      0.3663683831691742,
      0.3626810908317566,
      0.3591282367706299,
      0.3555918037891388,
      0.3523328900337219,
      0.3488442301750183,
      0.3456272780895233,
      0.3424134850502014,
      0.33924564719200134,
      0.3360055983066559,
      0.3331753611564636,
      0.3301744759082794,
      0.32727646827697754,
      0.32438114285469055,
      0.32164862751960754,
      0.31876757740974426,
      0.3161175847053528,
      0.3133723735809326,
      0.31074315309524536,
      0.30825620889663696,
      0.30565738677978516,
      0.30323493480682373,
      0.30070197582244873,
      0.29837313294410706,
      0.29603898525238037,
      0.29369276762008667,
      0.29142653942108154,
      0.28904619812965393,
      0.2868483066558838,
      0.2846512198448181,
      0.28253087401390076,
      0.2804437279701233,
      0.2782551050186157,
      0.2763531804084778,
      0.2742549777030945,
      0.2722958028316498,
      0.27030083537101746,
      0.2683991491794586,
      0.26648035645484924,
      0.26452818512916565,
      0.26274630427360535,
      0.26089951395988464,
      0.2592191994190216,
      0.25740957260131836,
      0.25563284754753113,
      0.2539289891719818,
      0.25222665071487427,
      0.2505522072315216,
      0.2488706409931183,
      0.24728475511074066,
      0.24571771919727325,
      0.24413736164569855,
      0.2425021231174469,
      0.24103006720542908,
      0.23946323990821838]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

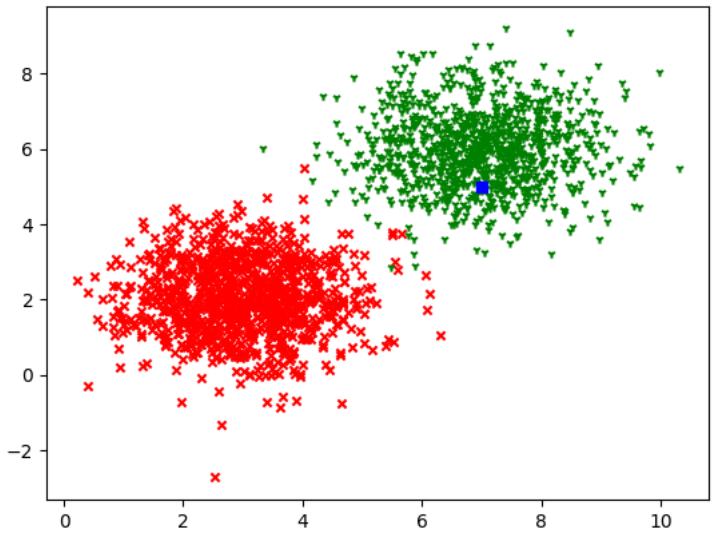
```
     [[ 0.11759357 -0.15217455]
      [ 0.64022464 -0.6263372 ]]
     [-2.972349   3.0955665]
```

```
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 69ms/step
array([[0.8959739 , 0.10402602]], dtype=float32)
```

Learning rate 0.1 optimizer ADAM

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)
```

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 2)                 6         

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

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

```
50/50 [==============================] - 0s 3ms/step - loss: 0.0099 - val_loss: 0.0207
Epoch 98/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0089 - val_loss: 0.0073
Epoch 99/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0078 - val_loss: 0.0287
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0101 - val_loss: 0.0099
```

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

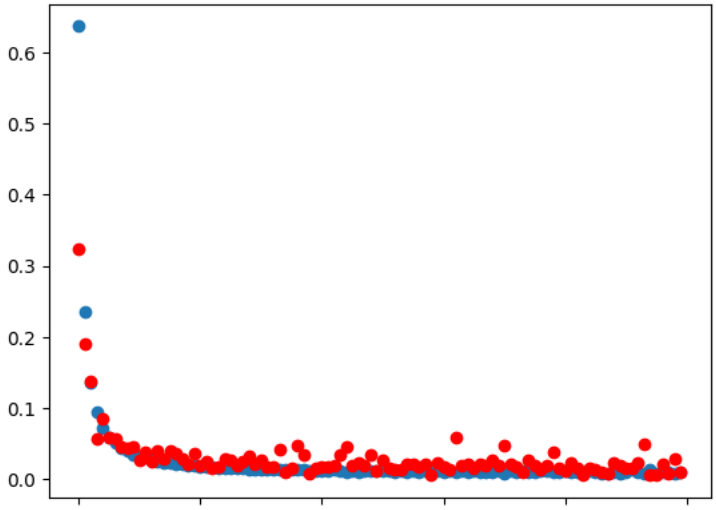
```
 0.012833449989557266,
 0.011409718543291092,
 0.010125366039574146,
 0.010579111985862255,
 0.010140027850866318,
 0.010764293372631073,
 0.011157814413309097,
 0.010554076172411442,
 0.011176113039255142,
 0.010810545645654202,
 0.009633072651922703,
 0.01186126098036766,
 0.010054307989776134,
 0.010739695280790329,
 0.00939125381410122,
 0.01071779616177082,
 0.00954269990324974,
 0.01067799236625433,
 0.009866883978247643,
 0.010534977540373802,
 0.009127454832196236,
 0.010564432479441166,
 0.009667929261922836,
 0.011964447796344757,
 0.00860249251127243,
 0.008929131552577019,
 0.009444888681173325,
 0.01037545781582594,
 0.008197394199669361,
 0.01090339943766594,
 0.009412504732608795,
 0.010516135953366756,
 0.009182869456708431,
 0.008788385428488255,
 0.010870234109461308,
 0.011787639930844307,
 0.008925792761147022,
 0.008838540874421597,
 0.011434349231421947,
 0.008333545178174973,
 0.009139381349086761,
 0.008918415755033493,
 0.010567758232355118,
 0.00863704134702682,
 0.007810349576175213,
 0.007911701686680317,
 0.008502490818500519,
 0.00801928248256445,
 0.009642244316637516,
 0.012517170049250126,
 0.008395851589739323,
 0.008217822760343552,
 0.012617850676178932,
 0.00750131718814373,
 0.009853808209300041,
 0.008860382251441479,
 0.007763553876429796,
 0.010119667276740074]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

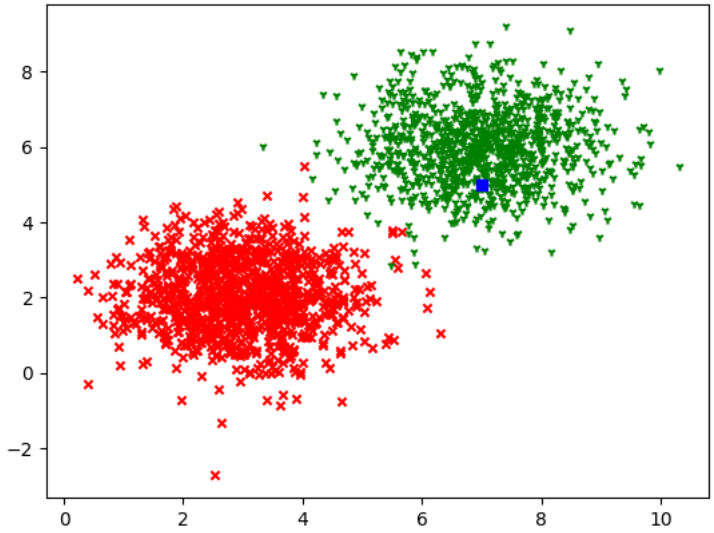
```
[[ 2.8504522 -2.8678107]
 [ 3.102164  -3.1187928]]
[-26.429867  26.581158]
```

```
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 60ms/step
array([[1.000000e+00, 1.348617e-08]], dtype=float32)
```

Learning rate 0.01 Optimizer Adam

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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

```
opt = tf.keras.optimizers.Adam(learning_rate=0.01)
#opt = tf.keras.optimizers.SGD(learning_rate=0.1)
```

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

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

```
Epoch 94/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0208 - val_loss: 0.0288
Epoch 95/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0205 - val_loss: 0.0277
Epoch 96/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0203 - val_loss: 0.0270
Epoch 97/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0199 - val_loss: 0.0314
Epoch 98/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0197 - val_loss: 0.0322
Epoch 99/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0195 - val_loss: 0.0240
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0193 - val_loss: 0.0319
```

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

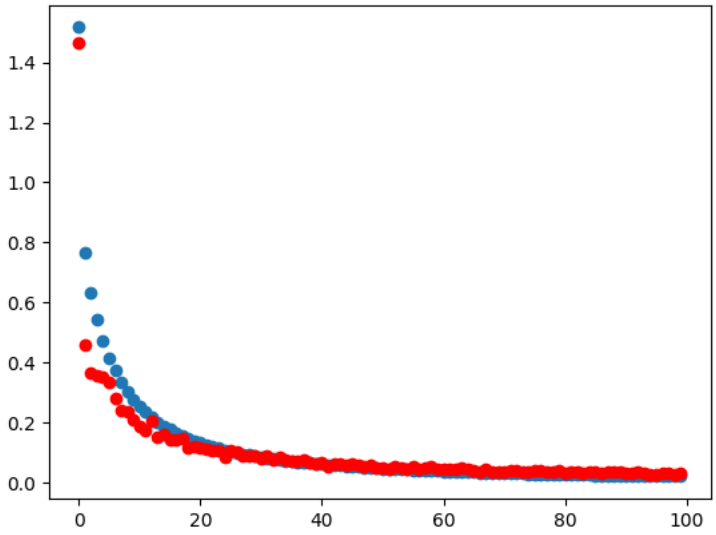
```
[1.516628384590149,
 0.766714334487915,
 0.6332913637161255,
 0.5429794788360596,
 0.4716256260871887,
 0.4158298969268799,
 0.3719572424888611,
 0.33405643701553345,
 0.30325648188591003,
 0.2766752541065216,
 0.2537612020969391,
 0.23413653671741486,
 0.2174025923013687,
 0.2014133334159851,
 0.18813297152519226,
 0.17584475874900818,
 0.16537722945213318,
 0.1558861881494522,
 0.14684484899044037,
 0.1387939304113388,
 0.13161085546016693,
 0.12485776841640472,
 0.11883654445409775,
 0.11316383630037308,
 0.10797787457704544,
 0.10349749773740768,
 0.0990072712302208,
 0.09470067173242569,
 0.09089019894599915,
 0.08736992627382278,
 0.08399143069982529,
 0.08118024468421936,
 0.07799442112445831,
 0.07542849332094193,
 0.07264034450054169,
 0.0701771080493927,
 0.0679037943482399,
 0.06591091305017471,
 0.06394119560718536,
 0.061774902045726776,
 0.0598868764936924,
 0.058120936155319214,
 0.05663297325372696,
 0.054928313940763474,
 0.05332569777965546,
 0.051914747804403305,
 0.05064528062939644,
 0.04916081950068474,
 0.04816620424389839,
 0.046940866857767105,
 0.04570458456873894,
 0.044671062380075455,
 0.04340018332004547,
 0.04243575409054756,
 0.041656218469142914,
 0.0404638797044754,
 0.03968034312129021,
 0.03885228931903839,
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

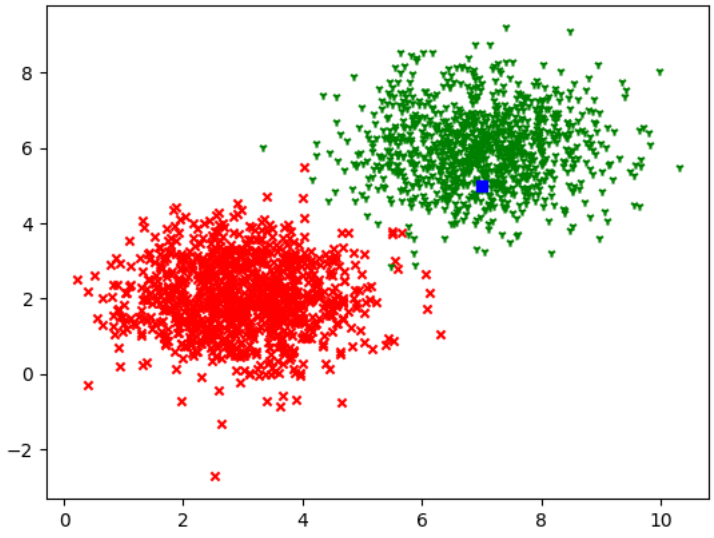
```
[[ 1.2964927 -1.3379517]
 [ 1.569778  -1.6121485]]
[-12.990105  13.388031]
```

```
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]])
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7e971f324550> t
1/1 [==============================] - 0s 81ms/step
array([[9.9965537e-01, 3.4466025e-04]], dtype=float32)
```

Number of epochs - 200

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)


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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

```
epochs = 200
h = model.fit(data_points,labels, verbose=1, epochs=epochs,validation_split=0.2)
```

```
Epoch 1/200
50/50 [==============================] - 1s 5ms/step - loss: 0.6784 - val_loss: 0.3702
Epoch 2/200
50/50 [==============================] - 0s 3ms/step - loss: 0.5356 - val_loss: 0.3391
Epoch 3/200
50/50 [==============================] - 0s 4ms/step - loss: 0.4600 - val_loss: 0.2245
Epoch 4/200
50/50 [==============================] - 0s 3ms/step - loss: 0.4085 - val_loss: 0.2349
Epoch 5/200
50/50 [==============================] - 0s 3ms/step - loss: 0.3694 - val_loss: 0.3204
Epoch 6/200
50/50 [==============================] - 0s 4ms/step - loss: 0.3371 - val_loss: 0.2520
Epoch 7/200
50/50 [==============================] - 0s 3ms/step - loss: 0.3080 - val_loss: 0.3332
Epoch 8/200
50/50 [==============================] - 0s 3ms/step - loss: 0.2851 - val_loss: 0.2028
Epoch 9/200
50/50 [==============================] - 0s 3ms/step - loss: 0.2649 - val_loss: 0.1968
Epoch 10/200
50/50 [==============================] - 0s 3ms/step - loss: 0.2471 - val_loss: 0.1831
Epoch 11/200
50/50 [==============================] - 0s 5ms/step - loss: 0.2322 - val_loss: 0.1931
Epoch 12/200
50/50 [==============================] - 0s 5ms/step - loss: 0.2195 - val_loss: 0.2003
Epoch 13/200
50/50 [==============================] - 0s 5ms/step - loss: 0.2073 - val_loss: 0.1486
Epoch 14/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1964 - val_loss: 0.1929
Epoch 15/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1878 - val_loss: 0.1789
Epoch 16/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1787 - val_loss: 0.1765
Epoch 17/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1712 - val_loss: 0.1573
```

```
Epoch 18/200
50/50 [==============================] - 0s 5ms/step - loss: 0.1645 - val_loss: 0.1279
Epoch 19/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1581 - val_loss: 0.1180
Epoch 20/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1522 - val_loss: 0.1371
Epoch 21/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1468 - val_loss: 0.1074
Epoch 22/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1422 - val_loss: 0.1346
Epoch 23/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1372 - val_loss: 0.1254
Epoch 24/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1330 - val_loss: 0.1249
Epoch 25/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1295 - val_loss: 0.1218
Epoch 26/200
50/50 [==============================] - 0s 3ms/step - loss: 0.1255 - val_loss: 0.1097
Epoch 27/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1219 - val_loss: 0.1224
Epoch 28/200
50/50 [==============================] - 0s 4ms/step - loss: 0.1186 - val_loss: 0.1009
Epoch 29/200
50/50 [==============================] - 0s 5ms/step - loss: 0.1158 - val_loss: 0.1086
```

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

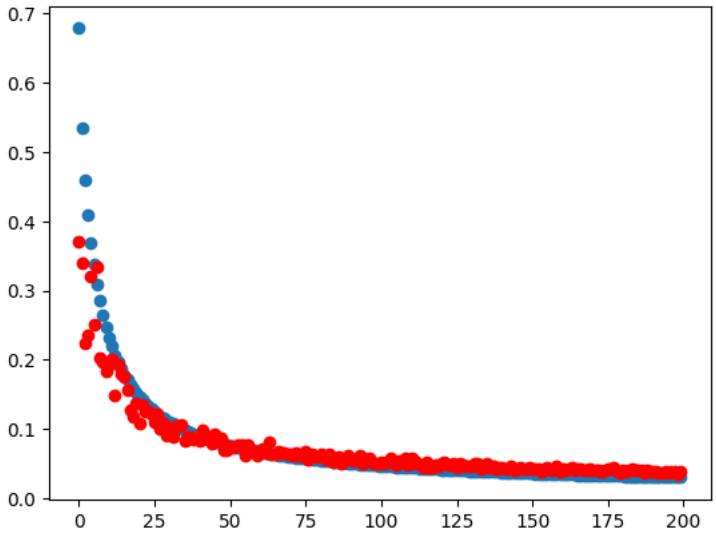
```
 0.037354834377765656,
 0.037177640944719315,
 0.03697577863931656,
 0.036893654614686966,
 0.036715954542160034,
 0.03650255128741264,
 0.036366499960422516,
 0.03621961548924446,
 0.03611750155687332,
 0.03588716685771942,
 0.03577771410346031,
 0.0355716273188591,
 0.03548375517129898,
 0.03524114191532135,
 0.035177454352378845,
 0.035026222467422485,
 0.034832119941711426,
 0.03468320891261101,
 0.03456680476665497,
 0.034421540796756744,
 0.03436252102255821,
 0.03410668298602104,
 0.034098993986845016,
 0.033914756029844284,
 0.03382403776049614,
 0.033630408346652985,
 0.03353374823927879,
 0.03339836001396179,
 0.03329092636704445,
 0.03322652727365494,
 0.03302273899316788,
 0.03289615735411644,
 0.03277533873915672,
 0.03269818425178528,
 0.03255663067102432,
 0.03240637481212616,
 0.03236261010169983,
 0.03218785300850868,
 0.032113298773765564,
 0.031981661915779114,
 0.03188832104206085,
 0.03176287189126015,
 0.03167341649532318,
 0.03159536421298981,
 0.031451430171728134,
 0.03139336779713631,
 0.031205225735902786,
 0.03120521456003189,
 0.031056106090545654,
 0.03094319999217987,
 0.030847031623125076,
 0.030719151720404625,
 0.03069242462515831,
 0.030524196103215218,
 0.0304960198700428,
 0.030372602865099907,
 0.030201435089111328,
 0.030209746211767197]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

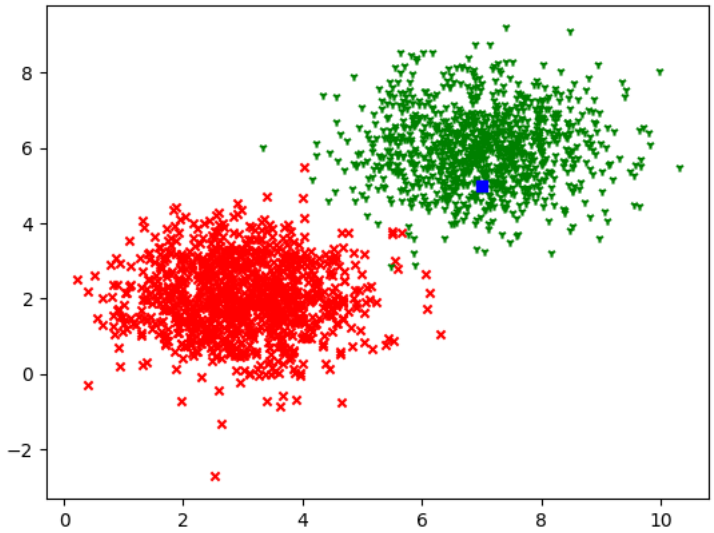
```
[[ 1.0474969 -1.0475117]
 [ 1.3355495 -1.3355637]]
[-10.632143  10.632277]
```

```
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]])
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7e971f587640> t
1/1 [==============================] - 0s 90ms/step
array([[0.99883777, 0.00116228]], dtype=float32)
```

Number of epochs - 10

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

```
epochs = 10
h = model.fit(data_points,labels, verbose=1, epochs=epochs,validation_split=0.2)
```

```
Epoch 1/10
50/50 [==============================] - 1s 8ms/step - loss: 0.6953 - val_loss: 0.5287
Epoch 2/10
50/50 [==============================] - 0s 3ms/step - loss: 0.5342 - val_loss: 0.3537
Epoch 3/10
50/50 [==============================] - 0s 3ms/step - loss: 0.4671 - val_loss: 0.4315
Epoch 4/10
50/50 [==============================] - 0s 2ms/step - loss: 0.4141 - val_loss: 0.3117
Epoch 5/10
50/50 [==============================] - 0s 3ms/step - loss: 0.3718 - val_loss: 0.3386
Epoch 6/10
50/50 [==============================] - 0s 3ms/step - loss: 0.3386 - val_loss: 0.2208
Epoch 7/10
50/50 [==============================] - 0s 3ms/step - loss: 0.3095 - val_loss: 0.2287
Epoch 8/10
50/50 [==============================] - 0s 3ms/step - loss: 0.2853 - val_loss: 0.2358
Epoch 9/10
50/50 [==============================] - 0s 3ms/step - loss: 0.2657 - val_loss: 0.2067
Epoch 10/10
50/50 [==============================] - 0s 2ms/step - loss: 0.2487 - val_loss: 0.1945
```

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

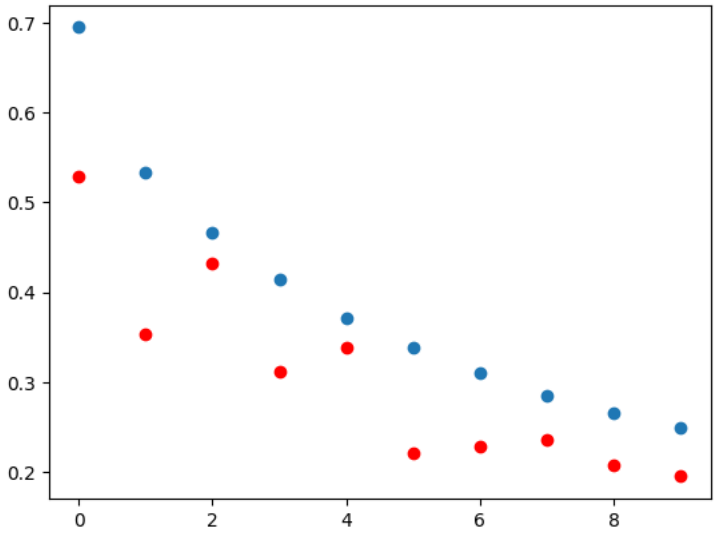
```
[0.6952502727508545,
 0.5341777205467224,
 0.46714943647384644,
 0.4141151010990143,
 0.37180405855178833,
 0.3386291563510895,
 0.30952635407447815,
 0.28533661365509033,
 0.2656831443309784,
 0.2486899048089981]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

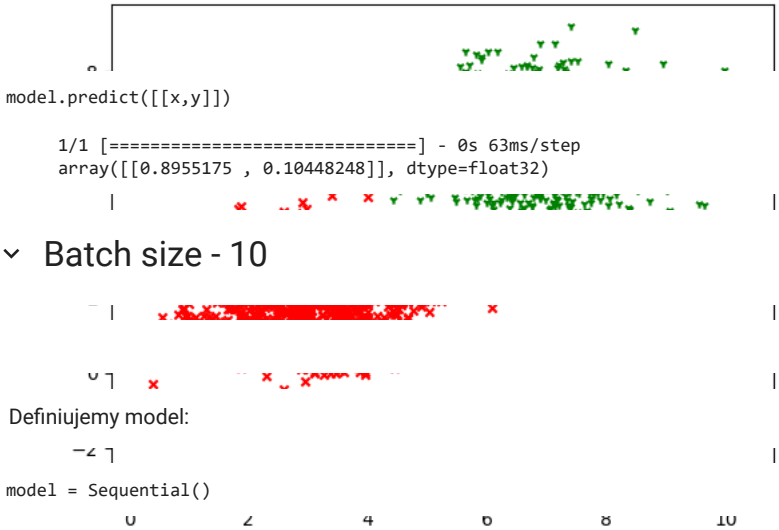
```
[[ 0.14794256 -0.11549832]
 [ 0.63155496 -0.63953596]]
[-3.0937936  2.9573646]
```

```
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 63ms/step
array([[0.8955175 , 0.10448248]], dtype=float32)
```

Batch size - 10

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

```
epochs = 100
h = model.fit(data_points,labels, verbose=1, epochs=epochs,validation_split=0.2, batch_size=10)
```

```
160/160 [==============================] - 0s 3ms/step - loss: 0.0272 - val_loss: 0.0403
Epoch 76/100
160/160 [==============================] - 1s 5ms/step - loss: 0.0269 - val_loss: 0.0332
Epoch 77/100
160/160 [==============================] - 1s 5ms/step - loss: 0.0267 - val_loss: 0.0338
Epoch 78/100
160/160 [==============================] - 1s 6ms/step - loss: 0.0266 - val_loss: 0.0396
Epoch 79/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0264 - val_loss: 0.0311
Epoch 80/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0262 - val_loss: 0.0352
Epoch 81/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0260 - val_loss: 0.0356
Epoch 82/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0258 - val_loss: 0.0318
Epoch 83/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0256 - val_loss: 0.0351
Epoch 84/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0254 - val_loss: 0.0311
Epoch 85/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0252 - val_loss: 0.0346
Epoch 86/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0249 - val_loss: 0.0391
Epoch 87/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0249 - val_loss: 0.0356
Epoch 88/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0247 - val_loss: 0.0313
Epoch 89/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0245 - val_loss: 0.0343
Epoch 90/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0245 - val_loss: 0.0315
Epoch 91/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0244 - val_loss: 0.0319
Epoch 92/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0241 - val_loss: 0.0346
Epoch 93/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0238 - val_loss: 0.0252
Epoch 94/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0236 - val_loss: 0.0409
Epoch 95/100
160/160 [==============================] - 1s 4ms/step - loss: 0.0234 - val_loss: 0.0238
Epoch 96/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0237 - val_loss: 0.0290
Epoch 97/100
160/160 [==============================] - 1s 4ms/step - loss: 0.0235 - val_loss: 0.0300
Epoch 98/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0233 - val_loss: 0.0289
Epoch 99/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0231 - val_loss: 0.0365
Epoch 100/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0231 - val_loss: 0.0274
```

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

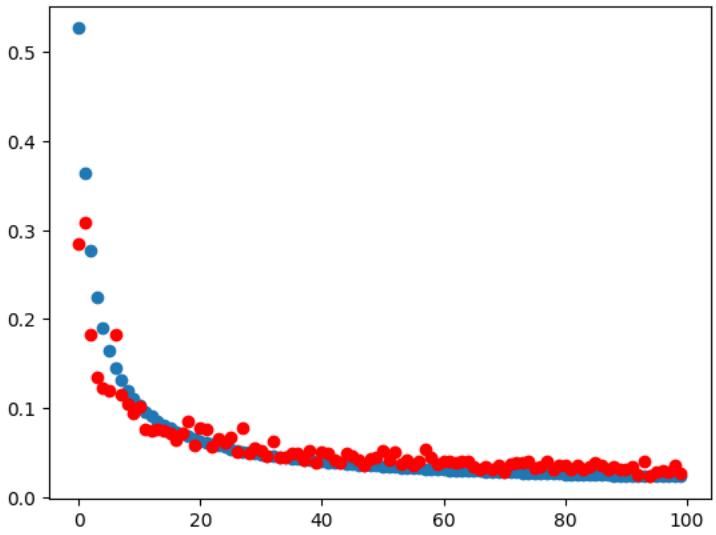
```
0.02637522481381893,
0.02623055875301361,
0.02596934884786606,
0.025808637961745262,
0.02563364990055561,
0.025439996272325516,
0.025239482522010803,
0.024875523522496223,
0.024888617917895317,
0.02470150962471962,
0.02448723465204239,
0.024471597746014595,
0.02435126341879368,
0.024105750024318695,
0.023777354508638382,
0.023644445464015007,
0.02339470200240612,
0.023671910166740417,
0.02346346341073513,
0.023256121203303337,
0.02306966856122017,
0.023059803992509842]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

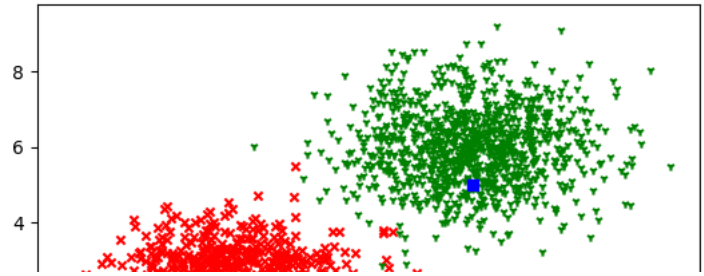
```
[[ 1.2287223 -1.2285511]
 [ 1.5051947 -1.5050318]]
[-12.097679  12.096138]
```

```
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 60ms/step
array([[9.9968362e-01, 3.1638655e-04]], dtype=float32)
```


Batch size - 20

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)
```

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

```
epochs = 100
h = model.fit(data_points,labels, verbose=1, epochs=epochs,validation_split=0.2,batch_size=20)
```

```
Epoch 72/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0433 - val_loss: 0.0518
Epoch 73/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0429 - val_loss: 0.0526
Epoch 74/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0424 - val_loss: 0.0474
Epoch 75/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0420 - val_loss: 0.0405
Epoch 76/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0418 - val_loss: 0.0472
Epoch 77/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0414 - val_loss: 0.0542
Epoch 78/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0410 - val_loss: 0.0476
Epoch 79/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0406 - val_loss: 0.0540
Epoch 80/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0403 - val_loss: 0.0523
Epoch 81/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0399 - val_loss: 0.0529
Epoch 82/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0396 - val_loss: 0.0446
Epoch 83/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0393 - val_loss: 0.0442
Epoch 84/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0389 - val_loss: 0.0500
Epoch 85/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0387 - val_loss: 0.0444
Epoch 86/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0384 - val_loss: 0.0409
Epoch 87/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0381 - val_loss: 0.0503
Epoch 88/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0379 - val_loss: 0.0485
Epoch 89/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0376 - val_loss: 0.0502
Epoch 90/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0372 - val_loss: 0.0461
Epoch 91/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0370 - val_loss: 0.0473
Epoch 92/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0369 - val_loss: 0.0427
Epoch 93/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0364 - val_loss: 0.0390
Epoch 94/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0363 - val_loss: 0.0436
Epoch 95/100
80/80 [==============================] - 0s 4ms/step - loss: 0.0359 - val_loss: 0.0447
Epoch 96/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0357 - val_loss: 0.0401
Epoch 97/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0355 - val_loss: 0.0446
Epoch 98/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0353 - val_loss: 0.0453
Epoch 99/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0350 - val_loss: 0.0387
Epoch 100/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0348 - val_loss: 0.0430
```

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

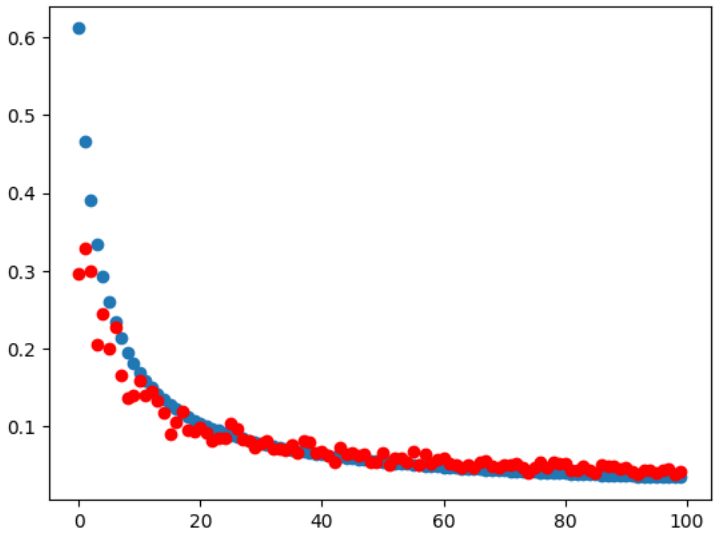
```
0.045363337282419205,
0.04494510963559151,
0.044541530311107635,
0.044119250029325485,
0.043680671602487564,
0.04325522854924202,
0.042873360216617584,
0.04241203889250755,
0.04196306690573692,
0.041769761592149734,
0.0413811169564724,
0.041022129356861115,
0.04062827676534653,
0.04027000814676285,
0.03993172571063042,
0.039560601115226746,
0.03929023817181587,
0.03888659551739693,
0.03872072696685791,
0.03844732418656349,
0.038096629083156586,
0.03788120299577713,
0.03755063936114311,
0.03721831366419792,
0.03703875467181206,
0.036895815283060074,
0.03643624484539032,
0.036308396607637405,
0.03594972565770149,
0.03571886941790581,
0.035505060106515884,
0.035271018743515015,
0.03501598909497261,
0.03481154516339302]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

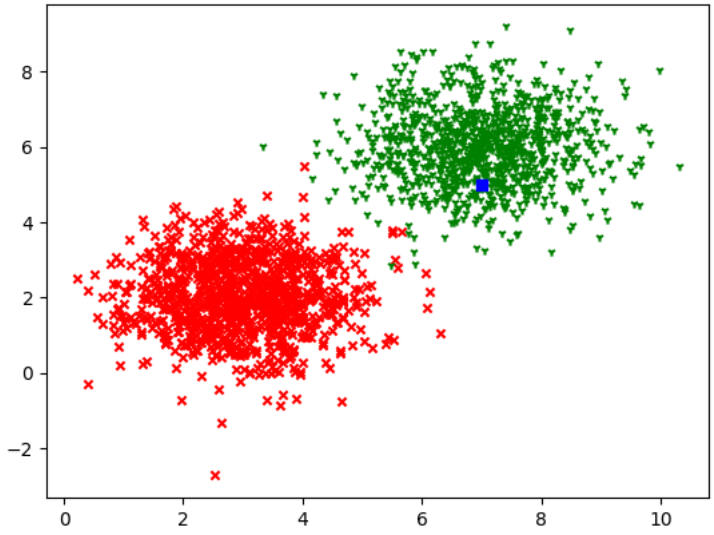
```
[[ 0.9734907  -0.97262555]
 [ 1.2679751  -1.2671809 ]]
[-9.973655  9.966012]
```

```
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 64ms/step
array([[0.9982717 , 0.00172823]], dtype=float32)
```

Batch size 50

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 = 2, use_bias=True, input_dim=2, activation = "softmax"))
```

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=0.1)
```

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

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Przygotowanie danych:

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

```
array([[ 0.40152196, -0.27913799],
       [ 2.93831545,  3.15416321],
       [ 3.41174677,  0.25690969],
       ...,
       [ 4.44440077,  5.8797627 ],
       [ 8.0019706 ,  6.79916435],
       [ 7.88941671,  7.03935654]])
```

Proces **uczenia**:

```
epochs = 100
h = model.fit(data_points,labels, verbose=1, epochs=epochs,validation_split=0.2, batch_size=50)
```

```
Epoch 72/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0818 - val_loss: 0.0880
Epoch 73/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0810 - val_loss: 0.0847
Epoch 74/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0801 - val_loss: 0.0809
Epoch 75/100
32/32 [==============================] - 0s 5ms/step - loss: 0.0794 - val_loss: 0.0834
Epoch 76/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0786 - val_loss: 0.0760
Epoch 77/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0778 - val_loss: 0.0851
Epoch 78/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0774 - val_loss: 0.0782
Epoch 79/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0764 - val_loss: 0.0737
Epoch 80/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0758 - val_loss: 0.0711
Epoch 81/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0751 - val_loss: 0.0837
Epoch 82/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0744 - val_loss: 0.0752
Epoch 83/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0738 - val_loss: 0.0754
Epoch 84/100
32/32 [==============================] - 0s 5ms/step - loss: 0.0731 - val_loss: 0.0801
Epoch 85/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0726 - val_loss: 0.0737
Epoch 86/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0719 - val_loss: 0.0727
Epoch 87/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0713 - val_loss: 0.0719
Epoch 88/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0706 - val_loss: 0.0662
Epoch 89/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0701 - val_loss: 0.0713
Epoch 90/100
32/32 [==============================] - 0s 5ms/step - loss: 0.0696 - val_loss: 0.0713
Epoch 91/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0690 - val_loss: 0.0745
Epoch 92/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0685 - val_loss: 0.0760
Epoch 93/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0680 - val_loss: 0.0742
Epoch 94/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0675 - val_loss: 0.0677
Epoch 95/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0670 - val_loss: 0.0691
Epoch 96/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0665 - val_loss: 0.0660
Epoch 97/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0660 - val_loss: 0.0648
Epoch 98/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0655 - val_loss: 0.0688
Epoch 99/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0650 - val_loss: 0.0643
Epoch 100/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0646 - val_loss: 0.0659
```

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

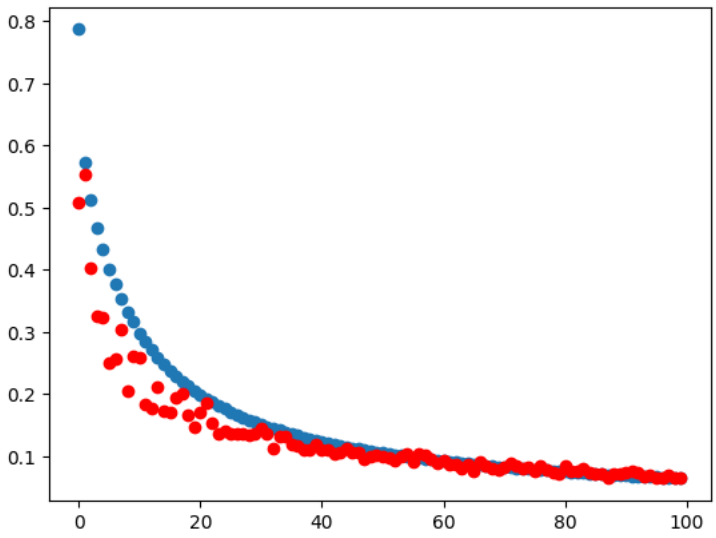
```
0.09948961700916291,
0.09827997535467148,
0.09675713628530502,
0.09576766192913055,
0.09448768943548203,
0.09339023381471634,
0.09226212650537491,
0.09129270911216736,
0.09017112106084824,
0.08907113969326019,
0.08809083700180054,
0.08702468127012253,
0.08615610748529434,
0.08521895110607147,
0.08439875394105911,
0.0833747610449791,
0.08253853768110275,
0.08183243125677109,
0.08095536381006241,
0.08012907207012177,
0.07939121127128601,
0.07859445363283157,
0.0778392031788826,
0.07738108932971954,
0.0763925090432167,
0.07581768184900284,
0.07505294680595398,
0.07443199306726456,
0.0737914890050888,
0.07311644405126572,
0.07257156819105148,
0.07194644957780838,
0.07131907343864441,
0.07062239944934845,
0.07006264477968216,
0.06958479434251785,
0.06904171407222748,
0.06845030188560486,
0.06798678636550903,
0.06748371571302414,
0.06701324135065079,
0.06646713614463806,
0.06601417809724808,
0.06547383964061737,
0.06496863067150116,
0.06461143493652344]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

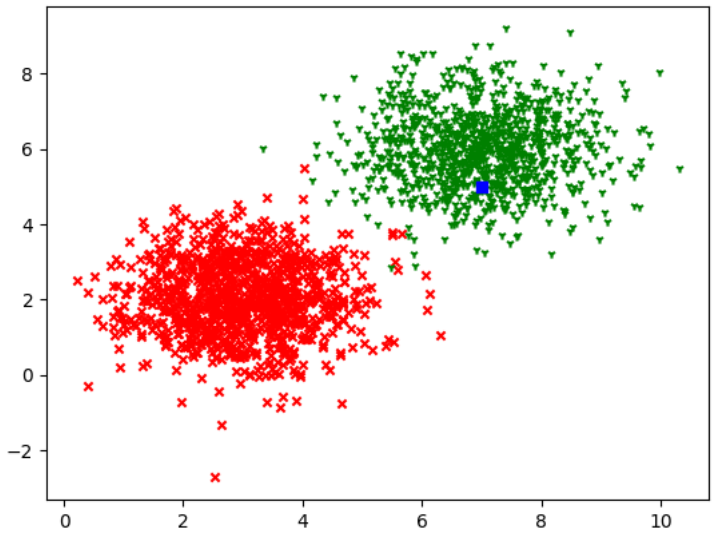
```
[[ 0.68553364 -0.6842611 ]
 [ 1.0107037  -1.0096275 ]]
[-7.406832   7.3957624]
```

```
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 67ms/step
array([[0.9925171, 0.0074829]], dtype=float32)
```

Najlepsze wyniki otrzymałem dla współczynnika uczenia 0.1, liczby epok 3000, batcha równego 20, najgorsze dla współczynnika uczenia 0.001, liczby epok 10, batcha równego 50