ZAD 1

```
import tensorflow as tf
```

```
x = tf.Variable(4.0)
y = tf.Variable(3.0)

with tf.GradientTape() as tape:
    f = (x**3)+(y**2)
    df_dx,df_dy = tape.gradient(f,(x,y))

print(df_dx.numpy())
print(df_dy.numpy())
```

```
48.0
6.0
```

ZAD 2

```
x = tf.Variable(1.0)
y = tf.Variable(2.0)

with tf.GradientTape() as tape:
    f = 4*(x**3)+11*(y**2)+9*y*x+10
    df_dx,df_dy = tape.gradient(f,(x,y))

print(df_dx.numpy())
print(df_dy.numpy())
```



```
53.0
```

Zad 3

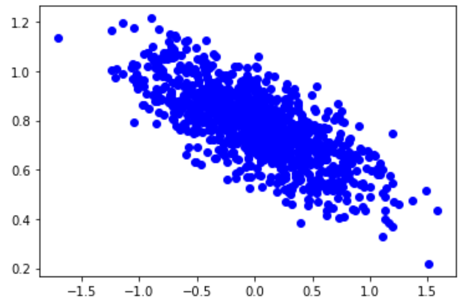
```
import matplotlib.pyplot as plt
import numpy as np
```

```
number_of_points = 1000
x_point = []
y_point = []
```

```
a = -0.22
b = 0.78
```

```
for i in range(number_of_points):
    x = np.random.normal(0.0,0.5)
    y = (a*x+b)+np.random.normal(0.0,0.1)
    x_point.append(x)
    y_point.append(y)
```

```
plt.scatter(x_point,y_point,c='b')
plt.show()
```




```
real_x = np.array(x_point)
real_y = np.array(y_point)
```

```
def loss_fn(real_y, pred_y):
    return tf.reduce_mean((real_y - pred_y)**2)
```

```
import·random
a·=·tf.Variable(random.random())
b·=·tf.Variable(random.random())
```

```
Loss = []
epochs = 1000
learning_rate = 0.01

for _ in range(epochs):
                                   :
    loss = loss_fn(real_y, pred_y)
    Loss.append(loss.numpy())

  dloss_da, dloss_db = tape.gradient(loss,(a, b))

  a.assign_sub(learning_rate*dloss_da)  #a = a - alpha*dloss_da
  b.assign_sub(learning_rate*dloss_db)  #b = b - alpha*dloss_db
```

Zapisano pomyślnie.

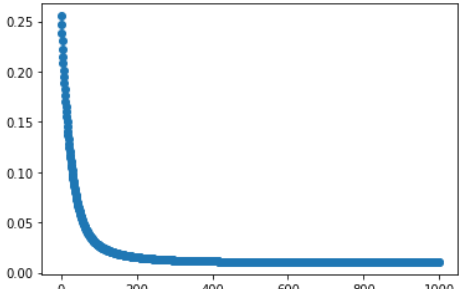
```
np.max(Loss),np.min(Loss)
```

    (0.2553488, 0.010641033)

```
print(a.numpy())
print(b.numpy())
```

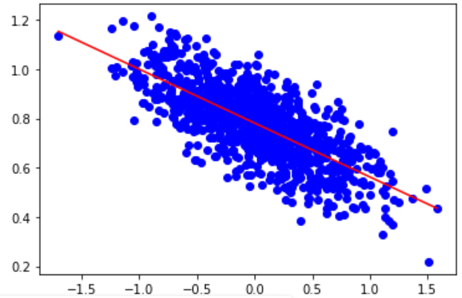
    -0.21870963
    0.78094

```
plt.scatter(np.arange(epochs),Loss)
plt.show()
```




```
max = np.max(x_point)
min = np.min(x_point)


X = np.linspace(min, max, num=10)
plt.plot(X,a.numpy()*X+b.numpy(),c='r')
plt.scatter(x_point,y_point,c="b")
plt.show()
```




```
def subset_dataset(x_dataset, y_dataset, subset_size):
    arr = np.arange(len(x_dataset))
    np.random.shuffle(arr)
    x_train = x_dataset[arr[0:subset_size]]
    y_train = y_dataset[arr[0:subset_size]]
    return x_train,y_train


a = tf.Variable(random.random())
b = tf.Variable(random.random())


Loss = []
epochs = 1000
learning_rate = 0.2
batch_size = 50

for i in range(epochs):
  real_x_batch,real_y_batch = subset_dataset(real_x,real_y,batch_size)
  with tf.GradientTape() as tape:
    pred_y = a * real_x_batch + b
    loss = loss_fn(real_y_batch, pred_y)
    Loss.append(loss.numpy())
```

```
  dloss_da, dloss_db = tape.gradient(loss,(a, b))

  a.assign_sub(learning_rate*dloss_da)  #a = a - alpha*dloss_da
  b.assign_sub(learning_rate*dloss_db)  #b = b - alpha*dloss_db
```

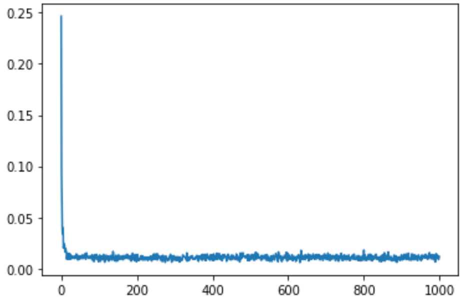
```
np.max(Loss),np.min(Loss)
```

```
(0.24657296, 0.0058904802)
```

```
print(a.numpy())
print(b.numpy())
```

```
-0.20982282
0.77441996
```

```
plt.plot(Loss)
plt.show()
```



Zapisano pomyślnie.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('Boston.csv')
print(df)
```

```
     Unnamed: 0     crim    zn  indus  chas    nox     rm   age     dis  rad  \
0             1  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1
1             2  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2
2             3  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2
3             4  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3
4             5  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3
..          ...      ...   ...    ...   ...    ...    ...   ...     ...  ...
501         502  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1
502         503  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1
503         504  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1
504         505  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1
505         506  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1

     tax  ptratio   black  lstat  medv
0    296     15.3  396.90   4.98  24.0
1    242     17.8  396.90   9.14  21.6
2    242     17.8  392.83   4.03  34.7
```

```
3    222     18.7  394.63   2.94  33.4
4    222     18.7  396.90   5.33  36.2
..   ...      ...     ...    ...   ...
501  273     21.0  391.99   9.67  22.4
502  273     21.0  396.90   9.08  20.6
503  273     21.0  396.90   5.64  23.9
504  273     21.0  393.45   6.48  22.0
505  273     21.0  396.90   7.88  11.9

[506 rows x 15 columns]
```

```
df.head()
```

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
rm=df.iloc[:,6]
```

```
rm
```

```
0      6.575
1      6.421
2      7.185
3      6.998
4      7.147
```

Zapisano pomyślnie.

```
503    6.976
504    6.794
505    6.030
Name: rm, Length: 506, dtype: float64
```

```
medv=df.iloc[:,14]
```

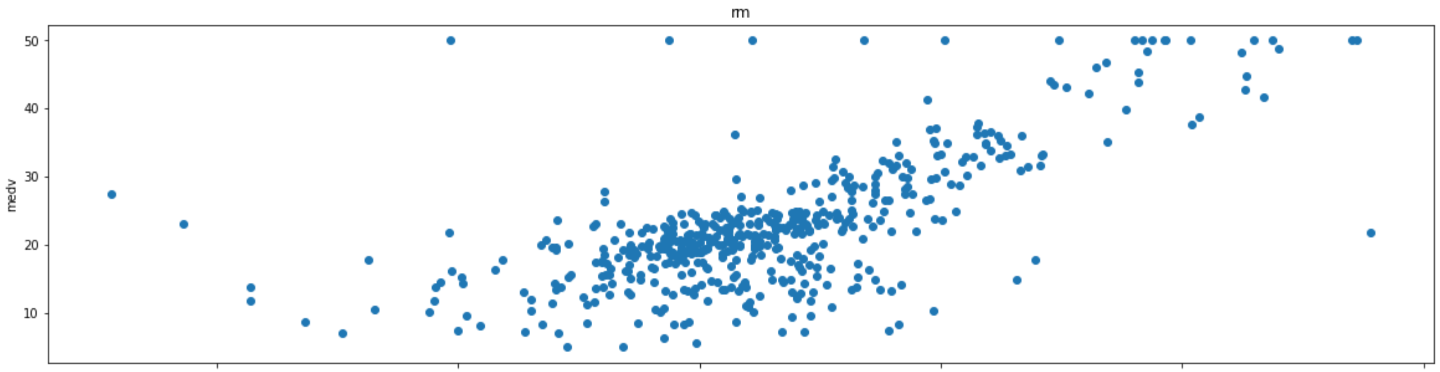
```
medv
```

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: medv, Length: 506, dtype: float64
```

```
plt.figure(figsize=(20, 5))

features = ['rm']
target = df['medv']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('medv')
```




```
real_rm = np.array(rm)
real_medv = np.array(medv)
```

Zapisano pomyślnie.

```
                      6.998, 7.147, 6.43 , 6.012, 6.172, 5.631,
       6.004, 6.377, 6.009, 5.889, 5.949, 6.096, 5.834, 5.935, 5.99 ,
       5.456, 5.727, 5.57 , 5.965, 6.142, 5.813, 5.924, 5.599, 5.813,
       6.047, 6.495, 6.674, 5.713, 6.072, 5.95 , 5.701, 6.096, 5.933,
       5.841, 5.85 , 5.966, 6.595, 7.024, 6.77 , 6.169, 6.211, 6.069,
       5.682, 5.786, 6.03 , 5.399, 5.602, 5.963, 6.115, 6.511, 5.998,
       5.888, 7.249, 6.383, 6.816, 6.145, 5.927, 5.741, 5.966, 6.456,
       6.762, 7.104, 6.29 , 5.787, 5.878, 5.594, 5.885, 6.417, 5.961,
       6.065, 6.245, 6.273, 6.286, 6.279, 6.14 , 6.232, 5.874, 6.727,
       6.619, 6.302, 6.167, 6.389, 6.63 , 6.015, 6.121, 7.007, 7.079,
       6.417, 6.405, 6.442, 6.211, 6.249, 6.625, 6.163, 8.069, 7.82 ,
       7.416, 6.727, 6.781, 6.405, 6.137, 6.167, 5.851, 5.836, 6.127,
       6.474, 6.229, 6.195, 6.715, 5.913, 6.092, 6.254, 5.928, 6.176,
       6.021, 5.872, 5.731, 5.87 , 6.004, 5.961, 5.856, 5.879, 5.986,
       5.613, 5.693, 6.431, 5.637, 6.458, 6.326, 6.372, 5.822, 5.757,
       6.335, 5.942, 6.454, 5.857, 6.151, 6.174, 5.019, 5.403, 5.468,
       4.903, 6.13 , 5.628, 4.926, 5.186, 5.597, 6.122, 5.404, 5.012,
       5.709, 6.129, 6.152, 5.272, 6.943, 6.066, 6.51 , 6.25 , 7.489,
       7.802, 8.375, 5.854, 6.101, 7.929, 5.877, 6.319, 6.402, 5.875,
       5.88 , 5.572, 6.416, 5.859, 6.546, 6.02 , 6.315, 6.86 , 6.98 ,
       7.765, 6.144, 7.155, 6.563, 5.604, 6.153, 7.831, 6.782, 6.556,
       7.185, 6.951, 6.739, 7.178, 6.8  , 6.604, 7.875, 7.287, 7.107,
       7.274, 6.975, 7.135, 6.162, 7.61 , 7.853, 8.034, 5.891, 6.326,
       5.783, 6.064, 5.344, 5.96 , 5.404, 5.807, 6.375, 5.412, 6.182,
       5.888, 6.642, 5.951, 6.373, 6.951, 6.164, 6.879, 6.618, 8.266,
       8.725, 8.04 , 7.163, 7.686, 6.552, 5.981, 7.412, 8.337, 8.247,
```

```
            6.726, 6.086, 6.631, 7.358, 6.481, 6.606, 6.897, 6.095, 6.358,
            6.393, 5.593, 5.605, 6.108, 6.226, 6.433, 6.718, 6.487, 6.438,
            6.957, 8.259, 6.108, 5.876, 7.454, 8.704, 7.333, 6.842, 7.203,
            7.52 , 8.398, 7.327, 7.206, 5.56 , 7.014, 8.297, 7.47 , 5.92 ,
            5.856, 6.24 , 6.538, 7.691, 6.758, 6.854, 7.267, 6.826, 6.482,
            6.812, 7.82 , 6.968, 7.645, 7.923, 7.088, 6.453, 6.23 , 6.209,
            6.315, 6.565, 6.861, 7.148, 6.63 , 6.127, 6.009, 6.678, 6.549,
            5.79 , 6.345, 7.041, 6.871, 6.59 , 6.495, 6.982, 7.236, 6.616,
            7.42 , 6.849, 6.635, 5.972, 4.973, 6.122, 6.023, 6.266, 6.567,
            5.705, 5.914, 5.782, 6.382, 6.113, 6.426, 6.376, 6.041, 5.708,
            6.415, 6.431, 6.312, 6.083, 5.868, 6.333, 6.144, 5.706, 6.031,
            6.316, 6.31 , 6.037, 5.869, 5.895, 6.059, 5.985, 5.968, 7.241,
            6.54 , 6.696, 6.874, 6.014, 5.898, 6.516, 6.635, 6.939, 6.49 ,
            6.579, 5.884, 6.728, 5.663, 5.936, 6.212, 6.395, 6.127, 6.112,
            6.398, 6.251, 5.362, 5.803, 8.78 , 3.561, 4.963, 3.863, 4.97 ,
            6.683, 7.016, 6.216, 5.875, 4.906, 4.138, 7.313, 6.649, 6.794,
            6.38 , 6.223, 6.968, 6.545, 5.536, 5.52 , 4.368, 5.277, 4.652,
            5.   , 4.88 , 5.39 , 5.713, 6.051, 5.036, 6.193, 5.887, 6.471,
            6.405, 5.747, 5.453, 5.852, 5.987, 6.343, 6.404, 5.349, 5.531,
            5.683, 4.138, 5.608, 5.617, 6.852, 5.757, 6.657, 4.628, 5.155,
            4.519, 6.434, 6.782, 5.304, 5.957, 6.824, 6.411, 6.006, 5.648,
            6.103, 5.565, 5.896, 5.837, 6.202, 6.193, 6.38 , 6.348, 6.833,
            6.425, 6.436, 6.208, 6.629, 6.461, 6.152, 5.935, 5.627, 5.818,
            6.406, 6.219, 6.485, 5.854, 6.459, 6.341, 6.251, 6.185, 6.417,
            6.749, 6.655, 6.297, 7.393, 6.728, 6.525, 5.976, 5.936, 6.301,
            6.081, 6.701, 6.376, 6.317, 6.513, 6.209, 5.759, 5.952, 6.003,
            5.926, 5.713, 6.167, 6.229, 6.437, 6.98 , 5.427, 6.162, 6.484,
            5.304, 6.185, 6.229, 6.242, 6.75 , 7.061, 5.762, 5.871, 6.312,
            6.114, 5.905, 5.454, 5.414, 5.093, 5.983, 5.983, 5.707, 5.926,
            5.67 , 5.39 , 5.794, 6.019, 5.569, 6.027, 6.593, 6.12 , 6.976,
            6.794, 6.03 ])
```

```
import random
a = tf.Variable(random.random())
b = tf.Variable(random.random())
```

Zapisano pomyślnie.

```
                               dataset, subset_size):
                               t))
    x_train = x_dataset[arr[0:subset_size]]
    y_train = y_dataset[arr[0:subset_size]]
    return x_train,y_train
```

```
def loss_fn(real_y, pred_y):
    return tf.reduce_mean((real_y - pred_y)**2)
```

```
Loss = []
epochs = 1000
learning_rate = 0.01
batch_size = 50
a = tf.Variable(random.random())
b = tf.Variable(random.random())
for _ in range(epochs):
  real_rm_batch,real_medv_batch = subset_dataset(real_rm,real_medv,batch_size)
  with tf.GradientTape() as tape:
    pred_medv = a * real_rm_batch + b
    loss = loss_fn(real_medv_batch, pred_medv)
    Loss.append(loss.numpy())
```

```
    dloss_da, dloss_db = tape.gradient(loss,(a, b))

    a.assign_sub(learning_rate*dloss_da)  #a = a - alpha*dloss_da
    b.assign_sub(learning_rate*dloss_db)  #b = b - alpha*dloss_db
```

```
  Loss
```

Zapisano pomyślnie.

```
 48.695568]
```

```
np.max(Loss),np.min(Loss)
```

```
(226.16956, 17.60775)
```

```
print(a.numpy())
print(b.numpy())
```

```
4.855049
-6.8641143
```

```
import tensorflow as tf
import keras
from keras.layers import Dense
from keras.models import Sequential
```

```
model=Sequential()
```

```
model.add(Dense(units = 3, use_bias=True, input_dim=1, activation = "linear"))
model.add(Dense(units = 1, use_bias=True,  activation = "linear"))
```

```
opt = tf.keras.optimizers.Adam()
```

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

```
model.summary()
```

Zapisano pomyślnie.

```
Model: "sequential_2"
_________________________________________________________________
                             Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 3)                 6

 dense_5 (Dense)             (None, 1)                 4

=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

```
epochs = 200
h = model.fit(real_rm,real_medv, verbose=1, epochs=epochs, batch_size=100, validation_split=0.3)
```

```
Epoch 1978/2000
4/4 [==============================] - 0s 14ms/step - loss: 31.8383 - val_loss: 117.2317
Epoch 1979/2000
4/4 [==============================] - 0s 14ms/step - loss: 31.8297 - val_loss: 117.3754
Epoch 1980/2000
4/4 [==============================] - 0s 13ms/step - loss: 31.8108 - val_loss: 117.2357
Epoch 1981/2000
4/4 [==============================] - 0s 13ms/step - loss: 31.7972 - val_loss: 117.0044
Epoch 1982/2000
4/4 [==============================] - 0s 14ms/step - loss: 31.7866 - val_loss: 116.8757
Epoch 1983/2000
4/4 [==============================] - 0s 26ms/step - loss: 31.7723 - val_loss: 116.7933
Epoch 1984/2000
4/4 [==============================] - 0s 16ms/step - loss: 31.7638 - val_loss: 116.7077
Epoch 1985/2000
4/4 [==============================] - 0s 13ms/step - loss: 31.7491 - val_loss: 116.5757
Epoch 1986/2000
4/4 [==============================] - 0s 20ms/step - loss: 31.7377 - val_loss: 116.3435
Epoch 1987/2000
4/4 [==============================] - 0s 18ms/step - loss: 31.7256 - val_loss: 116.3480
Epoch 1988/2000
4/4 [==============================] - 0s 13ms/step - loss: 31.7122 - val_loss: 116.4285
Epoch 1989/2000
4/4 [==============================] - 0s 14ms/step - loss: 31.6996 - val_loss: 116.4728
Epoch 1990/2000
4/4 [==============================] - 0s 13ms/step - loss: 31.6871 - val_loss: 116.4474
Epoch 1991/2000
4/4 [==============================] - 0s 12ms/step - loss: 31.6710 - val_loss: 116.3184
Epoch 1992/2000
4/4 [==============================] - 0s 20ms/step - loss: 31.6593 - val_loss: 116.2374
Epoch 1993/2000
4/4 [==============================] - 0s 14ms/step - loss: 31.6477 - val_loss: 116.2057
Epoch 1994/2000
4/4 [==============================] - 0s 16ms/step - loss: 31.6423 - val_loss: 116.3429
Epoch 1995/2000
4/4 [==============================] - 0s 21ms/step - loss: 31.6209 - val_loss: 116.3812
Epoch 1996/2000
4/4 [==============================] - 0s 15ms/step - loss: 31.6100 - val_loss: 116.4358
Epoch 1997/2000
=======] - 0s 19ms/step - loss: 31.6124 - val_loss: 116.1545
4/4 [==============================] - 0s 14ms/step - loss: 31.5878 - val_loss: 116.3866
Epoch 1999/2000
4/4 [==============================] - 0s 22ms/step - loss: 31.5705 - val_loss: 116.5085
Epoch 2000/2000
4/4 [==============================] - 0s 23ms/step - loss: 31.5529 - val_loss: 116.4900
```



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


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

Zadanie 5

```
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]
```

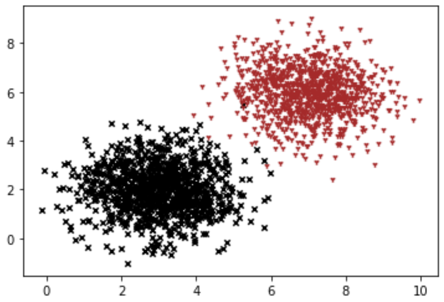
```
x_label1 = np.random.normal(3, 1, 1000)
y_label1 = np.random.normal(2, 1, 1000)
x_label2 = np.random.normal(7, 1, 1000)
y_label2 = np.random.normal(6, 1, 1000)

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))
```

Zapisano pomyślnie.

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

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




```
x_label1
```

```
       2.87281707,  1.58367747,  3.88420014,  2.55478095,  2.15807149,
       3.68333439,  1.72726028,  3.5687913 ,  4.11233697,  3.4268136 ,
       4.34640378,  1.11017676,  4.72302785,  2.38920203,  1.72423622,
       2.6277729 ,  1.90169701,  3.88267508,  3.19224728,  2.80271136,
       3.26185706,  4.14708234,  2.13777457,  3.49365545,  2.95707999,
       3.94331876,  4.56257098,  4.71592755,  2.93449384,  4.43588127,
       2.84955382,  4.46323793,  3.72131222,  3.42329685,  1.5978364 ,
       2.06669596,  0.33129155,  3.18902836,  2.84576703,  2.28069717,
       2.54315546,  2.45393176,  4.27172168,  3.2105997 ,  1.08797434,
       3.27322157,  2.47407349,  1.60518693,  3.91271626,  2.57253787,
       5.27979087,  5.19637644,  2.81445142,  3.91785737,  2.19161609,
       3.1032017 ,  2.78165144,  3.91222323,  2.9074205 ,  0.71802897,
       3.49427462,  3.99103004,  2.78834914,  1.30893805,  0.99988316,
       1.28291969,  4.59641549,  2.6847928 ,  2.2708045 ,  3.86596076,
       2.77583486,  3.66522755,  1.6574295 ,  1.44131345,  4.23399857,
       2.49379452,  2.14041886,  3.79886781,  1.56992793,  1.03418774,
       0.75989873,  2.48087853,  1.10347658,  1.7630807 ,  2.88598722,
       3.82178592,  1.02117055,  4.13785461,  2.67148084,  4.00108366,
       3.14714379,  4.01738423,  2.03058159,  2.57304191,  0.46285633,
       2.56250028,  2.6544916 ,  2.45094681,  4.68283622,  3.83983391,
       2.75697716,  2.37161601,  2.58818045,  3.27242581,  1.30909939,
       1.1080503 ,  3.58581309,  2.63878388,  2.12278776,  2.98894983,
       1.53419394,  3.24595535,  2.78491338,  3.54888456,  1.16552902,
       2.47424555,  3.42832533,  4.25289977,  3.20434172,  2.89171427,
       4.079381  ,  2.6296629 ,  3.95225452,  3.72636608,  0.81669239,
       1.99588903,  2.7842241 ,  2.44901538,  2.07113121,  3.41106595,
       3.9667961 ,  2.45221547,  3.58416967,  2.2026057 ,  3.7215302 ,
       1.85012214,  2.2415439 ,  4.36729253,  3.56902538,  4.3053372 ,
       3.27928897,  3.07575764,  2.18250951,  3.34065765,  4.07485405,
       1.75997117,  2.35460534,  5.21761163,  1.97432106,  4.09112363,
       2.96309054,  4.82713095,  5.44049085,  3.193873  ,  2.71496862,
       1.48216256,  3.06225466,  2.76872803,  2.74827979,  2.50485769,
       3.56090193,  2.94064316,  3.70564494,  1.23118426,  3.08804943,
       3.873968  ,  1.9280286 ,  0.92230025,  4.38478808,  2.04370817,
       2.79130384,  3.34177788,  1.41790157,  3.24371491,  3.74387096,
       4.18621567,  1.85310555,  4.01338475,  3.85075824,  1.96085005,
       1.54160883,  3.91336923,  3.53630974,  2.96490427,  3.74137926,
       3.29787954,  2.38921092,  1.82682565,  2.89370624,  3.37207996,
       [illegible]776,  2.4451369 ,  1.71567175,  3.78650402,
       [illegible]23 ,  4.46883244,  3.28194622,  3.40911055,
       [illegible]413,  1.94615701,  3.97992716,  2.85120918,
       2.24087004,  4.79425189,  3.99799999,  2.94840562,  2.59159485,
       2.72767023,  2.80850526,  3.82803249,  2.191248  ,  3.63904696,
       2.51397208,  2.65637164,  2.06660909,  5.42443334,  3.42021299,
       4.69339589,  2.20703154,  3.13961752,  2.7993742 ,  3.44402463,
       3.41376432,  2.74718167,  4.05958279,  4.68769567,  4.00423397,
       4.0045521 ,  1.98128776,  2.40154679,  2.69665852,  3.43046345,
       3.74958418,  2.54697451,  3.94427921,  2.94442926,  1.47467114,
       1.96645403,  3.20024764,  3.44180545,  1.63205641,  2.29347082,
       2.67973256,  2.47049549,  4.62231314,  0.62969365,  3.26837516,
       1.28053677,  2.23173266,  2.95407448,  3.44306979,  2.52977458,
       2.53313444,  1.89916989,  3.53748238,  1.28165328,  2.51552522,
       2.74305287,  1.18674762,  3.15975725,  2.8072833 ,  4.17357146,
       3.10223007,  2.14850283,  1.34221072,  1.43333184,  2.53062359,
       2.80768534,  3.80335609,  4.34062626,  4.14949433,  2.81851019,
       2.53913192,  3.68062807,  2.42823222,  1.74293025,  1.90434109,
       1.93195264,  3.64054835,  2.07841897,  2.79898905,  2.56837916,
       1.02610678,  3.25838022,  3.51232235,  2.57539136,  1.9215747 ])
```



```
def loss_fn_grad(y, y_model):
 return tf.reduce_mean(-y*tf.math.log(y_model)-(1-y)*tf.math.log(1-y_model))
```

```
def subset_dataset_2(x_dataset, y_dataset,label,subset_size):
    arr = np.arange(len(x_dataset))
```

```
    np.random.shuffle(arr)
    x_train = x_dataset[arr[0:subset_size]]
    y_train = y_dataset[arr[0:subset_size]]
    label_train = label[arr[0:subset_size]]
    return x_train,y_train,label_train
```

```
labels.shape
```

```
(2000,)
```

```
Loss = []
epochs = 1000
learning_rate = 0.01
batch_size = 50
a = tf.Variable(random.random())
b = tf.Variable(random.random())
c = tf.Variable(random.random())
for _ in range(epochs):
  xs_batch,ys_batch,labels_batch = subset_dataset_2(xs,ys,labels,batch_size)
  with tf.GradientTape() as tape:
    pred_l = tf.sigmoid(a * xs_batch + b * ys_batch + c)
    #print(label_batch.shape)
    loss = loss_fn_grad(labels_batch, pred_l)
    Loss.append(loss.numpy())

  dloss_da, dloss_db, dloss_dc = tape.gradient(loss,(a, b,c))

  a.assign_sub(learning_rate*dloss_da)  #a = a - alpha*dloss_da
  b.assign_sub(learning_rate*dloss_db)  #b = b - alpha*dloss_db
  c.assign_sub(learning_rate*dloss_dc)
```

```
np.max(Loss),np.min(Loss)
```

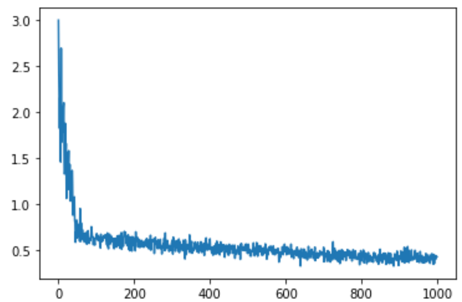
```
(2.9938192, 0.32082856)
```

Zapisano pomyślnie.

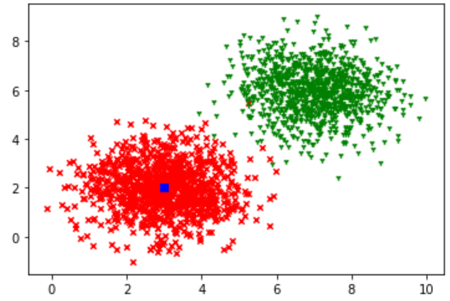
```
print(a.numpy())
print(b.numpy())
```

```
-0.058886457
0.52488345
```

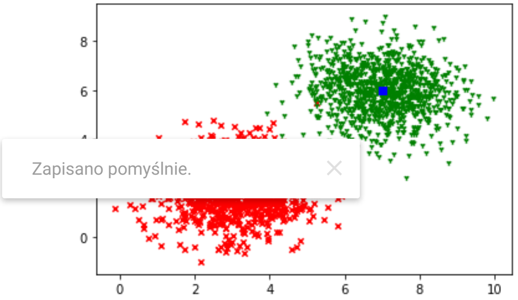
```
plt.plot(Loss)
plt.show()
```



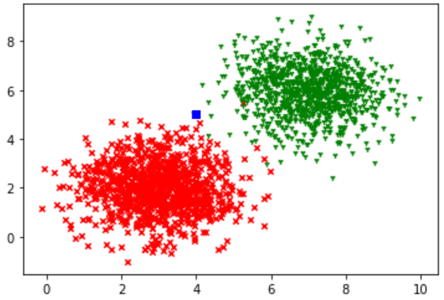
```
x=3.0
y=2.0
plt.scatter(x_label1, y_label1, c='r', marker='x', s=20)
plt.scatter(x_label2, y_label2, c='g', marker='1', s=20)
plt.scatter(x,y,c='b', marker='s')
plt.show()
```




```
x=7.0
y=6.0
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()
```




```
x=4.0
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()
```



✓ 7 s ukończono o 15:47

Zapisano pomyślnie.