Import biblioteki **TensorFlow** ([https://www.tensorflow.org/](https://www.tensorflow.org/)) z której będziemy korzystali w **uczeniu maszynowym**:

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

```
x_point = []
y_point = []
```

```
x_point = [2, 2, 0, -2, -2, 0, 4]
y_point= [1, 2, 6, 10, 0, 0, -20]
d = [1, 1, 1, -1, -1, -1, -1]
```

```
sd = [1, 1, 1, 0, 0, 0, 0]
```

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

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

```
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=2, activation = "sigmoid"))
```

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

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

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

```
model.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 1)                 3

=================================================================
Total params: 3 (12.00 Byte)
Trainable params: 3 (12.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Proces **uczenia**:

```
data = np.column_stack((real_x,real_y))
data_train = np.asarray(data)

d_train = np.asarray(sd)



print(data_train)
```

```
[[  2   1]
 [  2   2]
 [  0   6]
 [ -2  10]
 [ -2   0]
 [  0   0]
 [  4 -20]]
```

```
epochs = 6000
h = model.fit(data_train,d_train, verbose=1, epochs=epochs, batch_size=5)
```

```
Epoch 5987/6000
2/2 [==============================] - 0s 6ms/step - loss: 0.0032
Epoch 5988/6000
2/2 [==============================] - 0s 7ms/step - loss: 0.0032
Epoch 5989/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5990/6000
2/2 [==============================] - 0s 10ms/step - loss: 0.0032
Epoch 5991/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5992/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5993/6000
2/2 [==============================] - 0s 6ms/step - loss: 0.0032
Epoch 5994/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5995/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5996/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5997/6000
2/2 [==============================] - 0s 7ms/step - loss: 0.0032
Epoch 5998/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 5999/6000
2/2 [==============================] - 0s 5ms/step - loss: 0.0032
Epoch 6000/6000
2/2 [==============================] - 0s 7ms/step - loss: 0.0032
```

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

```
        0.018686648458242416,
        0.018167799338698387,
        0.018022330477833748,
        0.018597137182950974,
        0.01821703277528286,
        0.017999349161982536,
        0.017980875447392464,
        0.017964733764529228,
        0.017947468906641006,
        0.018147092312574387,
        0.018033217638731003,
        0.018237752839922905,
        0.01836075261235237,
        0.01794404909014702,
        0.017858130857348442,
        0.0178360678255558,
        0.018029039725661278,
        0.018484771251678467,
        0.0182406734675169,
        0.017978934571146965,
        0.01830093003809452,
        0.017917126417160034,
        0.017983445897698402,
        0.017680881544947624,
        0.017867865040898323,
        0.017906446009874344,
        0.017834339290857315,
        0.01804584264755249,
        0.01816573180258274,
        0.0185080636292696,
        0.01811528019607067,
        0.01772027276456356,
        0.01809767819941044,
        0.017832165583968163,
        0.017701465636491776,
        0.017739685252308846,
        0.0177160631865263,
        ...]
```

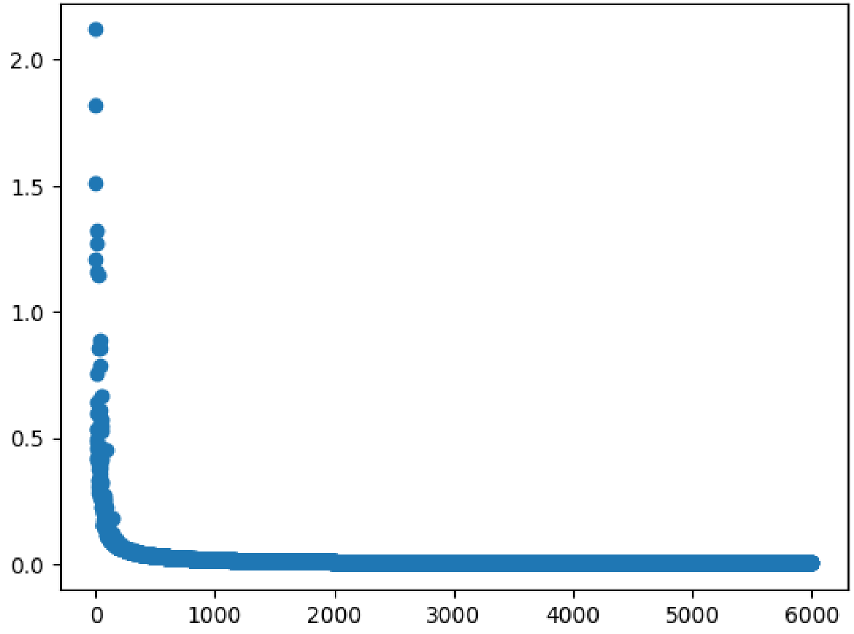
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

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

```
8.200812
-5.3895717
```

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




Sprawdzenie **modelu**:

```
#model.predict([0.6])
```

```
model.predict(data_train)
```

```
1/1 [==============================] - 0s 116ms/step
array([[9.9999696e-01],
       [9.9999946e-01],
       [9.9189115e-01],
       [8.1830053e-03],
       [3.4371894e-10],
       [4.5431927e-03],
       [1.3945935e-03]], dtype=float32)
```

```
X = np.linspace(min, max, num=10)
plt.plot(X,(weights[0][0][0])*X+(weights[1][0]),c='r')
plt.scatter(x_point,y_point,c='b')
plt.show()
```