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:

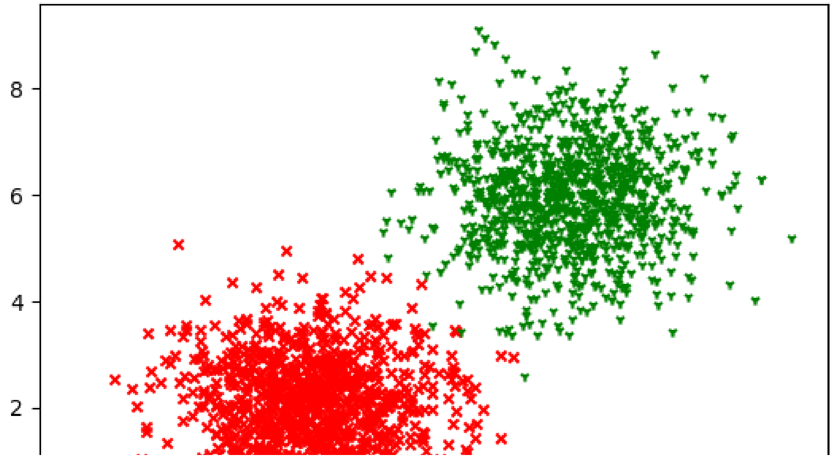
```
[0]*10+[1]*10
```

```
[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

```
       4.67013015,  2.70147121,  3.46922723,  3.52362214,  2.60168478,
       4.01621889,  4.12100604,  2.72752771,  2.75719359,  4.02926479,
       3.28814498,  2.54441411,  3.49969971,  1.84603106,  3.01861976,
       3.33037865,  4.63877474,  3.22667965,  3.42150617,  2.99610977,
       3.92730886,  2.93353636,  3.54677324,  2.18964223,  2.14112941,
       2.3725276 ,  3.63421524,  4.3018087 ,  1.12315323,  0.79435992,
       3.72061328,  3.65506081,  3.78239993,  3.7291989 ,  2.59910655,
       4.04697558,  4.04816327,  2.69502771,  4.34208597,  3.55934781,
       4.28531443,  1.87753953,  2.79867148,  2.72406779,  2.58175449,
       2.71320505,  4.0847115 ,  3.02323071,  2.31744951,  2.60453008,
       2.90800369,  2.14776034,  4.21060945,  1.77524038,  2.6312135 ,
       3.6106395 ,  5.23375638,  4.82731622,  3.27098369,  2.94126894,
       3.38526073,  4.3067985 ,  2.86913682,  3.22277753,  3.12253827,
       3.60975487,  2.27899681,  3.52791693,  1.61311302,  2.32490014,
       2.18145502,  2.30392162,  1.77847212,  3.19189672,  2.58963709,
       3.99746712,  3.6677791 ,  2.53024815,  3.54696023,  4.18354706,
       1.29657966,  2.45625325,  3.30326258,  4.56749604,  3.1815515 ,
       3.74255693,  3.16999129,  2.73032971,  2.9479818 ,  3.05047653,
       3.025813  ,  3.94078173,  2.46439622,  3.37261736,  4.46956822,
       1.7741385 ,  3.22896292,  1.84104922,  2.45036858,  3.61103535,
       4.03725328,  1.23050797,  3.48538395,  4.73910316,  3.4573818 ,
       2.30814498,  5.21912853,  3.56975434,  3.93582052,  4.0883278 ,
       1.76238825,  2.3287282 ,  2.53005539,  5.4803398 ,  5.41313985,
       5.21649602,  1.70691479,  3.07901139,  5.59423695,  3.78434683,
       3.41518979,  3.01700775,  1.12320482,  2.4850929 ,  4.31186964,
       2.44639538,  3.44416348,  2.74779462,  3.31551388,  3.15156282])
```

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 = "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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
Epoch 78/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0359 - val_loss: 0.0348
Epoch 79/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0355 - val_loss: 0.0331
Epoch 80/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0354 - val_loss: 0.0348
Epoch 81/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0351 - val_loss: 0.0336
Epoch 82/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0348 - val_loss: 0.0383
Epoch 83/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0346 - val_loss: 0.0350
Epoch 84/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0344 - val_loss: 0.0346
Epoch 85/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0341 - val_loss: 0.0347
Epoch 86/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0338 - val_loss: 0.0327
Epoch 87/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0336 - val_loss: 0.0322
Epoch 88/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0334 - val_loss: 0.0363
Epoch 89/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0332 - val_loss: 0.0371
Epoch 90/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0328 - val_loss: 0.0392
Epoch 91/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0328 - val_loss: 0.0364
Epoch 92/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0325 - val_loss: 0.0366
Epoch 93/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0323 - val_loss: 0.0320
Epoch 94/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0318 - val_loss: 0.0286
Epoch 95/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0317 - val_loss: 0.0364
Epoch 96/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0316 - val_loss: 0.0353
Epoch 97/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0315 - val_loss: 0.0353
Epoch 98/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0313 - val_loss: 0.0290
Epoch 99/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0308 - val_loss: 0.0422
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0309 - val_loss: 0.0349
```

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

```
        0.05150304734706879,
        0.05073777213692665,
        0.05022202432155609,
        0.04928869754076004,
        0.048732005059719086,
        0.04790578782558441,
        0.04718722403049469,
        0.046520642936229706,
        0.04612436890602112,
        0.04560814052820206,
```

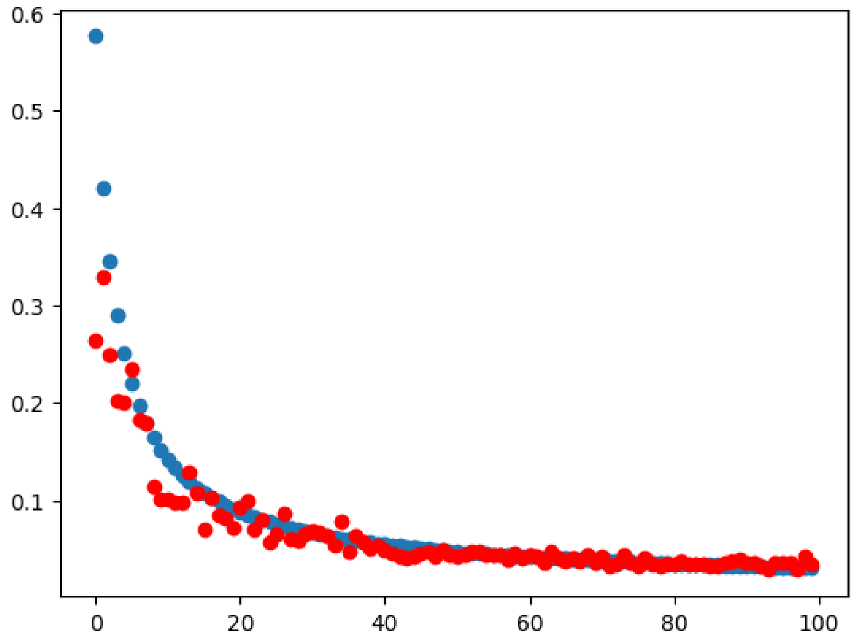
```
 0.03148457035422325,
 0.03125711530447006,
 0.030768103897571564,
 0.03090226836502552]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

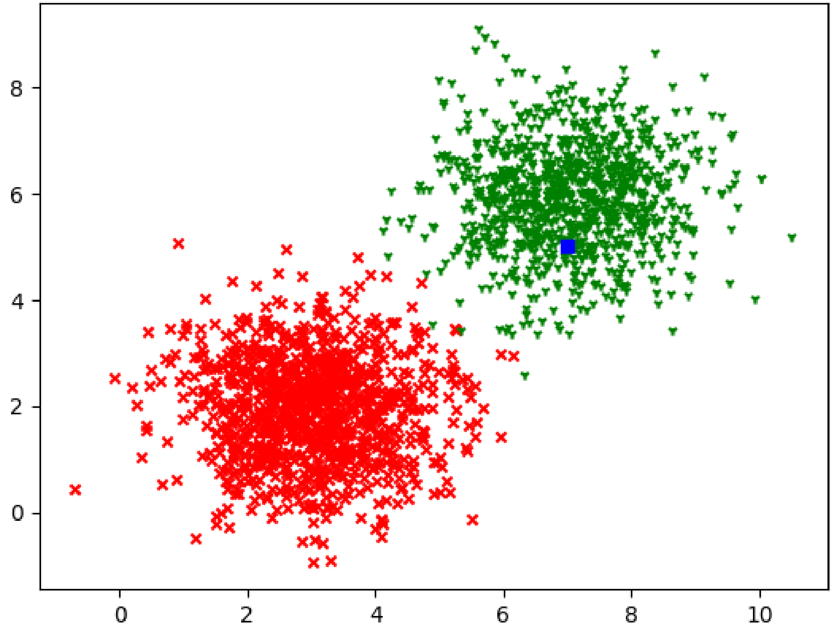
```
[[0.976188 ]
 [1.4137076]]
[-10.545252]
```

```
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.]], 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 = 1, 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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
50/50 [==============================] - 0s 2ms/step - loss: 0.1557 - val_loss: 0.1326
Epoch 95/100
50/50 [==============================] - 0s 2ms/step - loss: 0.1545 - val_loss: 0.1389
Epoch 96/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1535 - val_loss: 0.1307
Epoch 97/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1522 - val_loss: 0.1314
Epoch 98/100
50/50 [==============================] - 0s 2ms/step - loss: 0.1511 - val_loss: 0.1250
Epoch 99/100
50/50 [==============================] - 0s 2ms/step - loss: 0.1501 - val_loss: 0.1223
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1490 - val_loss: 0.1268
```

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

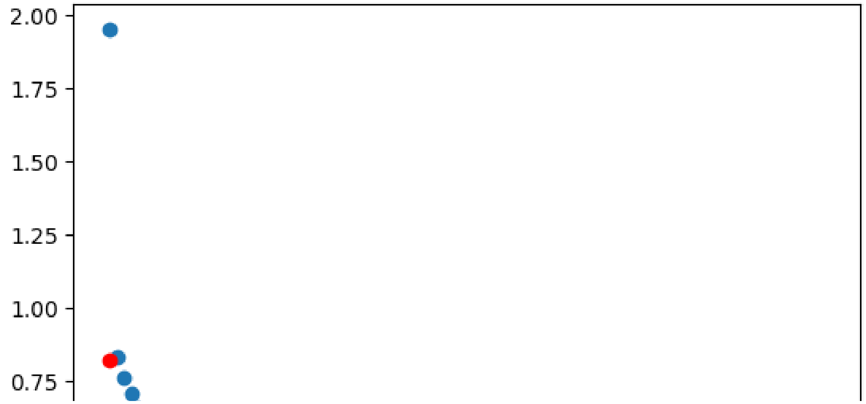
```
0.1791458725929260З,
0.17762134969234467,
0.1760701686143875,
0.1745370328426361,
0.17292411625385284,
0.17147022485733032,
0.17005662620067596,
0.16858820617198944,
0.16722595691680908,
0.165882408618927,
0.16447201371192932,
0.16315360367298126,
0.16186299920082092,
0.16056708991527557,
0.1593204140663147,
0.1581363081932068,
0.15687452256679535,
0.15574510395526886,
0.15445458889007568,
0.15347152948379517,
0.15224675834178925,
0.151114821434021,
0.15006545186042786,
0.14902494847774506]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

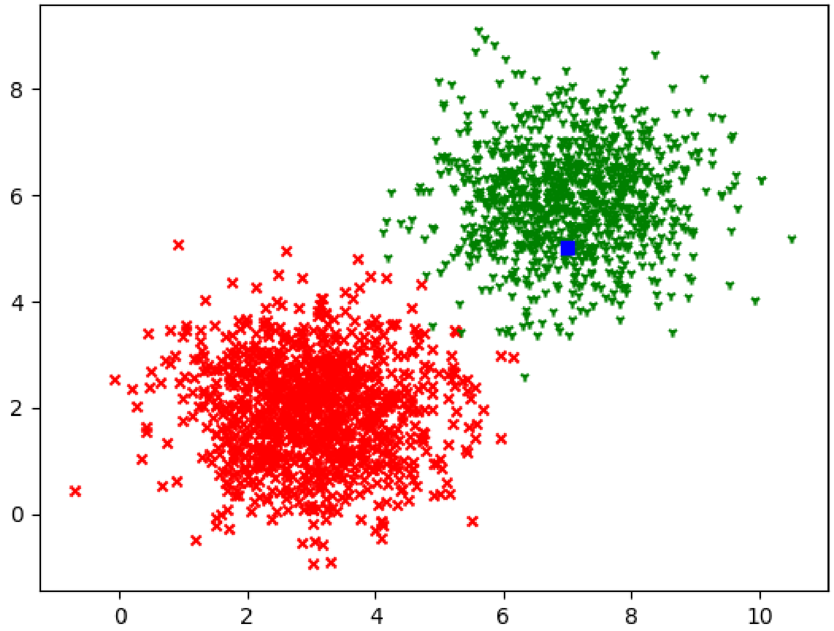
```
[[0.27534017]
 [0.8080815 ]]
[-4.4306183]
```

```
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 74ms/step
array([[1.]], 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 = 1, 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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
50/50 [==============================] - 0s 2ms/step - loss: 0.0070 - val_loss: 0.0054
Epoch 86/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0071 - val_loss: 0.0083
Epoch 87/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0071 - val_loss: 0.0232
Epoch 88/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0072 - val_loss: 0.0101
Epoch 89/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0111 - val_loss: 0.0289
Epoch 90/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0096 - val_loss: 0.0129
Epoch 91/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0067 - val_loss: 0.0089
Epoch 92/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0092 - val_loss: 0.0329
Epoch 93/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0067 - val_loss: 0.0145
Epoch 94/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0073 - val_loss: 0.0139
Epoch 95/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0068 - val_loss: 0.0186
Epoch 96/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0076 - val_loss: 0.0175
Epoch 97/100
50/50 [==============================] - 0s 5ms/step - loss: 0.0067 - val_loss: 0.0123
Epoch 98/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0077 - val_loss: 0.0172
Epoch 99/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0070 - val_loss: 0.0070
Epoch 100/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0080 - val_loss: 0.0278
```

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

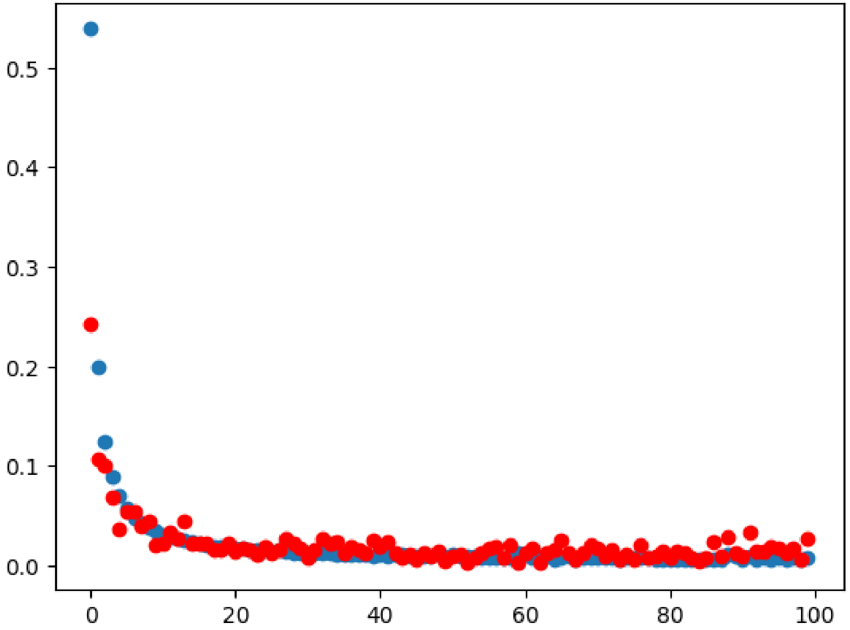
```
        0.012868493795394897,
        0.011629335582256317,
        0.007521096616983414,
        0.009643647819757462,
        0.010077561251819134,
        0.00716122193261981,
        0.00870305672287941,
        0.009027077816426754,
        0.007176931947469711,
        0.008700431324541569,
        0.007629499305039644,
        0.007525546010583639,
        0.007324058562517166,
        0.007840651087462902,
        0.006922069936990738,
        0.008075136691331863,
        0.0075485240668058395,
        0.008625146001577377,
        0.008245621807873249,
        0.006644793786108494,
        0.007209888193756342,
        0.0069216229021549225,
        0.006888733711093664,
        0.00666001858189702,
        0.006527409423142672,
        0.0070349290035665035,
        0.007070205174386501,
        0.007125018630176783,
        0.00721875112503767,
        0.011109217070043087,
        0.009618628770112991,
        0.006656251382082701,
        0.00920072291046381,
        0.00670555280521512,
        0.007280466612428427,
        0.006781424395740032,
        0.007577568292617798,
        0.006670651491731405,
        0.007742272689938545,
        0.0070458813570439816,
        0.007955965586006641]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

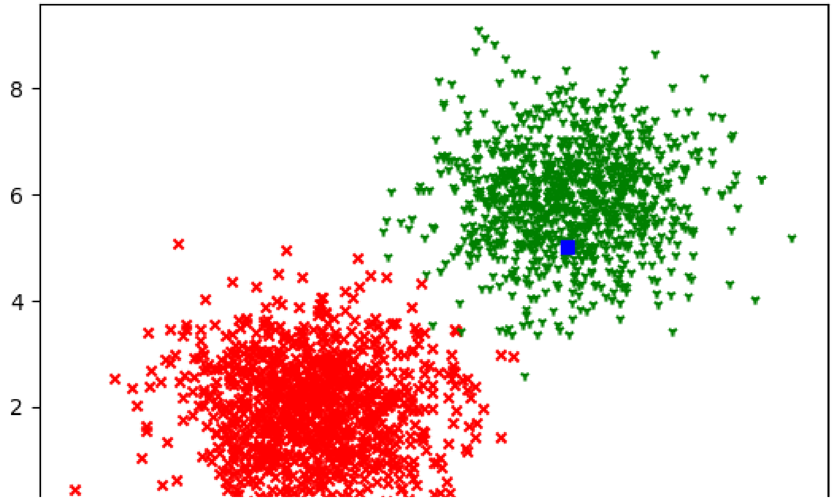
```
[[2.8030927]
 [3.1745062]]
[-28.42742]
```

```
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 88ms/step
array([[1.]], 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 = 1, 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_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
50/50 [==============================] - 0s 2ms/step - loss: 0.0291 - val_loss: 0.0338
Epoch 76/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0287 - val_loss: 0.0318
Epoch 77/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0281 - val_loss: 0.0305
Epoch 78/100
50/50 [==============================] - 0s 5ms/step - loss: 0.0278 - val_loss: 0.0266
Epoch 79/100
50/50 [==============================] - 0s 5ms/step - loss: 0.0272 - val_loss: 0.0306
Epoch 80/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0267 - val_loss: 0.0280
Epoch 81/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0266 - val_loss: 0.0285
Epoch 82/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0259 - val_loss: 0.0307
Epoch 83/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0255 - val_loss: 0.0256
Epoch 84/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0251 - val_loss: 0.0266
Epoch 85/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0247 - val_loss: 0.0271
Epoch 86/100
50/50 [==============================] - 0s 5ms/step - loss: 0.0243 - val_loss: 0.0253
Epoch 87/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0240 - val_loss: 0.0244
Epoch 88/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0236 - val_loss: 0.0228
Epoch 89/100
50/50 [==============================] - 0s 5ms/step - loss: 0.0235 - val_loss: 0.0231
Epoch 90/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0229 - val_loss: 0.0243
Epoch 91/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0227 - val_loss: 0.0230
Epoch 92/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0223 - val_loss: 0.0223
Epoch 93/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0221 - val_loss: 0.0222
Epoch 94/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0217 - val_loss: 0.0257
Epoch 95/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0214 - val_loss: 0.0236
Epoch 96/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0211 - val_loss: 0.0250
Epoch 97/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0208 - val_loss: 0.0237
Epoch 98/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0205 - val_loss: 0.0205
Epoch 99/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0203 - val_loss: 0.0225
Epoch 100/100
50/50 [==============================] - 0s 2ms/step - loss: 0.0200 - val_loss: 0.0212
```

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

```
     0.058860503137111664,
     0.0572054386138916,
     0.05562101677060127,
     0.05422927439212799,
     0.05283689498901367,
     0.051276255398988724,
```

```
        0.022680750116705894,
        0.022267919033765793,
        0.0221176128834486,
        0.021740354597568512,
        0.021388988941907883,
        0.02106151357293129,
        0.020833753049373627,
        0.02054629847407341,
        0.020297998562455177,
        0.019981322810053825]
```

Sprawdźmy jakie są **wartości wag**:

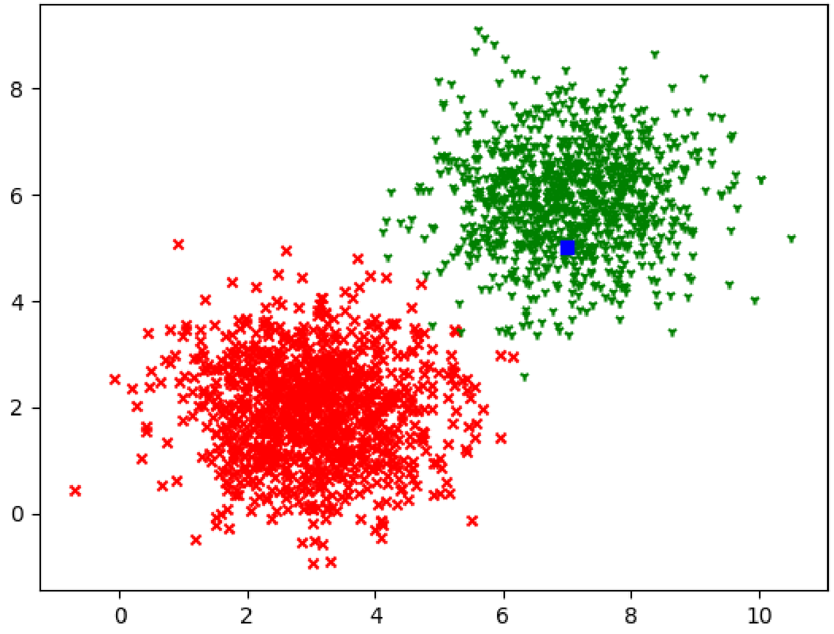
```
weights = model.get_weights()

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

```
[[1.2608098]
 [1.6906005]]
[-12.958403]
```

```
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 0x7d1f18db2050> triggered tf.funct
1/1 [==============================] - 0s 61ms/step
array([[1.]], 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 = 1, 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_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_5 (Dense)             (None, 1)                 3         
                                                                 
=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
```

```
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
Epoch 191/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0213 - val_loss: 0.0217
Epoch 192/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0212 - val_loss: 0.0240
Epoch 193/200
50/50 [==============================] - 0s 3ms/step - loss: 0.0211 - val_loss: 0.0214
Epoch 194/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0211 - val_loss: 0.0253
Epoch 195/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0210 - val_loss: 0.0233
Epoch 196/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0210 - val_loss: 0.0229
Epoch 197/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0208 - val_loss: 0.0239
Epoch 198/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0208 - val_loss: 0.0234
Epoch 199/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0208 - val_loss: 0.0223
Epoch 200/200
50/50 [==============================] - 0s 2ms/step - loss: 0.0208 - val_loss: 0.0220
```

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

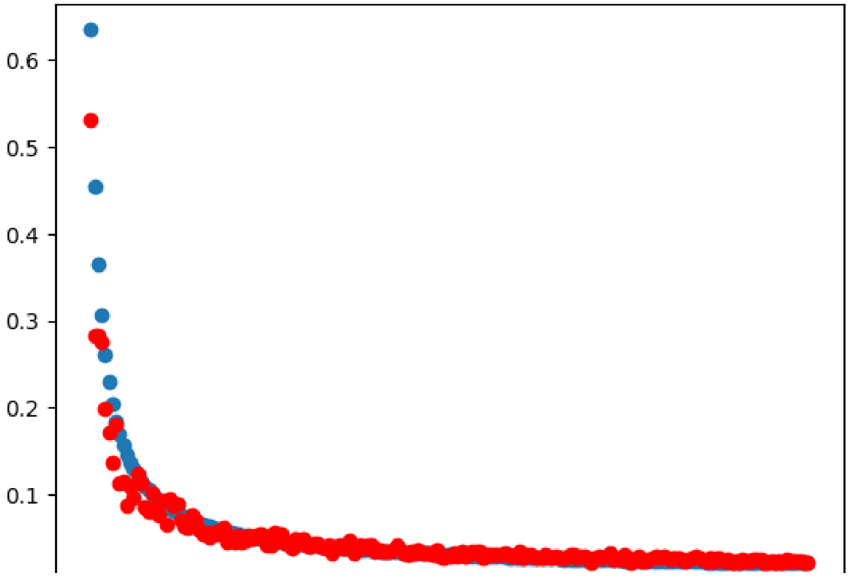
```
    0.02259901538491249,
    0.022498061880469322,
    0.022502753883600235,
    0.02228054776787758,
    0.022296320647001266,
    0.02225557528436184,
    0.0222723837941885,
    0.022042419761419296,
    0.02201276831328869,
    0.02203170210123062,
    0.02192268893122673,
    0.02181338332593441,
    0.02175765298306942,
    0.02170962281525135,
    0.02156413532793522,
    0.02156808227300644,
    0.02149846777319908,
    0.021432815119624138,
    0.021359015256166458,
    0.02141267992556095,
    0.02129703015089035,
    0.021153021603822708,
    0.021115155890583992,
    0.02106969989836216,
    0.02104480005800724,
    0.02095261961221695,
    0.020845530554652214,
    0.02084415592253208,
    0.020754141733050346,
    0.020753316581249237]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

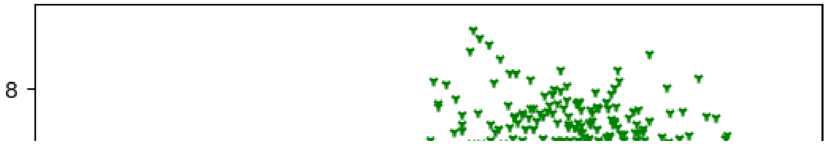
```
[[1.2378157]
 [1.6589917]]
[-12.722816]
```

```
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 0x7d1f18b09cf0> triggered tf.funct
1/1 [==============================] - 0s 67ms/step
array([[1.]], 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 = 1, 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, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
Epoch 1/10
50/50 [==============================] - 0s 5ms/step - loss: 0.7866 - val_loss: 0.3952
Epoch 2/10
50/50 [==============================] - 0s 2ms/step - loss: 0.4571 - val_loss: 0.2438
Epoch 3/10
50/50 [==============================] - 0s 3ms/step - loss: 0.3674 - val_loss: 0.2025
Epoch 4/10
50/50 [==============================] - 0s 2ms/step - loss: 0.3076 - val_loss: 0.2083
Epoch 5/10
50/50 [==============================] - 0s 3ms/step - loss: 0.2644 - val_loss: 0.2335
Epoch 6/10
50/50 [==============================] - 0s 2ms/step - loss: 0.2302 - val_loss: 0.2031
Epoch 7/10
50/50 [==============================] - 0s 2ms/step - loss: 0.2054 - val_loss: 0.1173
Epoch 8/10
50/50 [==============================] - 0s 3ms/step - loss: 0.1858 - val_loss: 0.1901
Epoch 9/10
50/50 [==============================] - 0s 2ms/step - loss: 0.1700 - val_loss: 0.1430
```

```
Epoch 10/10
50/50 [==============================] - 0s 2ms/step - loss: 0.1575 - val_loss: 0.1046
```

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

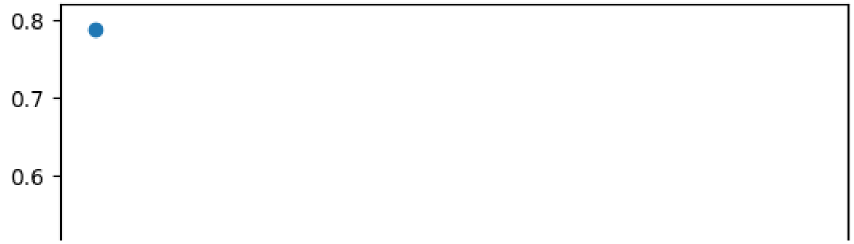
```
[0.7865601181983948,
 0.45706140995025635,
 0.3674454987049103,
 0.3076130747795105,
 0.26441678404808044,
 0.23018169403076172,
 0.20535224676132202,
 0.18581746518611908,
 0.16995273530483246,
 0.15750856697559357]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

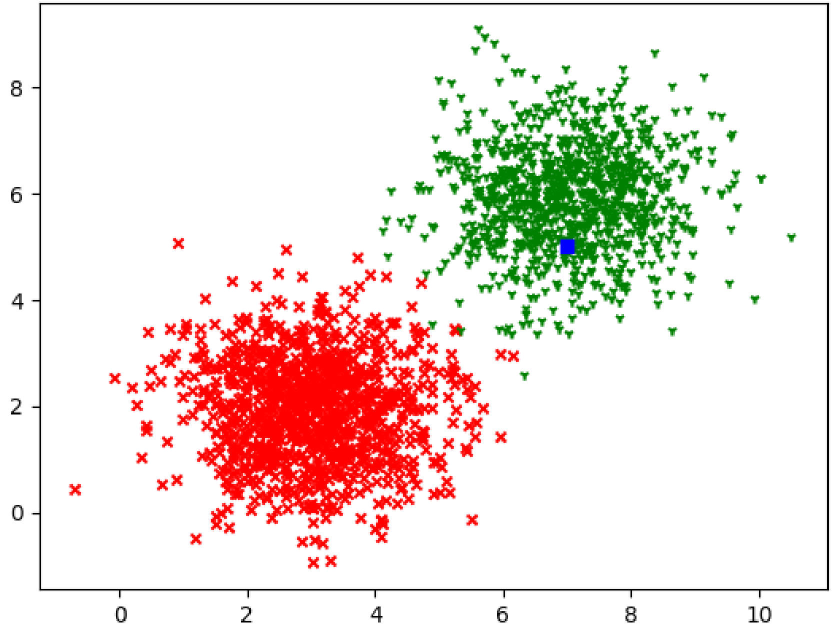
```
[[0.29183912]
 [0.81965226]]
[-4.3890634]
```

```
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 68ms/step
array([[1.]], 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 = 1, 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, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
160/160 [==============================] - 0s 2ms/step - loss: 0.0174 - val_loss: 0.0139
Epoch 88/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0175 - val_loss: 0.0231
Epoch 89/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0175 - val_loss: 0.0184
Epoch 90/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0173 - val_loss: 0.0262
Epoch 91/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0168 - val_loss: 0.0132
Epoch 92/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0171 - val_loss: 0.0185
Epoch 93/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0170 - val_loss: 0.0206
Epoch 94/100
160/160 [==============================] - 0s 3ms/step - loss: 0.0166 - val_loss: 0.0281
Epoch 95/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0169 - val_loss: 0.0195
Epoch 96/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0168 - val_loss: 0.0186
Epoch 97/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0167 - val_loss: 0.0236
Epoch 98/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0165 - val_loss: 0.0227
Epoch 99/100
160/160 [==============================] - 1s 3ms/step - loss: 0.0164 - val_loss: 0.0162
Epoch 100/100
160/160 [==============================] - 0s 2ms/step - loss: 0.0165 - val_loss: 0.0178
```

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

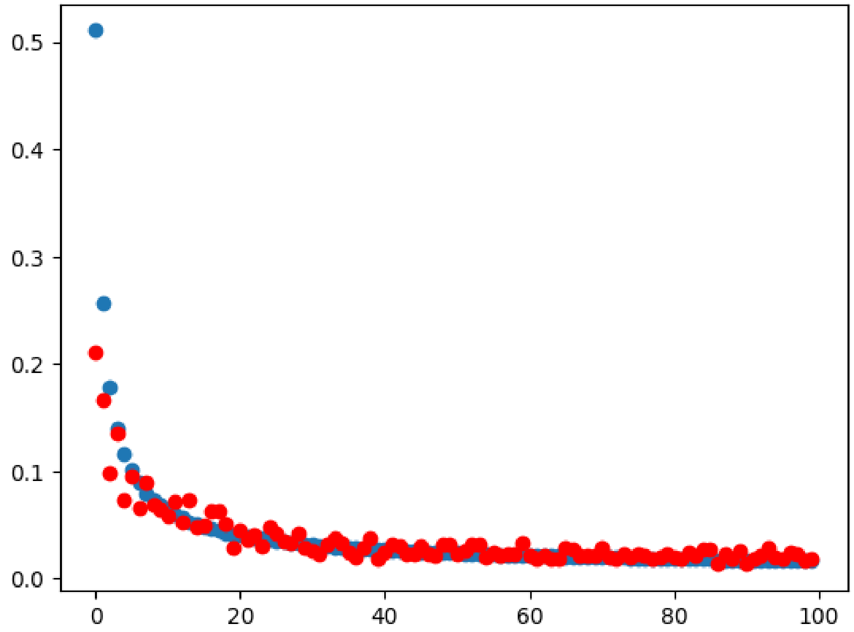
```
       0.020663216710090637,
       0.020555077120661736,
       0.020151283591985703,
       0.02014980837702751,
       0.02003246359527111,
       0.019833596423268318,
       0.019581222906708717,
       0.019504787400364876,
       0.019469643011689186,
       0.01912849023938179,
       0.01909606158733368,
       0.018833132460713387,
       0.018953680992126465,
       0.018806057050824165,
       0.018672440201044083,
       0.018537817522883415,
       0.01850062981247902,
       0.01818443089723587,
       0.018275076523423195,
       0.017901865765452385,
       0.01781781017780304,
       0.017891157418489456,
       0.01760541833937168,
       0.0173624437302351,
       0.01749018393456936,
       0.01753355748951435,
       0.017299102619290352,
       0.016815299168229103,
       0.017127567902207375,
       0.01703708805143833,
       0.016594650223851204,
       0.01686752215027809,
       0.016810324043035507,
       0.016653338447213173,
       0.01652146875858307,
       0.016429556533694267,
       0.016491206362843513]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

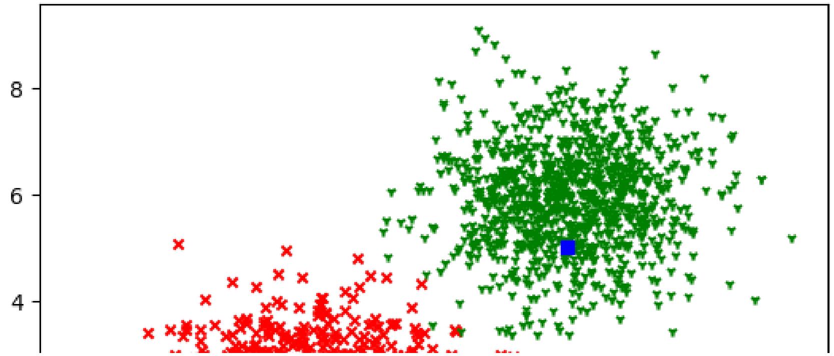
```
[[1.418916 ]
 [1.8477014]]
[-14.371391]
```

```
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([[1.]], 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 = 1, 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, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
Epoch 78/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0272 - val_loss: 0.0314
Epoch 79/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0267 - val_loss: 0.0299
Epoch 80/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0266 - val_loss: 0.0245
Epoch 81/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0265 - val_loss: 0.0315
Epoch 82/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0263 - val_loss: 0.0258
Epoch 83/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0261 - val_loss: 0.0285
Epoch 84/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0260 - val_loss: 0.0246
Epoch 85/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0258 - val_loss: 0.0244
Epoch 86/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0258 - val_loss: 0.0284
Epoch 87/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0254 - val_loss: 0.0292
Epoch 88/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0254 - val_loss: 0.0292
Epoch 89/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0249 - val_loss: 0.0383
Epoch 90/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0249 - val_loss: 0.0206
Epoch 91/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0248 - val_loss: 0.0225
Epoch 92/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0246 - val_loss: 0.0256
Epoch 93/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0245 - val_loss: 0.0279
Epoch 94/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0244 - val_loss: 0.0261
Epoch 95/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0241 - val_loss: 0.0265
Epoch 96/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0240 - val_loss: 0.0211
Epoch 97/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0239 - val_loss: 0.0275
Epoch 98/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0239 - val_loss: 0.0235
Epoch 99/100
80/80 [==============================] - 0s 3ms/step - loss: 0.0234 - val_loss: 0.0275
Epoch 100/100
80/80 [==============================] - 0s 2ms/step - loss: 0.0233 - val_loss: 0.0185
```

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

```
         0.03918968141078949,
         0.03852969780564308,
         0.03800367936491966,
         0.03734232485294342,
         0.036846622824668884,
         0.03639788553118706,
         0.036081910133361816,
         0.03542118892073631,
         0.035184700042009354,
```

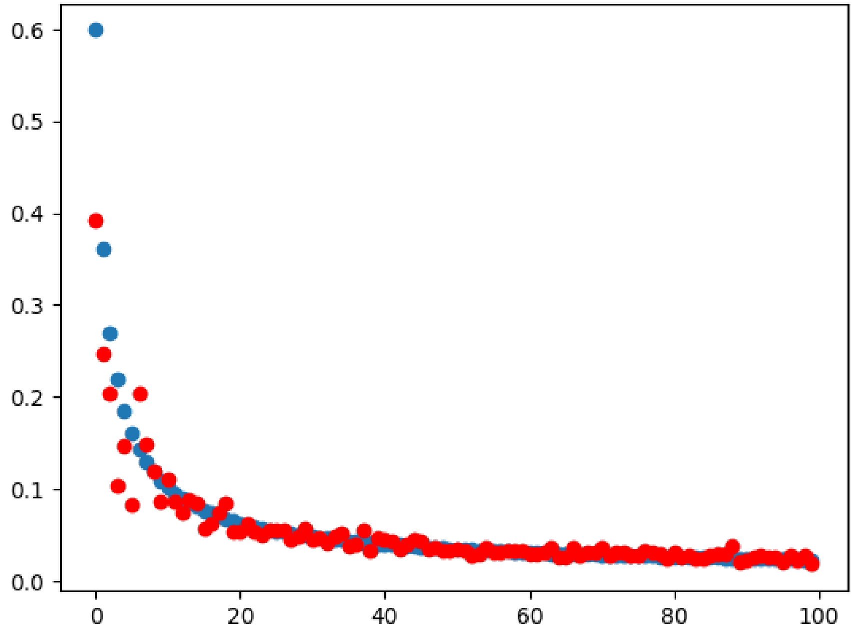
```
0.024368232116103172,
0.02413579449057579,
0.023975389078259468,
0.023930862545967102,
0.02390175312757492,
0.023398425430059433,
0.023250408470630646]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

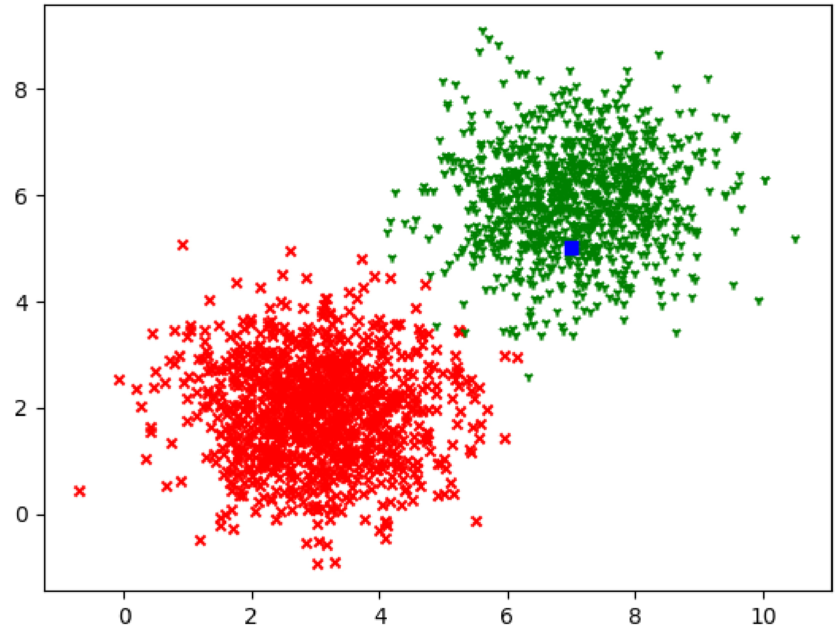
```
[[1.1901294]
 [1.6065422]]
[-12.003977]
```

```
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 56ms/step
array([[1.]], 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 = 1, 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, 1)                 3         
                                                                 
=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 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([[4.59911355, 0.97664184],
       [5.12942909, 1.3036076 ],
```

```
       [2.86082513, 3.42102928],
       ...,
       [6.95434383, 6.18464347],
       [6.07670661, 6.40313423],
       [8.22009436, 6.28592032]])
```

Proces **uczenia**:

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

```
Epoch 91/100
32/32 [==============================] - 0s 3ms/step - loss: 0.0431 - val_loss: 0.0368
Epoch 92/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0429 - val_loss: 0.0454
Epoch 93/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0425 - val_loss: 0.0476
Epoch 94/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0423 - val_loss: 0.0460
Epoch 95/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0421 - val_loss: 0.0426
Epoch 96/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0417 - val_loss: 0.0453
Epoch 97/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0413 - val_loss: 0.0402
Epoch 98/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0410 - val_loss: 0.0455
Epoch 99/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0410 - val_loss: 0.0438
Epoch 100/100
32/32 [==============================] - 0s 4ms/step - loss: 0.0406 - val_loss: 0.0423
```

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

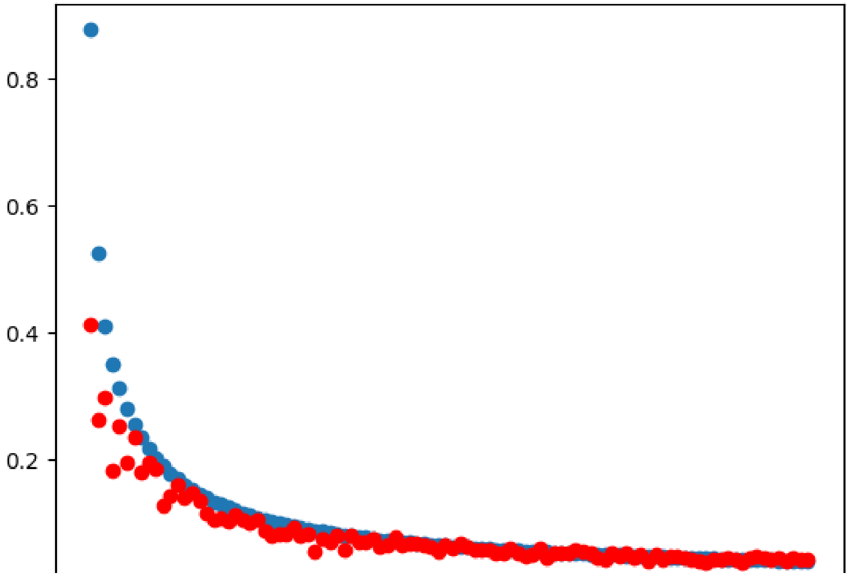
```
      0.05102117359638214,
      0.050185270607471466,
      0.049949631094932556,
      0.04948272183537483,
      0.04902293533086777,
      0.048761483281850815,
      0.04820432513952255,
      0.04773036018013954,
      0.047397080808877945,
      0.04695950821042061,
      0.04658462479710579,
      0.04623636603355408,
      0.04589775577187538,
      0.04555024951696396,
      0.045185331255197525,
      0.04477369785308838,
      0.04453828185796738,
      0.04416794702410698,
      0.0438012070953846,
      0.04348074272274971,
      0.043111030012369156,
      0.04287131875753403,
      0.0424988716840744,
      0.0423225462436676,
      0.04207630082964897,
      0.04169486090540886,
      0.04128500074148178,
      0.041049886494874954,
      0.04099629074335098,
      0.040603771805763245]
```

Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

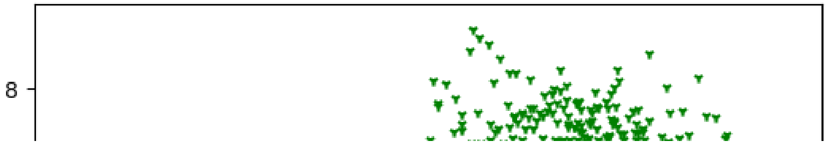
```
[[0.8336686]
 [1.2803266]]
[-9.240546]
```

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



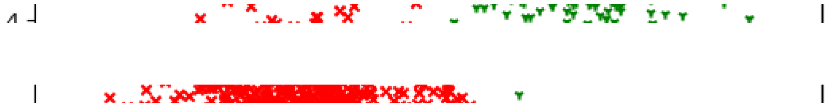
Sprawdzamy działanie modelu dla punktu o współrzędnych **x** i **y**:

```
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 58ms/step
array([[1.]], 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