Przykład 1 (liczenie gradientu)

```
import tensorflow as tf
import math
```

1. Policz gradient (pochodną) funkcji $f(x,y) = 4x^3 + 11x^2 + 9x + 10$ w punkcie (1,2).

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

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

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

```
43.0
```

2. Policz gradient funkcji:

$$f(x,y,z) = xye^{x^2+z^2-5}$$

w dowolnym punkcie. Wykorzystaj `tf.math.exp()`.

```
x = tf.Variable(1.0)
y = tf.Variable(2.0)
z = tf.Variable(3.0)
```

```
with tf.GradientTape() as tape:
    f = x*y*math.exp((x**2)+(z**2)-5)
    df_dx,df_dy,df_dz = tape.gradient(f,(x,y,z))

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

Zapisano pomyślnie.

Policz gradient funkcji $f(x,y) = 2x^3 + 3y^2 + 4$ w punkcie $(4,5)$.

```
x = tf.Variable(4.0)
y = tf.Variable(5.0)

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

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

```
96.0
30.0
```

Przykład 2 (regresja liniowa - liczenie gradientów)

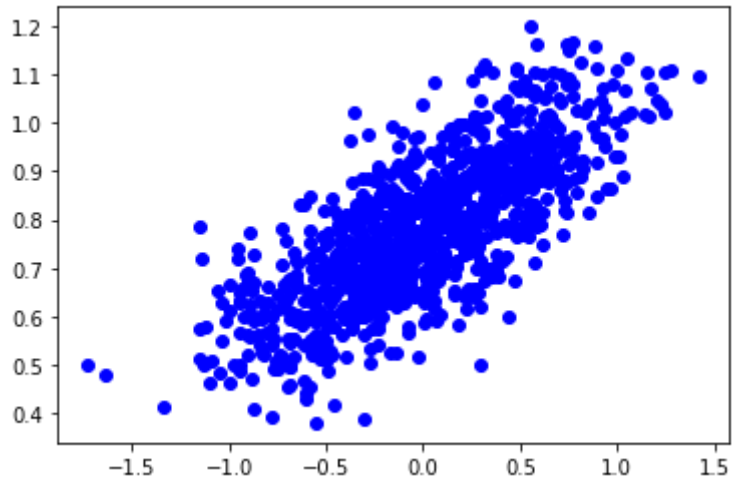
```
import matplotlib.pyplot as plt
import numpy as np
```

Wygenrujemy **zbiór danych** złożony z 1000 punktów.

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



```
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)
```

Definicja błędu:

```
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())
```

Pętla treningowa:

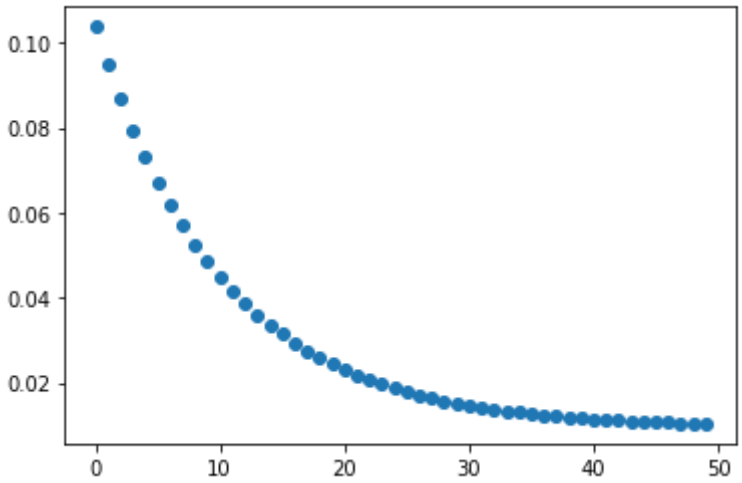
```
Loss = []
epochs = 50

for _ in range(epochs):
  with tf.GradientTape() as tape:
    pred_y = a * real_x + b
    loss = loss_fn(real_y, pred_y)
    Loss.append(loss.numpy())

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

  a.assign_sub(dloss_da * 0.1)  #a = a - alpha*dloss_da
  b.assign_sub(dloss_db * 0.1)  #b = b - alpha*dloss_db


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

```
    (0.10372553, 0.010425453)
```

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

```
0.2878917
0.7810024
```

Zapisano pomyślnie.




```
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()
```

Przykład 3 (regresja liniowa - Keras)

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

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=1, activation = "linear"))
```

Definiujemy **optymalizator** i **błąd** (średni błąd kwadratowy - MSE). **Współczynnik uczenia = 0.1**

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

Kompilacja modelu:

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

Informacje o modelu:


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Ustalamy **liczbę epok** w czasie której **model będzie uczony**.

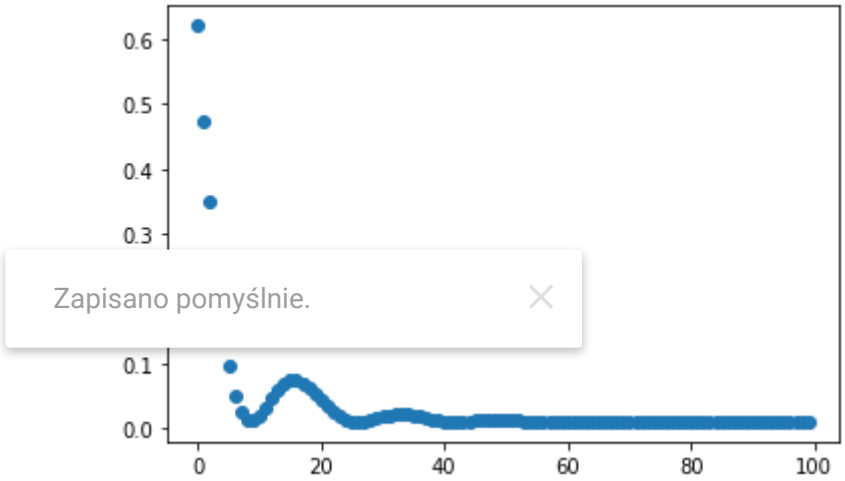
```
number_epochs=100
```

Proces **uczenia**:

```
h = model.fit(real_x,real_y, verbose=0, epochs=number_epochs, batch_size=1000)
```

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

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

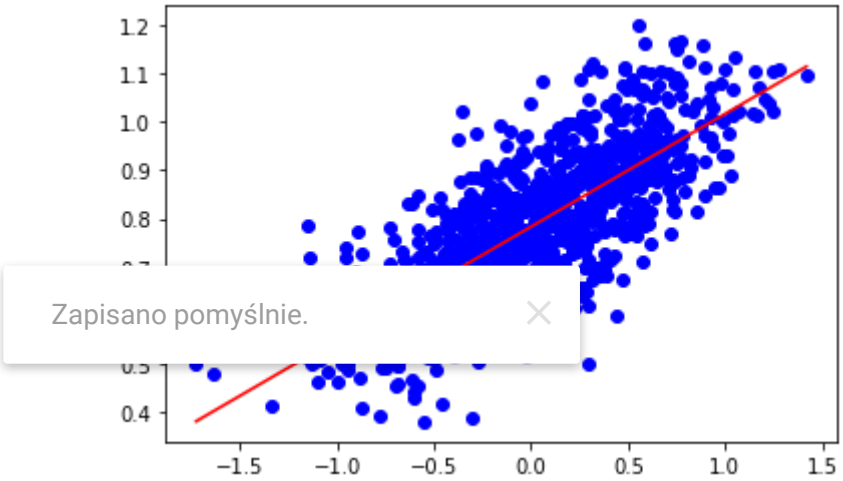


Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

a = weights[0][0][0]
b = weights[1][0]    #bias

a,b
```

```
(0.23248206, 0.7835197)
```

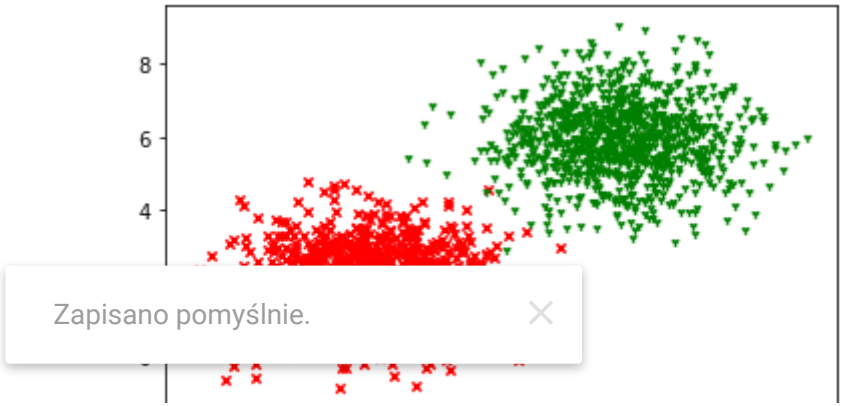
```
max = np.max(x_point)
min = np.min(x_point)


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



Przykład 4 (regresja logistyczna - liczenie gradientów)

Zbiór danych:

```
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))


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()
```



Definiujemy funkcję błędu (entropia krzyżowa):

```
def loss_fn(label, label_model):
    return tf.reduce_mean(-label*tf.math.log(label_model)-(1-label)*tf.math.log(1-label_model))
```

Parametry modelu:

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

Pętla treningowa:

```
Loss = []
epochs = 2000
lr = 0.01
for _ in range(epochs):
  with tf.GradientTape() as tape:
    labels_model = tf.sigmoid(a*xs + b*ys + c)
```

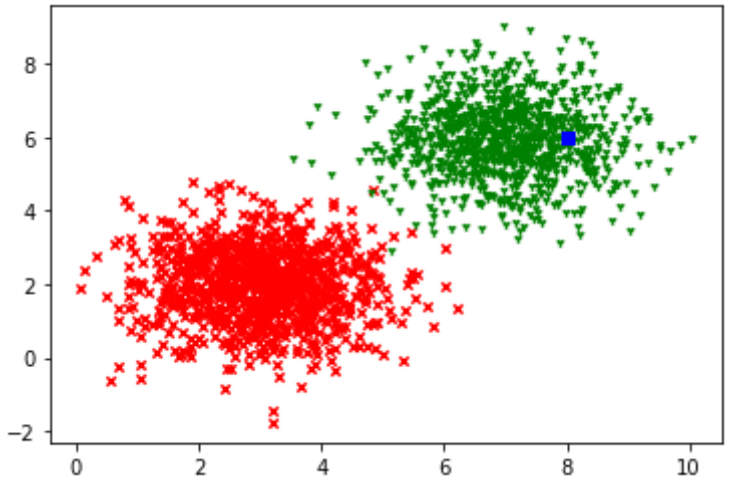
```
    loss = loss_fn(labels, labels_model)
    Loss.append(loss.numpy())

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

  a.assign_sub(lr*dloss_da)
  b.assign_sub(lr*dloss_db)
  c.assign_sub(lr*dloss_dc)
```



```
    (1.2138839, 0.30223346)


print(a.numpy())
print(b.numpy())
print(c.numpy())

    0.0034050525
    0.6508277
    -2.0283709


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

Sprawdzamy dla pewnego punktu:

```
x=8.0
y=6.0
plt.scatter(x_label1, y_label1, c='n', marker='x', s=20)
                                ', marker='1', s=20)
                                ')
plt.show()
```




```
tf.sigmoid(a*x + b*y + c).numpy()
```

```
0.8703251
```

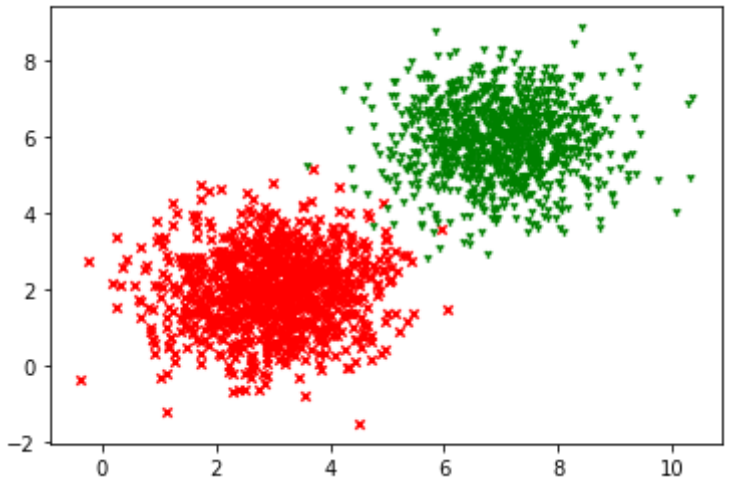
Przykład 5 (regresja logistyczna - Keras)

Zbiór danych:

```
#zbiór czerowny
x_label1 = np.random.normal(3, 1, 1000)
y_label1 = np.random.normal(2, 1, 1000)

#zbiór zielony
x_label2 = np.random.normal(7, 1, 1000)
y_label2 = np.random.normal(6, 1, 1000)
```

Zapisano pomyślnie.

```
ys = np.append(y_label1, y_label2)


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()
```



Zbiór treningowy złożony jest z punktów:

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

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

```
array([[4.5732957 , 1.79717308],
       [3.19622133, 3.39948374],
       [3.13116011, 3.09621451],
       ...,
       [7.15472044, 4.60220741],
                                ,
                                ])
```


Wartości oczekiwane: **0** dla zbioru **czerwonego**,**1** dla zbioru **zielonego**.

```
labels = np.asarray([0.]*len(x_label1)+[1.]*len(x_label2))
labels
```

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

```
data_points.shape,labels.shape
```

```
((2000, 2), (2000,))
```

Definiujemy model:

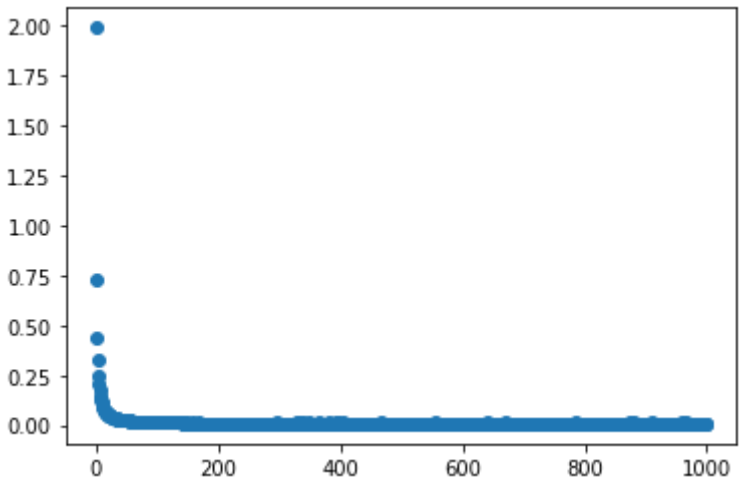
```
model = Sequential()
model.add(Dense(units = 1, use_bias=True, input_dim=2, activation = "sigmoid"))

opt = tf.keras.optimizers.Adam(learning_rate=0.1)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
```

```
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_5 (Dense)              (None, 1)                 3

=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```


Pętla ucząca:

```
epochs = 1000
h = model.fit(data_points,labels, verbose=0, epochs=epochs, batch_size=100)


Loss = h.history['loss']


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

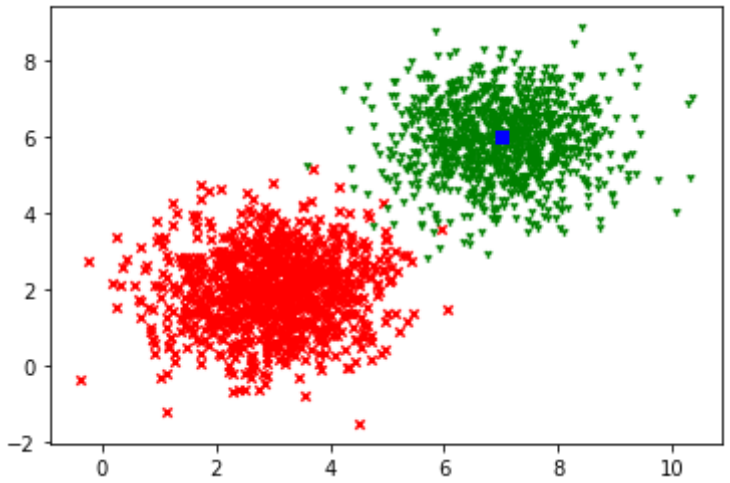


Sprawdzamy dla punktu o współrzędnych:

```
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)
                                                    ')
```

Zapisano pomyślnie.




```
model.predict([[x,y]])
```

```
array([[1.]], dtype=float32)
```

Zapisano pomyślnie.

0 s ukończono o 13:26