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
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

Dwa gangi

Przetesuj poniższe instrukcje:

```
[2]*12
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

```
[-3]*10+[4]*5
```

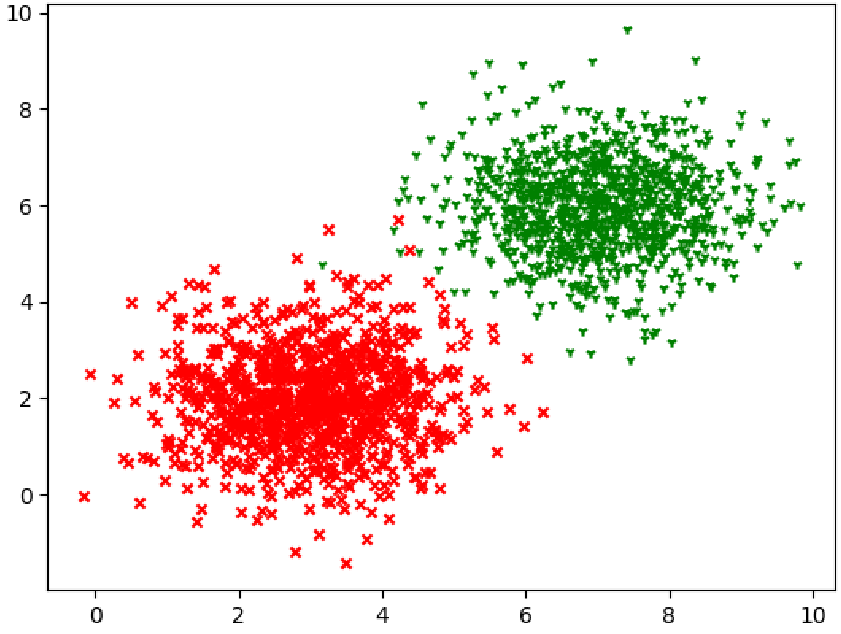
```
[-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, 4, 4, 4, 4, 4]
```

```
np.append([1,2,3],[4,5])
```

```
array([1, 2, 3, 4, 5])
```

Przygotowujemy 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) #tablica wsp. x dla 2000 punktów
ys = np.append(y_label1, y_label2) #tablica wsp. y dla 2000 punktów
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()
```



Przygotowanie danych:

```
xs[0:10].reshape(-1,1)
```

```
array([[3.08456138],
       [3.09235896],
       [2.59064648],
```

```
           [3.80345406],
           [2.4284587 ],
           [3.03407197],
           [3.06640068],
           [3.65832989],
           [2.9812158 ],
           [4.14363636]])
```

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

```
    array([[3.08456138, 2.38798423],
           [3.09235896, 1.52477974],
           [2.59064648, 2.53923651],
           ...,
           [6.72630046, 5.77070292],
           [8.07632849, 6.05743724],
           [7.71097923, 6.29796197]])
```

```
def subset_dataset(data_points, label,subset_size):
    arr = np.arange(len(data_points))
    l=len(data_points)
    s=int(subset_size*l)
    np.random.shuffle(arr)
    data_points_val = data_points[arr[0:s]]
    label_val = label[arr[0:s]]
    #print(type(label_train))
    data_points_train = data_points[arr[s:int(l*(1-subset_size))]]
    label_train = label[arr[s:int(l*(1-subset_size))]]
    data_points_test = data_points[arr[int(l*(1-subset_size)):]]
    label_test = label[arr[int(l*(1-subset_size)):]]
    return data_points_train,label_train,data_points_val,label_val,data_points_test,label_test
```

```
data_points_train,label_train,data_points_val,label_val,data_points_test,label_test = subset_dataset(data_points, labels,0.1)
```

```
print(data_points_train.size,label_train.size,data_points_val.size,label_val.size,data_points_test.size,label_test.size)
```

```
    3200 1600 400 200 400 200
```

Wersja podstawowa

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** (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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_17 (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**:

```
epochs = 100
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val))
```

```
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.0338 - accuracy: 0.9969 - val_loss: 0.0240 - val_accuracy: 1.0000
```

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

```
 0.0567958727478981,
 0.055730681866407394,
 0.055267129093408585,
 0.054217930883169174,
```

```
       0.03600847036957741,
       0.03590194508433342,
       0.03576294332742691,
       0.035543907433748245,
       0.03518076613545418,
       0.034874871373176575,
       0.03483172878623009,
       0.03459509089589119,
       0.03440941870212555,
       0.03417322784662247,
       0.03403875604271889,
       0.03379802033305168]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
        0.029222876179218292,
        0.02847749926149845,
        0.030848747119307518,
        0.029196403920650482,
        0.0273799579590559,
        0.0271052997559309,
        0.02749774232506752,
        0.027541721239686012,
        0.026402167975902557,
        0.026233287528157234,
        0.02603345923125744,
        0.025337379425764084,
        0.02559799700975418,
        0.024784423410892487,
        0.024433257058262825,
        0.025787053629755974,
        0.023953523486852646,
        0.025009674951434135,
        0.024577956646680832,
        0.025236638262867928,
        0.02456455677747264,
        0.023642180487513542,
        0.024047985672950745]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

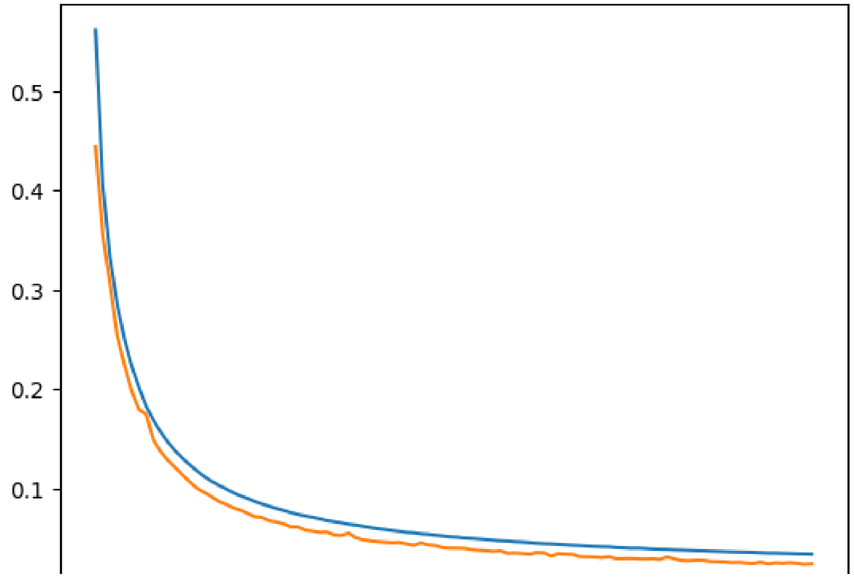
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()
```

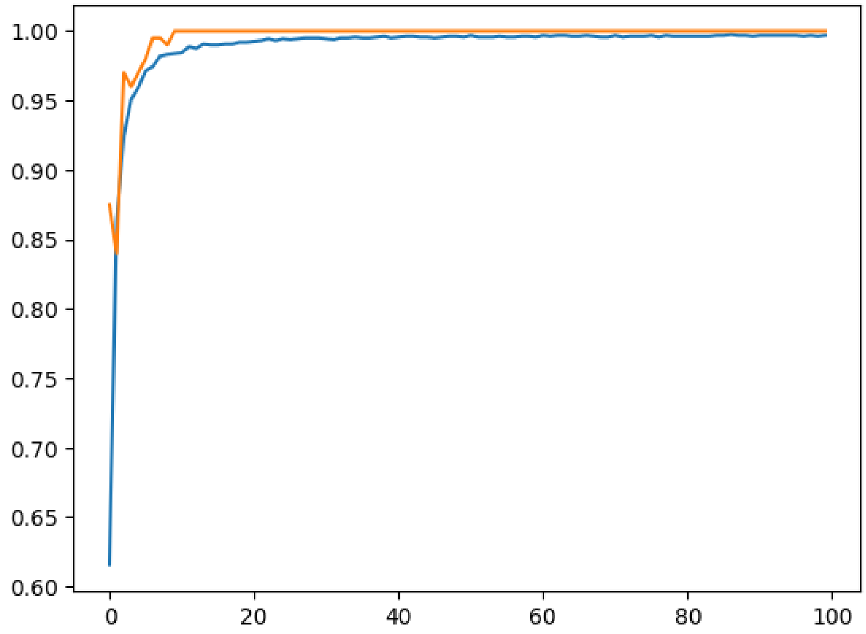
```
print(weights[0])
print(weights[1])    #bias
```

```
[[1.033173 ]
 [1.3348391]]
[-10.38845]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```




```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 4ms/step - loss: 0.0297 - accuracy: 0.9950
test loss, test acc: [0.02971092239022255, 0.9950000047683716]
```

Model.predict for test dataset

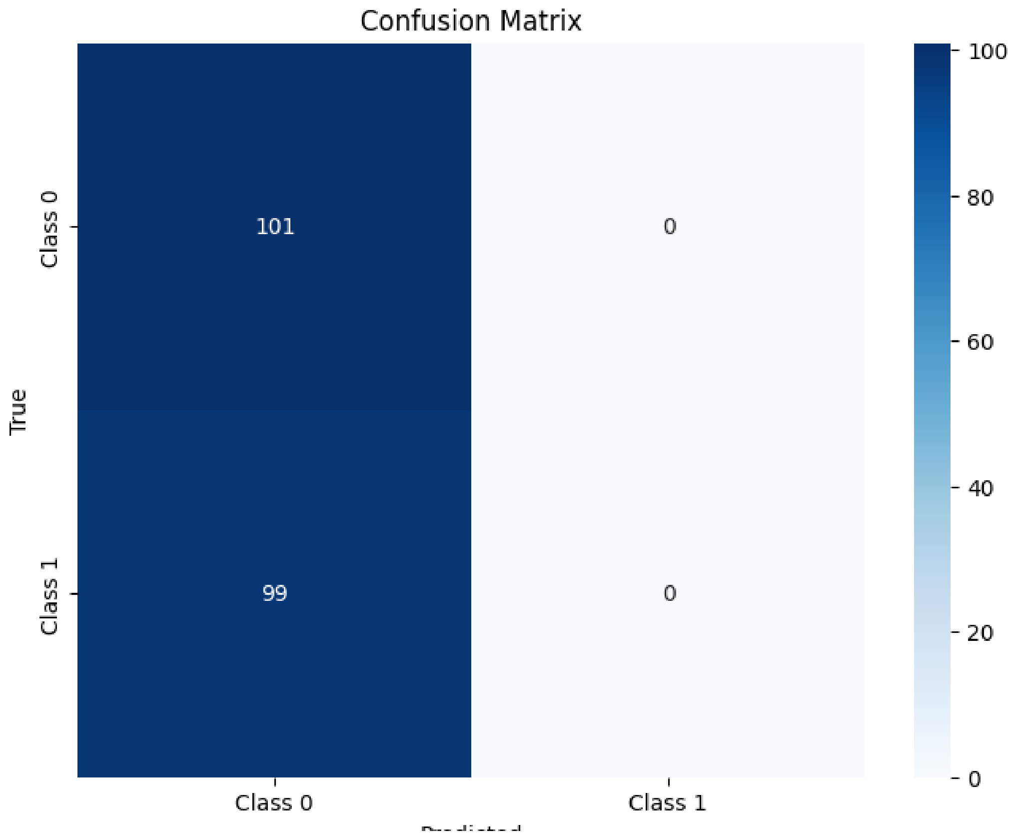
```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 3ms/step
```

```
predictions
```

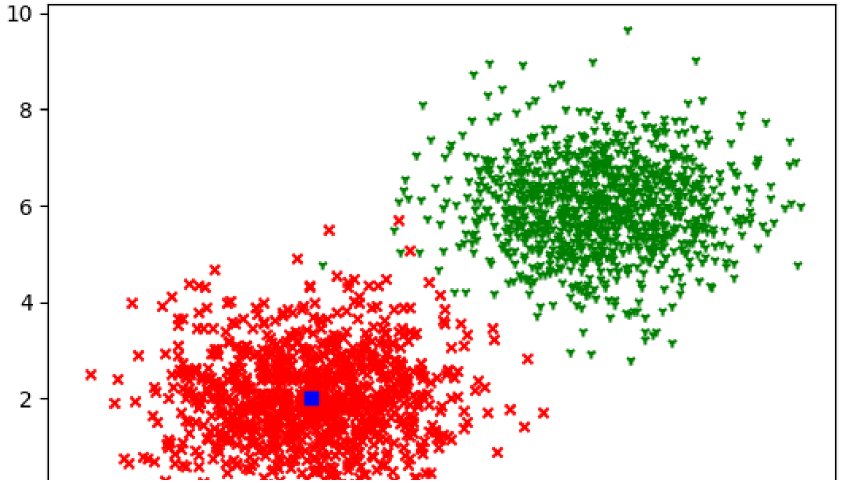
```
       [1.69464853e-02],
       [9.92200553e-01],
       [9.94777679e-01],
       [9.97071385e-01],
       [9.93488312e-01],
       [9.51081634e-01],
       [9.98852909e-01],
       [2.71563288e-02],
       [3.91200855e-02],
       [9.99586284e-01],
       [9.80741799e-01],
       [9.96558368e-01],
       [1.80431269e-03],
       [1.55899988e-03],
       [2.68539321e-03],
       [9.28829312e-01],
       [9.97165143e-01],
       [2.66725402e-02],
       [9.99196231e-01],
       [9.90144372e-01],
       [7.23959506e-02],
       [6.68882905e-03],
       [2.76654726e-03],
       [9.78625715e-01],
       [9.98737514e-01],
       [2.14623529e-02],
       [9.93190289e-01]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



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

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



Liczba epok 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 = "sigmoid"))
```

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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (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**:

```
epochs = 50
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val))
```

```
Epoch 37/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0627 - accuracy: 0.9956 - val_loss: 0.0505 - val_accuracy: 1.0000
Epoch 38/50
50/50 [==============================] - 0s 3ms/step - loss: 0.0617 - accuracy: 0.9956 - val_loss: 0.0487 - val_accuracy: 1.0000
Epoch 39/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0601 - accuracy: 0.9956 - val_loss: 0.0478 - val_accuracy: 1.0000
Epoch 40/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0594 - accuracy: 0.9956 - val_loss: 0.0461 - val_accuracy: 1.0000
Epoch 41/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0584 - accuracy: 0.9950 - val_loss: 0.0452 - val_accuracy: 1.0000
Epoch 42/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0577 - accuracy: 0.9956 - val_loss: 0.0446 - val_accuracy: 1.0000
Epoch 43/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0566 - accuracy: 0.9962 - val_loss: 0.0449 - val_accuracy: 1.0000
Epoch 44/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0559 - accuracy: 0.9956 - val_loss: 0.0438 - val_accuracy: 1.0000
Epoch 45/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0550 - accuracy: 0.9956 - val_loss: 0.0421 - val_accuracy: 1.0000
Epoch 46/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0543 - accuracy: 0.9962 - val_loss: 0.0420 - val_accuracy: 1.0000
Epoch 47/50
50/50 [==============================] - 0s 3ms/step - loss: 0.0533 - accuracy: 0.9956 - val_loss: 0.0433 - val_accuracy: 1.0000
Epoch 48/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0529 - accuracy: 0.9962 - val_loss: 0.0404 - val_accuracy: 1.0000
Epoch 49/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0522 - accuracy: 0.9956 - val_loss: 0.0424 - val_accuracy: 1.0000
Epoch 50/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0516 - accuracy: 0.9956 - val_loss: 0.0400 - val_accuracy: 1.0000
```

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

```
[0.7431750297546387,
 0.42075684666633606,
 0.3319653272628784,
 0.2831713855266571,
 0.24753496050834656,
 0.21947048604488373,
 0.19833961129188538,
 0.18198128044605255,
 0.1674334704875946,
 0.1562575399875641,
 0.14546747505664825,
 0.13699114322662354,
 0.13025763630867004,
 0.12270532548427582,
 0.11712243407964706,
 0.11182510107755661,
 0.10709516704082489,
 0.10287774354219437,
 0.09905615448951721,
 0.0958385020494461,
```

```
 0.09217116981744766,
 0.08943009376525879,
 0.08668186515569687,
 0.08339094370603561,
 0.08212518692016602,
 0.07953925430774689,
 0.07759203761816025,
 0.07564333081245422,
 0.0743584930896759,
 0.07212219387292862,
 0.0704309493303299,
 0.06908492743968964,
 0.06761016696691513,
 0.06617526710033417,
 0.06514616310596466,
 0.06357365101575851,
 0.062660351395607,
 0.06166725233197212,
 0.06013286113739014,
 0.05939648300409317,
 0.058405570685863495,
 0.057665131986141205,
 0.0565909817814827,
 0.05585955083370209,
 0.05500354617834091,
 0.054296430200338364,
 0.053337644785642624,
 0.05292847007513046,
 0.05219866707921028,
 0.05159341171383858]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
[0.4609641134738922,
 0.3653091490268707,
 0.28971487283706665,
 0.25181570649147034,
 0.2194747030735016,
 0.2002658098936081,
 0.17580966651439667,
 0.1598692685365677,
 0.1538253277540207,
 0.13765177130699158,
 0.12648901343345642,
 0.12142007797956467,
 0.11565384268760681,
 0.1060359925031662,
 0.10148147493600845,
 0.09559199213981628,
 0.09183896332979202,
```

```
        0.08639652281999588,
        0.09225808829069138,
        0.07992543280124664,
        0.07794103026390076,
        0.07511506229639053,
        0.07196810841560364,
        0.06878525763750076,
        0.06759601831436157,
        0.06602486968040466,
        0.06282365322113037,
        0.060868483036756516,
        0.06080273166298866,
        0.05763554945588112,
        0.060901228338479996,
        0.05631069839000702,
        0.05466238409280777,
        0.053258877247571945,
        0.052473507821559906,
        0.05487394332885742,
        0.05045425519347191,
        0.0486861951649189,
        0.04776332527399063,
        0.046147264540195465,
        0.0452188141644001,
        0.044550538063049316,
        0.044948771595954895,
        0.043800558894872665,
        0.042144130915403366,
        0.0420335978269577,
        0.04327167198061943,
        0.04035782441496849,
        0.042406681925058365,
        0.03995607793331146]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

Sprawdźmy jakie są **wartości wag**:

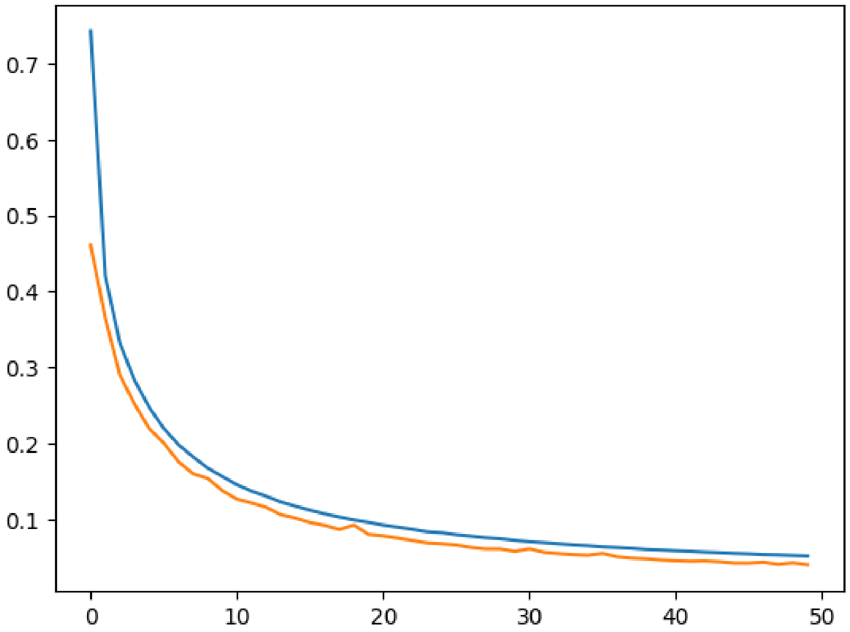
```
weights = model.get_weights()
```

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

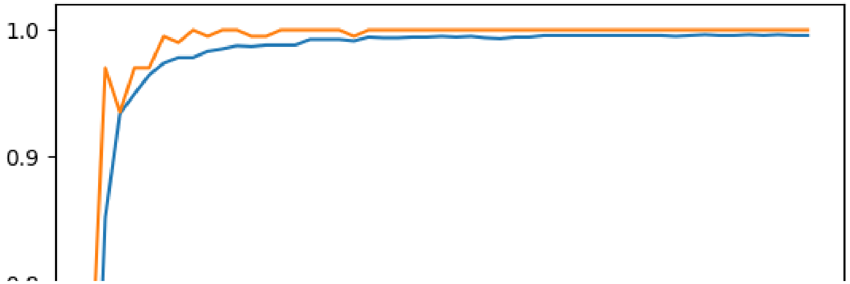
```
[[0.79450125]
 [1.1485249 ]]
```

```
[-8.358202]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```




```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.0471 - accuracy: 0.9950
test loss, test acc: [0.04705239459872246, 0.9950000047683716]
```

Model.predict for test dataset

```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 4ms/step
```

```
predictions
```

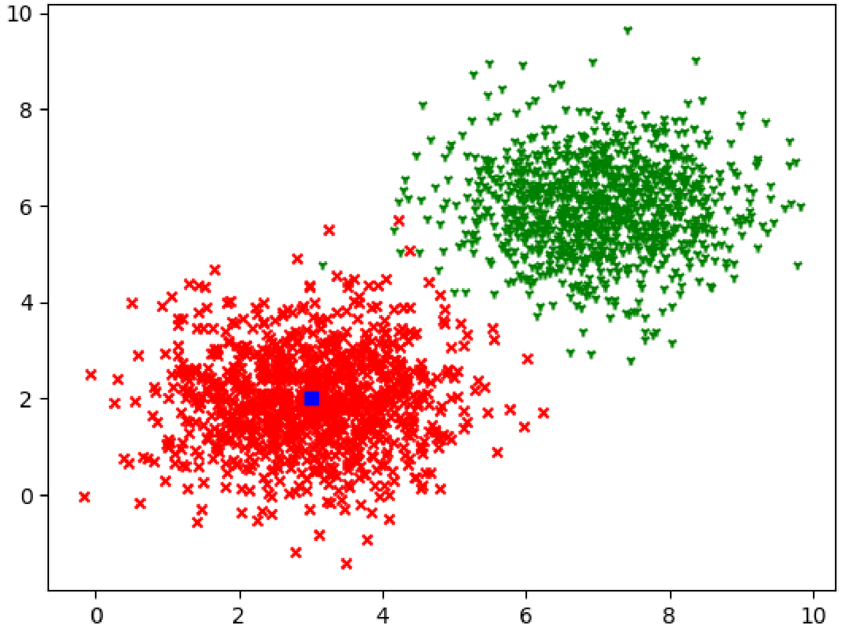
```
       [9.12338436e-01],
       [8.61433625e-01],
       [9.30181816e-02],
       [8.77043232e-03],
       [9.77709472e-01],
       [9.83322859e-01],
       [9.98001933e-01],
       [1.22536253e-02],
       [9.91983056e-01],
       [1.85804628e-02],
       [9.66659307e-01],
       [9.94091034e-01],
       [3.96512598e-02],
       [9.85349298e-01],
       [9.86546278e-01],
       [9.91790950e-01],
       [9.87205565e-01],
       [9.32214081e-01],
       [9.96595562e-01],
       [6.07732981e-02],
       [9.13419649e-02],
       [9.98411179e-01],
       [9.60948408e-01],
       [9.91282701e-01],
       [6.74709212e-03],
       [5.64772170e-03],
       [7.82450195e-03],
       [8.99659157e-01],
       [9.93245006e-01],
       [5.54685742e-02],
       [9.97301936e-01],
       [9.81394351e-01],
       [1.18006781e-01],
       [1.79754067e-02],
       [9.86339897e-03],
       [9.61015940e-01],
       [9.96112883e-01],
       [4.12967354e-02],
       [9.85855281e-01]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

Confusion Matrix

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

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




```
1/1 [==============================] - 0s 50ms/step
array([[0.02465892]], dtype=float32)
```

Liczba epok 150

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** (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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_19 (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**:

```
epochs = 150
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val))
```

```
50/50 [==============================] - 0s 2ms/step - loss: 0.0298 - accuracy: 0.9969 - val_loss: 0.0204 - val_accuracy: 1.0000
Epoch 126/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0299 - accuracy: 0.9969 - val_loss: 0.0208 - val_accuracy: 1.0000
Epoch 127/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0296 - accuracy: 0.9969 - val_loss: 0.0205 - val_accuracy: 1.0000
Epoch 128/150
50/50 [==============================] - 0s 3ms/step - loss: 0.0295 - accuracy: 0.9969 - val_loss: 0.0196 - val_accuracy: 1.0000
Epoch 129/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0294 - accuracy: 0.9962 - val_loss: 0.0205 - val_accuracy: 1.0000
Epoch 130/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0290 - accuracy: 0.9969 - val_loss: 0.0190 - val_accuracy: 1.0000
Epoch 131/150
50/50 [==============================] - 0s 3ms/step - loss: 0.0291 - accuracy: 0.9969 - val_loss: 0.0191 - val_accuracy: 1.0000
Epoch 132/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0289 - accuracy: 0.9962 - val_loss: 0.0215 - val_accuracy: 1.0000
Epoch 133/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0289 - accuracy: 0.9969 - val_loss: 0.0195 - val_accuracy: 1.0000
Epoch 134/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0287 - accuracy: 0.9969 - val_loss: 0.0189 - val_accuracy: 1.0000
Epoch 135/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0285 - accuracy: 0.9962 - val_loss: 0.0216 - val_accuracy: 1.0000
Epoch 136/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0285 - accuracy: 0.9969 - val_loss: 0.0186 - val_accuracy: 1.0000
Epoch 137/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0284 - accuracy: 0.9962 - val_loss: 0.0184 - val_accuracy: 1.0000
Epoch 138/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0283 - accuracy: 0.9969 - val_loss: 0.0185 - val_accuracy: 1.0000
Epoch 139/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0282 - accuracy: 0.9962 - val_loss: 0.0188 - val_accuracy: 1.0000
Epoch 140/150
50/50 [==============================] - 0s 3ms/step - loss: 0.0280 - accuracy: 0.9969 - val_loss: 0.0196 - val_accuracy: 1.0000
Epoch 141/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0282 - accuracy: 0.9969 - val_loss: 0.0185 - val_accuracy: 1.0000
Epoch 142/150
50/50 [==============================] - 0s 3ms/step - loss: 0.0278 - accuracy: 0.9969 - val_loss: 0.0181 - val_accuracy: 1.0000
Epoch 143/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0278 - accuracy: 0.9975 - val_loss: 0.0175 - val_accuracy: 1.0000
Epoch 144/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0277 - accuracy: 0.9962 - val_loss: 0.0175 - val_accuracy: 1.0000
Epoch 145/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0276 - accuracy: 0.9969 - val_loss: 0.0180 - val_accuracy: 1.0000
Epoch 146/150
50/50 [==============================] - 0s 3ms/step - loss: 0.0275 - accuracy: 0.9969 - val_loss: 0.0179 - val_accuracy: 1.0000
Epoch 147/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0274 - accuracy: 0.9969 - val_loss: 0.0180 - val_accuracy: 1.0000
Epoch 148/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0272 - accuracy: 0.9962 - val_loss: 0.0179 - val_accuracy: 1.0000
Epoch 149/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0271 - accuracy: 0.9969 - val_loss: 0.0185 - val_accuracy: 1.0000
Epoch 150/150
50/50 [==============================] - 0s 2ms/step - loss: 0.0271 - accuracy: 0.9969 - val_loss: 0.0174 - val_accuracy: 1.0000
```

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

```
     0.028234068485984704,
     0.027827134355902672,
     0.027760470286011696,
     0.027676617726683617,
     0.02759750373661518,
     0.027501873672008514,
     0.0273530762642622,
     0.02724071592092514,
     0.027088308706879616,
     0.02707619033753872]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
      0.01908051781356334/,
      0.021528147161006927,
      0.019493065774440765,
      0.01889656111598015,
      0.021589193493127823,
      0.018644222989678383,
      0.01839245855808258,
      0.018489381298422813,
      0.018828270956873894,
      0.019572127610445023,
      0.01852303370833397,
      0.018123693764209747,
      0.017501290887594223,
      0.01754179410636425,
      0.017993967980146408,
      0.017877284437417984,
      0.01801108755171299,
      0.01793956570327282,
      0.018485790118575096,
      0.01740364357829094]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

Sprawdźmy jakie są **wartości wag**:

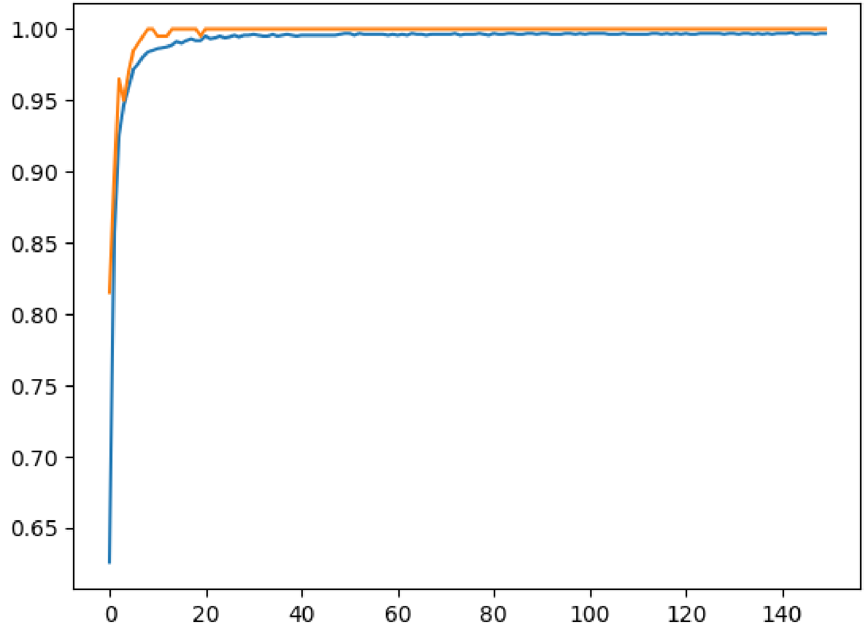
```
weights = model.get_weights()
```

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

```
[[1.1949971]
 [1.4641174]]
[-11.657985]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```


```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.0231 - accuracy: 0.9950
test loss, test acc: [0.02313968911767006, 0.9950000047683716]
```

Model.predict for test dataset

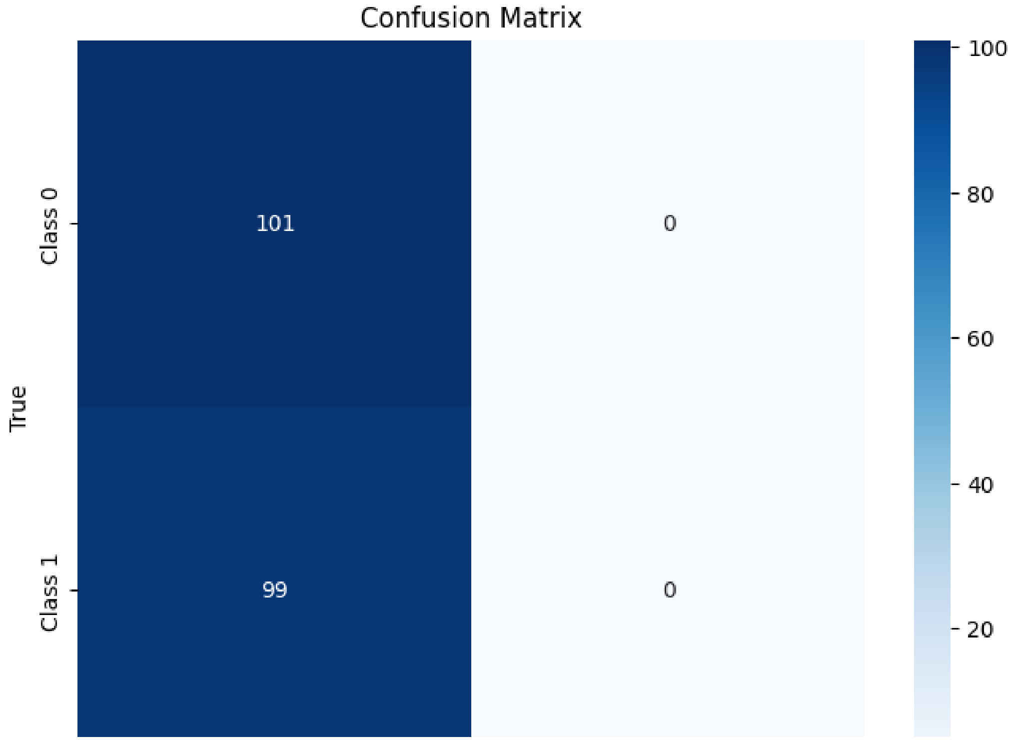
```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 2ms/step
```

```
predictions
```

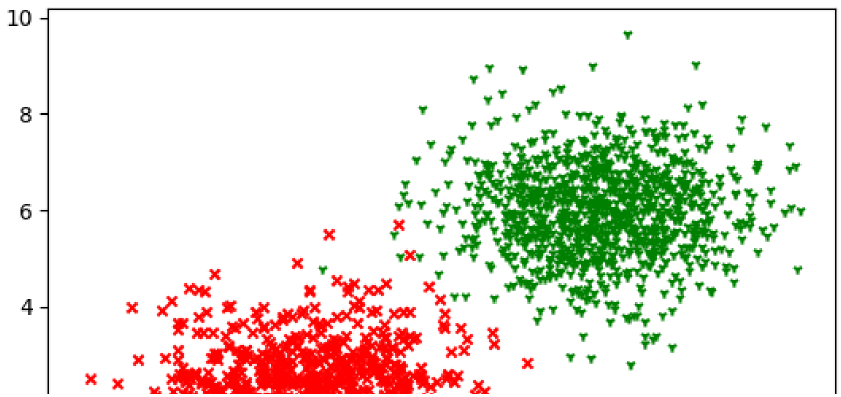
```
       [9.98715639e-01],
       [9.96379197e-01],
       [9.65322495e-01],
       [9.99517441e-01],
       [1.74239278e-02],
       [2.42660698e-02],
       [9.99853730e-01],
       [9.89479482e-01],
       [9.98385489e-01],
       [8.23839684e-04],
       [7.26457103e-04],
       [1.44679565e-03],
       [9.50007915e-01],
       [9.98616755e-01],
       [1.79934707e-02],
       [9.99688625e-01],
       [9.94363189e-01],
       [5.72608076e-02],
       [3.80958151e-03],
       [1.30510447e-03],
       [9.87458467e-01],
       [9.99481261e-01],
       [1.52702741e-02],
       [9.96349275e-01]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

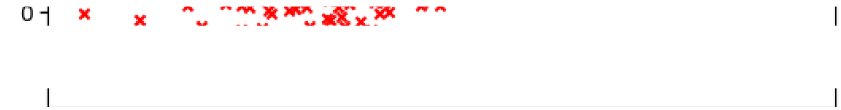


Sprawdzamy działanie modelu dla punktu o współrzędnych **x** i **y**:

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



współczynnik uczenia 0.01 (SGD)



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** (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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (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**:

```
epochs = 100
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val))
```

```
Epoch 88/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1651 - accuracy: 0.9837 - val_loss: 0.1513 - val_accuracy: 1.0000
Epoch 89/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1637 - accuracy: 0.9831 - val_loss: 0.1508 - val_accuracy: 1.0000
Epoch 90/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1626 - accuracy: 0.9837 - val_loss: 0.1491 - val_accuracy: 1.0000
Epoch 91/100
50/50 [==============================] - 0s 4ms/step - loss: 0.1614 - accuracy: 0.9837 - val_loss: 0.1477 - val_accuracy: 1.0000
Epoch 92/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1602 - accuracy: 0.9837 - val_loss: 0.1469 - val_accuracy: 1.0000
Epoch 93/100
50/50 [==============================] - 0s 4ms/step - loss: 0.1590 - accuracy: 0.9837 - val_loss: 0.1459 - val_accuracy: 1.0000
Epoch 94/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1580 - accuracy: 0.9837 - val_loss: 0.1446 - val_accuracy: 1.0000
Epoch 95/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1568 - accuracy: 0.9837 - val_loss: 0.1433 - val_accuracy: 1.0000
Epoch 96/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1557 - accuracy: 0.9837 - val_loss: 0.1422 - val_accuracy: 1.0000
Epoch 97/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1546 - accuracy: 0.9837 - val_loss: 0.1410 - val_accuracy: 1.0000
Epoch 98/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1536 - accuracy: 0.9837 - val_loss: 0.1402 - val_accuracy: 1.0000
Epoch 99/100
50/50 [==============================] - 0s 4ms/step - loss: 0.1525 - accuracy: 0.9837 - val_loss: 0.1394 - val_accuracy: 1.0000
Epoch 100/100
50/50 [==============================] - 0s 3ms/step - loss: 0.1515 - accuracy: 0.9850 - val_loss: 0.1381 - val_accuracy: 1.0000
```

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

```
     0.20056034624576569,
     0.19871696829795837,
     0.19689084589481354,
     0.19502052664756775,
     0.19322887063026428,
     0.1913527548313141,
     0.18969197571277618,
     0.18802137672901154,
     0.1863611787557602,
     0.18480044603347778,
     0.18319088220596313,
     0.1816094070672989,
     0.1801118403673172,
     0.17856159806251526,
     0.17713943123817444,
     0.17575515806674957,
     0.17426328361034393,
     0.1728007197380066,
     0.17155052721500397,
     0.1701667457818985,
     0.16886191070079803,
     0.1675482541322708,
     0.16630998253822327,
     0.16505159437656403,
     0.16371652483940125,
     0.16261248290538788,
     0.16140075027942657,
     0.16020746529102325,
     0.15903154015541077,
     0.15795128047466278,
     0.1568477749824524,
     0.15573464334011078,
     0.15459415316581726,
     0.1535678207874298,
     0.15247337520122528,
     0.15149308741092682]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
 0.21234798431396484,
 0.20964840054512024,
 0.2071646749973297,
 0.20433452725410461,
 0.20235797762870789,
 0.2007608562707901,
 0.1975407749414444,
 0.19560125470161438,
 0.19319257140159607,
 0.19168560206890106,
 0.18973073363304138,
 0.1872919797897339,
 0.18538635969161987,
 0.18398688733577728,
 0.18207313120365143,
 0.1801450550556183,
 0.177986279129982,
 0.1765170693397522,
 0.17483137547969818,
 0.17318256199359894,
 0.17197149991989136,
 0.16969406604766846,
 0.16879194974899292,
 0.1670960783958435,
 0.16612188518047333,
 0.1643739640712738,
 0.16233588755130768,
 0.1609823852777481,
 0.15997417271137238,
 0.15849928557872772,
 0.15729907155036926,
 0.15530230104923248,
 0.15418511629104614,
 0.1529242843389511,
 0.15128645300865173,
 0.1507999300956726,
 0.1491374522447586,
 0.14774233102798462,
 0.14685699343681335,
 0.1459127813577652,
 0.14463120698928833,
 0.14325861632823944,
 0.14220888912677765,
 0.14096786081790924,
 0.14022082090377808,
 0.13941673934459686,
 0.1381445676088333]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

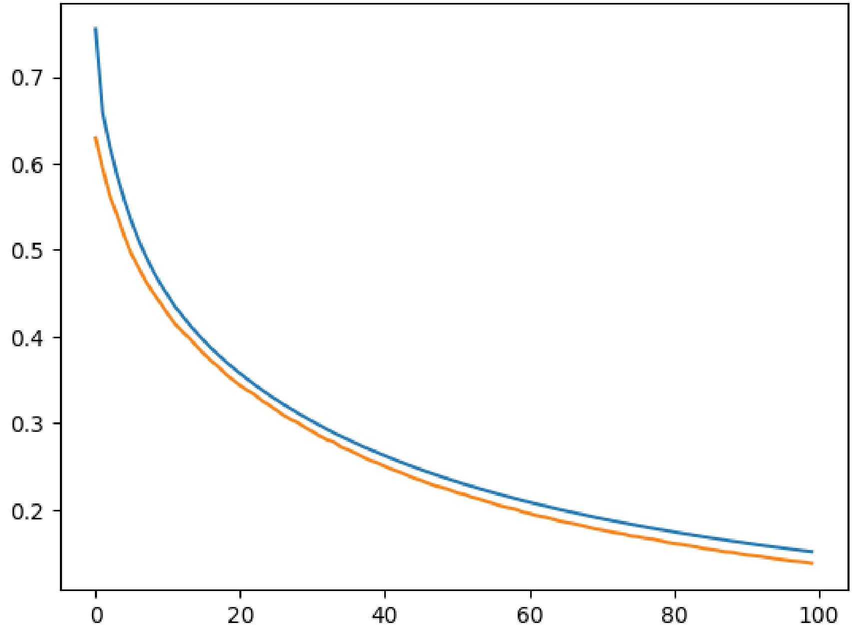
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()
```

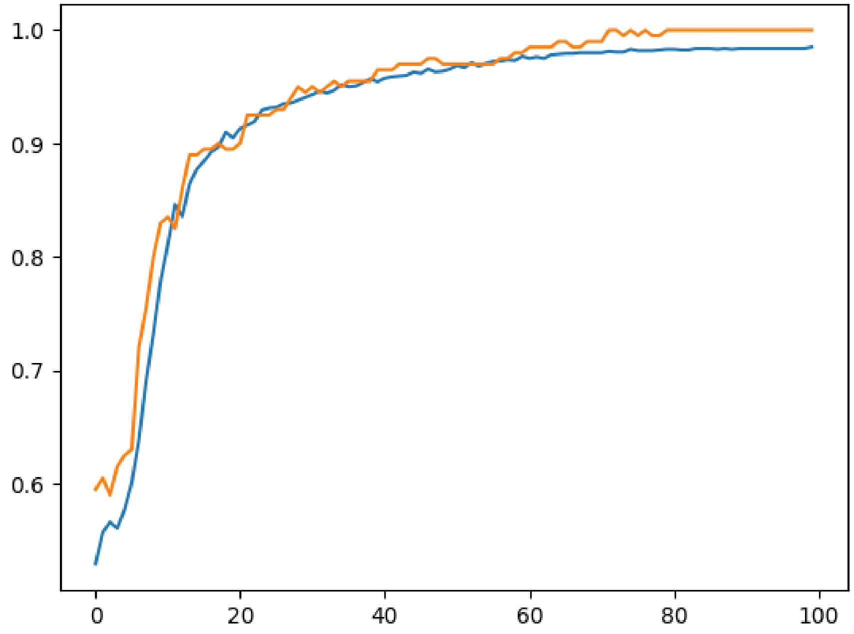
```
print(weights[0])
print(weights[1])    #bias
```

```
[[0.29073623]
 [0.80489254]]
[-4.249588]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```




```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.1487 - accuracy: 0.9850
test loss, test acc: [0.14870315790176392, 0.9850000143051147]
```

Model.predict for test dataset

```
predictions = model.predict(data_points_test)
```

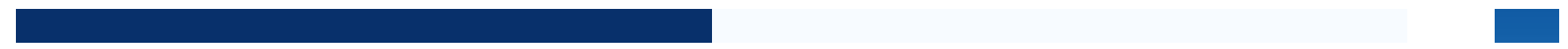
```
7/7 [==============================] - 0s 2ms/step
```

```
predictions
```

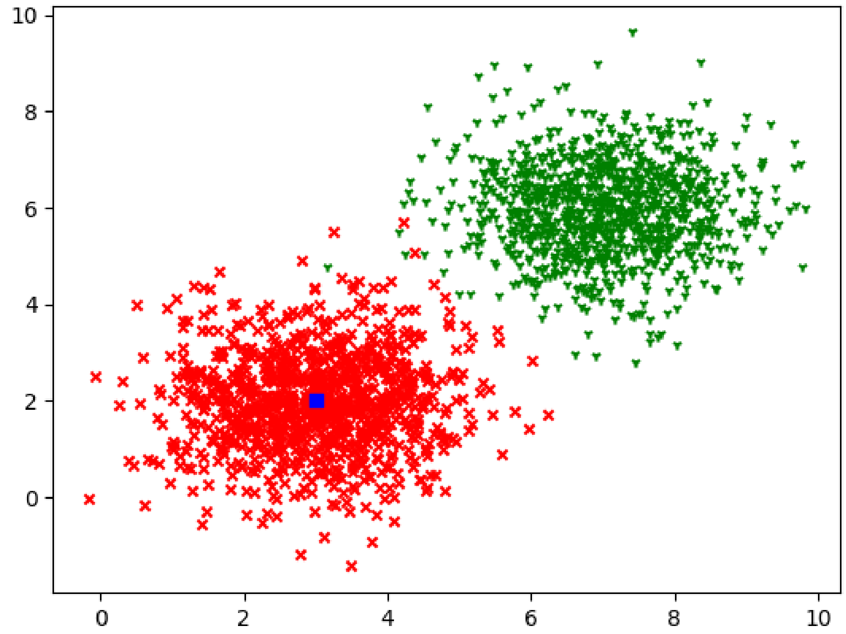
```
[0.9441177 ]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

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



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



```
1/1 [==============================] - 0s 38ms/step
array([[0.14584175]], dtype=float32)
```

współczynnik uczenia 0.01 (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 = "sigmoid"))
```

Definiujemy **optymalizator** i **błąd** (entropia krzyżowa). **Współczynnik uczenia = 0.01**

```
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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (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**:

```
epochs = 100
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val))
```

```
Epoch 100/100
50/50 [==============================] - 0s 4ms/step - loss: 0.0260 - accuracy: 0.9969 - val_loss: 0.0172 - val_accuracy: 1.0000
```

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

```
 0.07759977877140045,
 0.07526867091655731,
 0.07320705056190491,
 0.07114241272211075,
```

```
 0.030123250558972,
 0.029564738273620605,
 0.029459383338689804,
 0.028790391981601715,
 0.028467733412981033,
 0.027938559651374817,
 0.027630411088466644,
 0.027289949357509613,
 0.026920989155769348,
 0.026573894545435905,
 0.026212144643068314,
 0.0260310098528862]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
     0.02475529350534050b4,
     0.02501864917576313,
     0.024216074496507645,
     0.024894513189792633,
     0.023462919518351555,
     0.02345597930252552,
     0.023490060120821,
     0.02237200178205967,
     0.02129662036895752,
     0.021138455718755722,
     0.020573487505316734,
     0.020201964303851128,
     0.0209331177175045,
     0.020232191309332848,
     0.019562937319278717,
     0.01877252198755741,
     0.01895551010966301,
     0.018758609890937805,
     0.018780380487442017,
     0.01810307428240776,
     0.017332740128040314,
     0.0186034943908453,
     0.017229650169610977]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

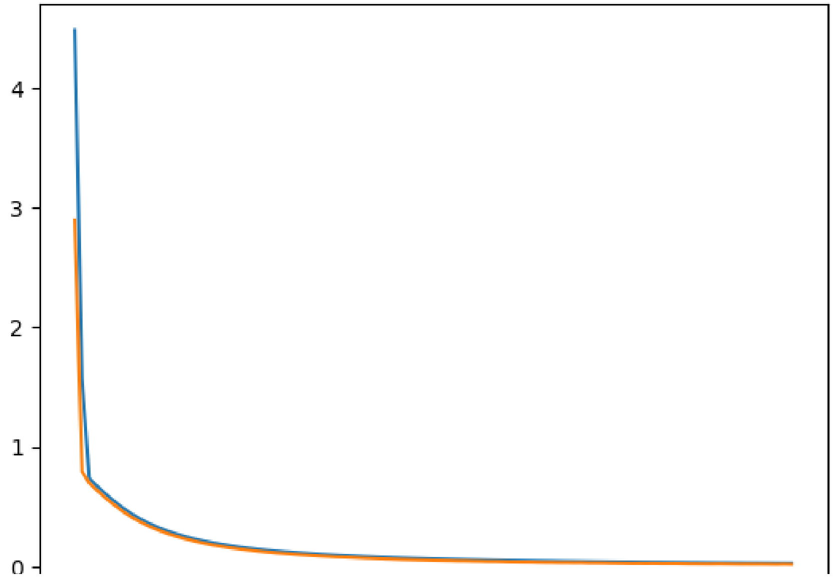
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()
```

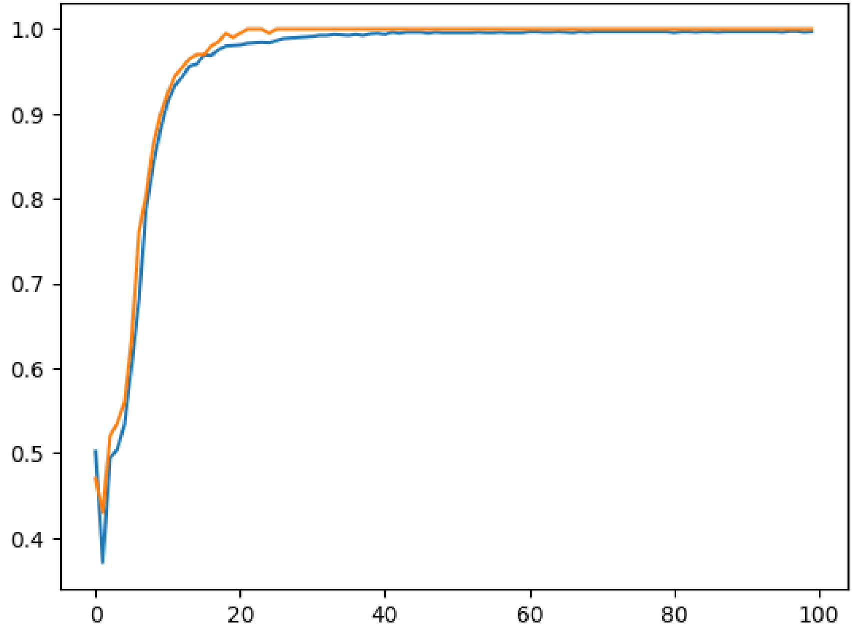
```
print(weights[0])
print(weights[1])    #bias
```

```
[[1.2067593]
 [1.486919 ]]
[-11.953843]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```



```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.0219 - accuracy: 0.9950
test loss, test acc: [0.021869869902729988, 0.9950000047683716]
```

Model.predict for test dataset

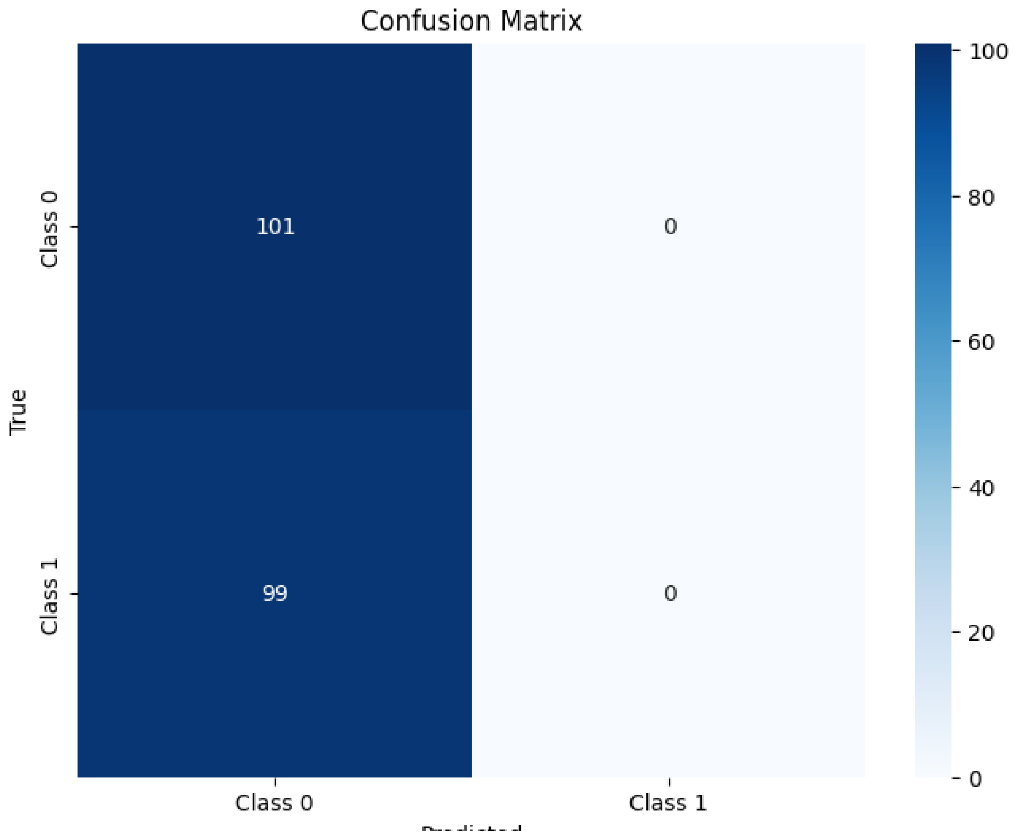
```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 3ms/step
```

```
predictions
```

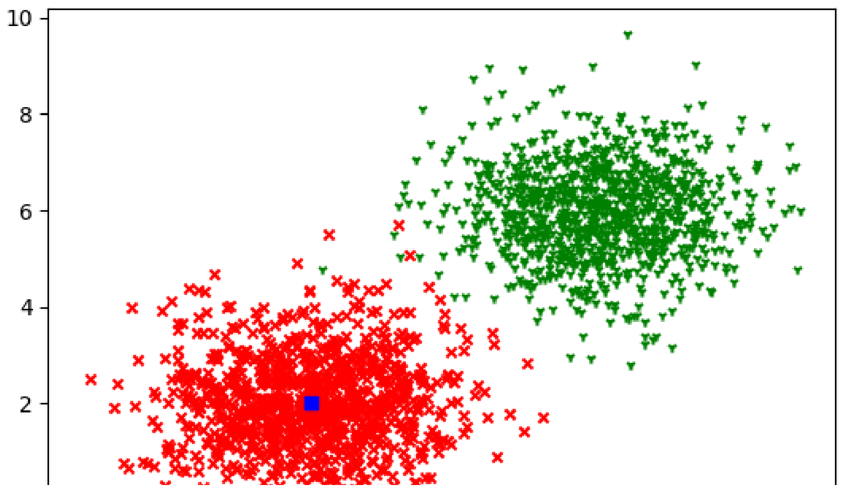
```
       [8.63527507e-03],
       [9.95209336e-01],
       [9.97375667e-01],
       [9.98624265e-01],
       [9.96127605e-01],
       [9.61828053e-01],
       [9.99493718e-01],
       [1.43895103e-02],
       [2.03348417e-02],
       [9.99848127e-01],
       [9.88402367e-01],
       [9.98274386e-01],
       [6.51492970e-04],
       [5.71148528e-04],
       [1.13827933e-03],
       [9.44426775e-01],
       [9.98534620e-01],
       [1.47829922e-02],
       [9.99673843e-01],
       [9.93921578e-01],
       [4.79757451e-02],
       [3.05602234e-03],
       [1.04067882e-03],
       [9.86231804e-01],
       [9.99453187e-01],
       [1.24018295e-02],
       [9.96077299e-01]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



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

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



Batch 100

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** (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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
 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**:

```
epochs = 100
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val),batch_size=100)
```

```
Epoch 87/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0760 - accuracy: 0.9950 - val_loss: 0.0633 - val_accuracy: 1.0000
Epoch 88/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0754 - accuracy: 0.9950 - val_loss: 0.0623 - val_accuracy: 1.0000
Epoch 89/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0748 - accuracy: 0.9950 - val_loss: 0.0620 - val_accuracy: 1.0000
Epoch 90/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0742 - accuracy: 0.9956 - val_loss: 0.0613 - val_accuracy: 1.0000
Epoch 91/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0736 - accuracy: 0.9950 - val_loss: 0.0610 - val_accuracy: 1.0000
Epoch 92/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0732 - accuracy: 0.9944 - val_loss: 0.0604 - val_accuracy: 1.0000
Epoch 93/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0726 - accuracy: 0.9950 - val_loss: 0.0594 - val_accuracy: 1.0000
Epoch 94/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0721 - accuracy: 0.9950 - val_loss: 0.0594 - val_accuracy: 1.0000
Epoch 95/100
16/16 [==============================] - 0s 3ms/step - loss: 0.0715 - accuracy: 0.9962 - val_loss: 0.0586 - val_accuracy: 1.0000
Epoch 96/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0711 - accuracy: 0.9950 - val_loss: 0.0584 - val_accuracy: 1.0000
Epoch 97/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0706 - accuracy: 0.9956 - val_loss: 0.0576 - val_accuracy: 1.0000
Epoch 98/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0700 - accuracy: 0.9950 - val_loss: 0.0571 - val_accuracy: 1.0000
Epoch 99/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0695 - accuracy: 0.9950 - val_loss: 0.0564 - val_accuracy: 1.0000
Epoch 100/100
16/16 [==============================] - 0s 5ms/step - loss: 0.0690 - accuracy: 0.9950 - val_loss: 0.0570 - val_accuracy: 1.0000
```

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

```
      0.09510884433984756,
      0.09408058226108551,
      0.09310483187437057,
      0.09211430698633194,
      0.09115633368492126,
      0.09002282470464706,
      0.08907909691333771,
      0.08858247101306915,
      0.08753516525030136,
      0.08665456622838974,
      0.08573874831199646,
      0.08493298292160034,
      0.08425812423229218,
      0.08345074951648712,
      0.08265413343906403,
      0.08179871737957001,
      0.0812048614025116,
      0.08040919154882431,
      0.07988685369491577,
      0.07912836968898773,
      0.07840942591428757,
      0.07797347009181976,
      0.07725008577108383,
      0.07659262418746948,
      0.07597753405570984,
      0.07535538077354431,
      0.07482575625181198,
      0.0742175355553627,
      0.07356461882591248,
      0.07318326830863953,
      0.0726126879453659,
      0.07207027822732925,
      0.07150795310735703,
      0.07108244299888611,
      0.07055582851171494,
      0.06995594501495361,
      0.06950551271438599,
      0.06898888200521469]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
        0.09499341994524002,
        0.09536468237638474,
        0.09263116866350174,
        0.09086103737354279,
        0.0894506424665451,
        0.08903728425502777,
        0.08703113347291946,
        0.08522022515535355,
        0.08466675877571106,
        0.08370838314294815,
        0.08227367699146271,
        0.0817686915397644,
        0.0808502584695816,
        0.079201839864254,
        0.07909177243709564,
        0.07757923752069473,
        0.0776924416422844,
        0.07530153542757034,
        0.07512091845273972,
        0.07532822340726852,
        0.07276222109794617,
        0.07358121126890182,
        0.07259852439165115,
        0.07114215940237045,
        0.06988360732793808,
        0.06952285766601562,
        0.0698176845908165,
        0.0680810958147049,
        0.06684692203998566,
        0.06791244447231293,
        0.0661996379494667,
        0.06686114519834518,
        0.06501101702451706,
        0.06409647315740585,
        0.06362605094909668,
        0.0632840171456337,
        0.062281373888254166,
        0.06197082996368408,
        0.061320897191762924,
        0.060978807508945465,
        0.06039450690150261,
        0.05938421189785004,
        0.05939878895878792,
        0.05858723819255829,
        0.058394089341163635,
        0.05764712020754814,
        0.05706232041120529,
        0.05644652247428894,
        0.05703643709421158]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

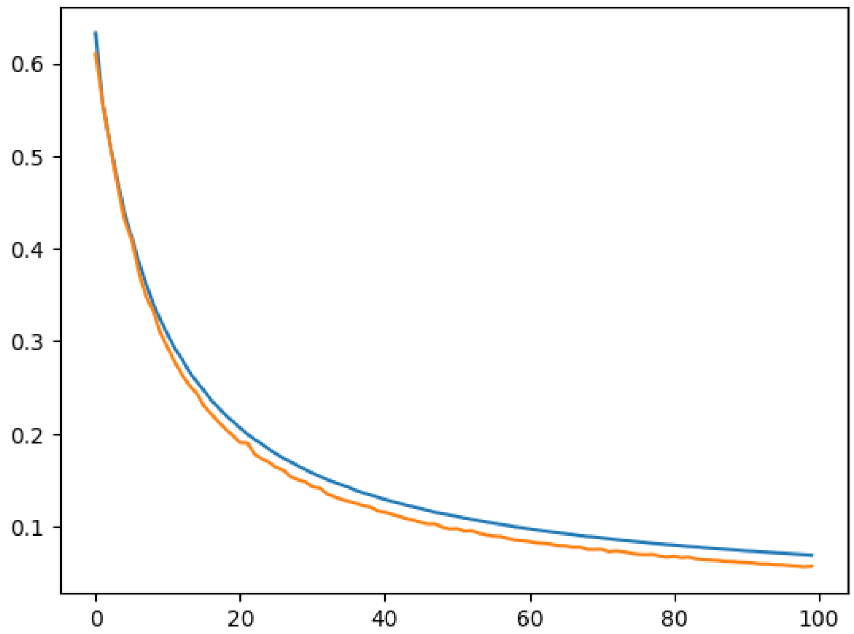
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()
```

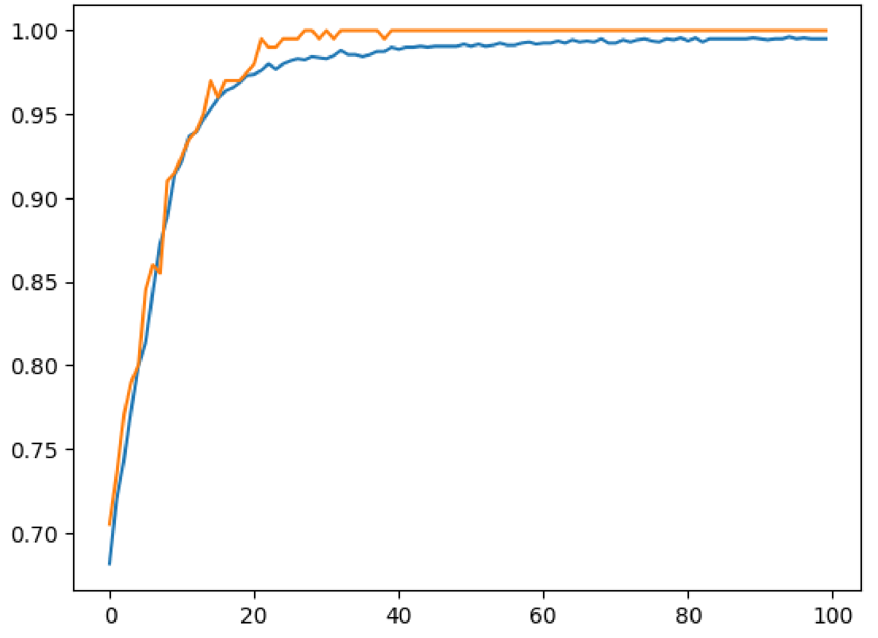
```
print(weights[0])
print(weights[1])    #bias
```

```
[[0.6415532]
 [1.0350044]]
[-7.0897894]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```



```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.0651 - accuracy: 0.9900
test loss, test acc: [0.06507639586925507, 0.9900000095367432]
```

Model.predict for test dataset

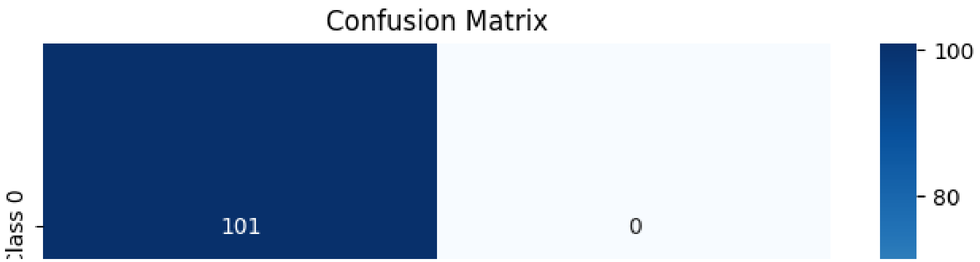
```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 2ms/step
```

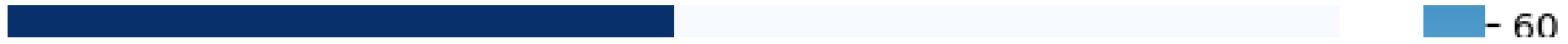
```
predictions
```

```
             [0.15666138],
             [0.03285969],
             [0.02169874],
             [0.9429975 ],
             [0.9920685 ],
             [0.06081996],
             [0.9775991 ]], dtype=float32)
```

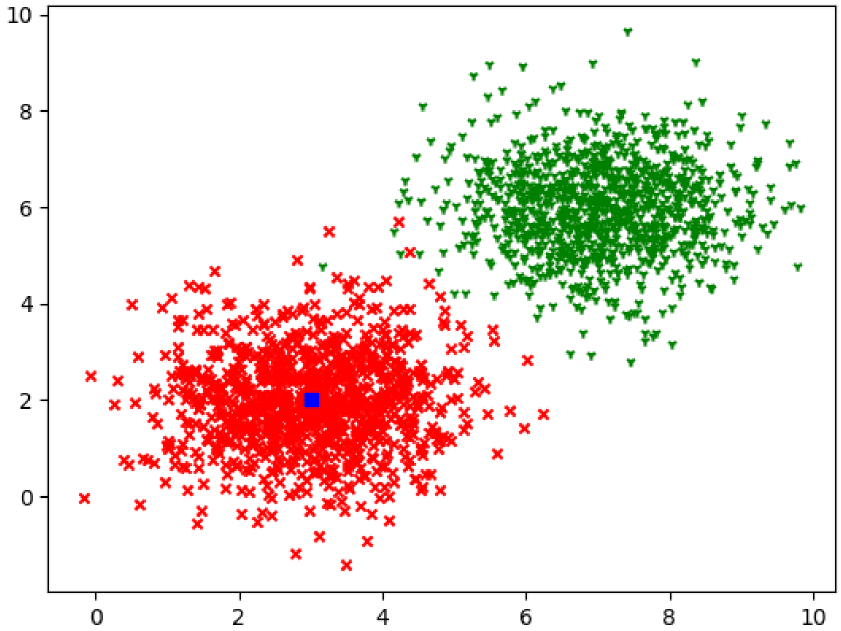
```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



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




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




```
1/1 [==============================] - 0s 46ms/step
array([[0.04330896]], dtype=float32)
```

Batch 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 = "sigmoid"))
```

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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_23 (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**:

```
epochs = 100
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val),batch_size=200)
```

```
Epoch 96/100
8/8 [==============================] - 0s 5ms/step - loss: 0.1123 - accuracy: 0.9912 - val_loss: 0.0989 - val_accuracy: 1.0000
Epoch 97/100
8/8 [==============================] - 0s 6ms/step - loss: 0.1115 - accuracy: 0.9919 - val_loss: 0.0978 - val_accuracy: 1.0000
Epoch 98/100
8/8 [==============================] - 0s 6ms/step - loss: 0.1108 - accuracy: 0.9912 - val_loss: 0.0973 - val_accuracy: 1.0000
Epoch 99/100
8/8 [==============================] - 0s 6ms/step - loss: 0.1101 - accuracy: 0.9919 - val_loss: 0.0968 - val_accuracy: 1.0000
Epoch 100/100
8/8 [==============================] - 0s 7ms/step - loss: 0.1092 - accuracy: 0.9919 - val_loss: 0.0963 - val_accuracy: 1.0000
```

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

```
 0.12641550600528717,
 0.12507416307926178,
 0.12415044754743576,
 0.12323710322380066,
 0.12230238318443298,
 0.12110915780067444,
 0.12030860036611557,
 0.11939743906259537,
 0.11830034106969833,
 0.11747345328330994,
 0.11655350029468536,
 0.1157480999827385,
 0.11489810794591904,
 0.11403873562812805,
 0.1131690964102745,
 0.11233840137720108,
 0.11154348403215408,
 0.11083657294511795,
 0.11005190014839172,
 0.10921715945005417]
```

```
val_loss = h.history['val_loss']
val_loss
```

```
[0.5361842513084412,
 0.5096993446350098,
 0.4805546700954437,
 0.4590274691581726,
 0.44056305289268494,
 0.42570868134498596,
 0.4096633791923523,
 0.39676952362060547,
 0.3806786835193634,
 0.3689177632331848,
 0.3574894666671753,
 0.3504687249660492,
 0.337431401014328,
 0.32764771580696106,
 0.3194611370563507,
 0.3120904266834259,
 0.3028360605239868,
 0.2947499454021454,
 0.28662747144699097,
 0.2796083688735962,
 0.27255669236183167,
 0.26895958185195923,
 0.26173824071884155,
 0.2569095194339752,
 0.2499256581068039,
 0.24331413209438324,
 0.23912519216537476,
```

```
      0.23500610888004303,
      0.22933639585971832,
      0.22476685047149658,
      0.2205430567264557,
      0.2152010202407837,
      0.21178649365901947,
      0.20708492398262024,
      0.20382417738437653,
      0.2012517899274826,
      0.19656501710414886,
      0.19365626573562622,
      0.1908787190914154,
      0.18668793141841888,
      0.18307793140411377,
      0.18097586929798126,
      0.1778327226638794,
      0.1757078915834427,
      0.1733546406030655,
      0.16982829570770264,
      0.16681087017059326,
      0.16493423283100128,
      0.1619570553302765,
      0.1600339561700821,
      0.1581609547138214,
      0.15578553080558777,
      0.15398269891738892,
      0.15166276693344116,
      0.1499858796596527,
      0.1485176384449005,
      0.14606696367263794,
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

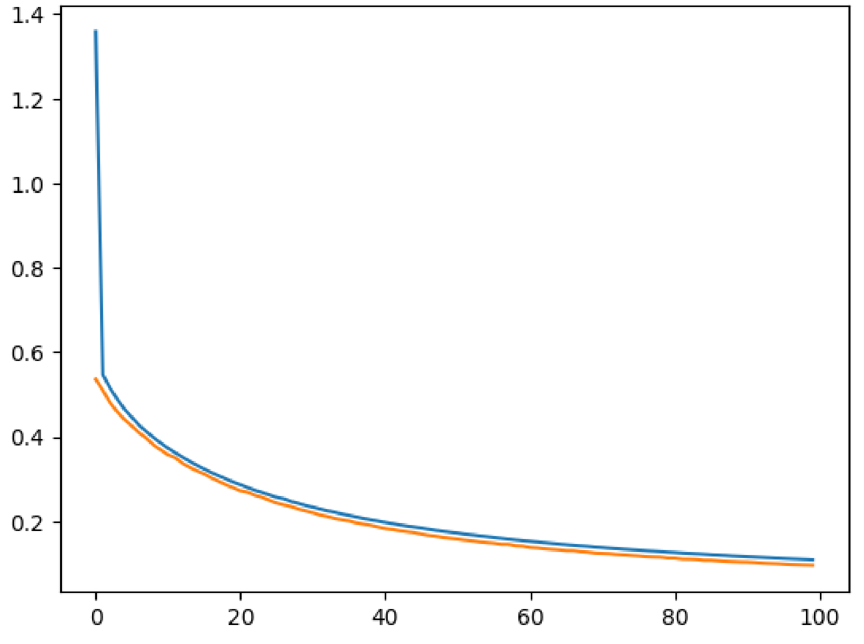
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()
```

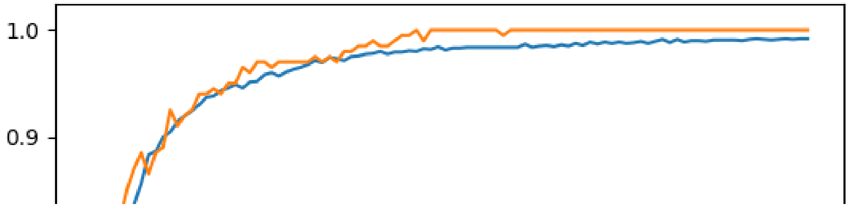
```
print(weights[0])
print(weights[1])    #bias
```

```
[[0.43002713]
 [0.8891428 ]]
[-5.3686643]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```




```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.1058 - accuracy: 0.9850
test loss, test acc: [0.1057654321193695, 0.9850000143051147]
```

Model.predict for test dataset

```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 2ms/step
```

```
predictions
```

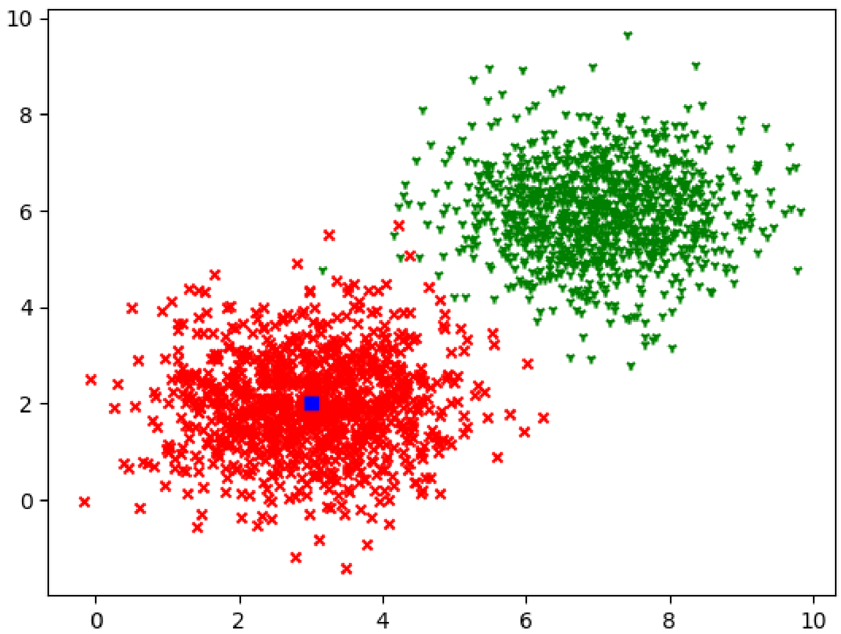
```
       [0.03887419],
       [0.95320237],
       [0.9420866 ],
       [0.9877457 ],
       [0.04222259],
       [0.96528405],
       [0.05916271],
       [0.91661304],
       [0.9772602 ],
       [0.1299509 ],
       [0.9644208 ],
       [0.9456073 ],
       [0.96236557],
       [0.96671623],
       [0.89355946],
       [0.98333085],
       [0.18453756],
       [0.28537428],
       [0.9884014 ],
       [0.89039725],
       [0.9660665 ],
       [0.04596994],
       [0.03643499],
       [0.03568943],
       [0.8362182 ],
       [0.97641605],
       [0.15189932],
       [0.9839335 ],
       [0.9540657 ],
       [0.2255915 ],
       [0.0730867 ],
       [0.06248192],
       [0.9071301 ],
       [0.9796771 ],
       [0.10051491],
       [0.95956624]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

Confusion Matrix

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

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




```
1/1 [==============================] - 0s 36ms/step
array([[0.09109835]], dtype=float32)
```

Batch 400

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** (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,metrics=['accuracy'])
```

Informacja o modelu:

```
model.summary()
```

```
Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_24 (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**:

```
epochs = 100
h = model.fit(data_points_train,label_train, verbose=1 ,epochs=epochs,validation_data=(data_points_val,label_val),batch_size=400)
```

```
4/4 [==============================] - 0s 11ms/step - loss: 0.2197 - accuracy: 0.9737 - val_loss: 0.2068 - val_accuracy: 0.9750
Epoch 76/100
4/4 [==============================] - 0s 13ms/step - loss: 0.2179 - accuracy: 0.9719 - val_loss: 0.2049 - val_accuracy: 0.9750
Epoch 77/100
4/4 [==============================] - 0s 12ms/step - loss: 0.2161 - accuracy: 0.9712 - val_loss: 0.2035 - val_accuracy: 0.9700
Epoch 78/100
4/4 [==============================] - 0s 12ms/step - loss: 0.2143 - accuracy: 0.9725 - val_loss: 0.2024 - val_accuracy: 0.9800
Epoch 79/100
4/4 [==============================] - 0s 11ms/step - loss: 0.2127 - accuracy: 0.9750 - val_loss: 0.1999 - val_accuracy: 0.9750
Epoch 80/100
4/4 [==============================] - 0s 11ms/step - loss: 0.2107 - accuracy: 0.9744 - val_loss: 0.1977 - val_accuracy: 0.9750
Epoch 81/100
4/4 [==============================] - 0s 11ms/step - loss: 0.2091 - accuracy: 0.9737 - val_loss: 0.1967 - val_accuracy: 0.9800
Epoch 82/100
4/4 [==============================] - 0s 11ms/step - loss: 0.2074 - accuracy: 0.9750 - val_loss: 0.1953 - val_accuracy: 0.9850
Epoch 83/100
4/4 [==============================] - 0s 13ms/step - loss: 0.2059 - accuracy: 0.9787 - val_loss: 0.1929 - val_accuracy: 0.9800
Epoch 84/100
4/4 [==============================] - 0s 12ms/step - loss: 0.2040 - accuracy: 0.9769 - val_loss: 0.1912 - val_accuracy: 0.9800
Epoch 85/100
4/4 [==============================] - 0s 20ms/step - loss: 0.2025 - accuracy: 0.9769 - val_loss: 0.1898 - val_accuracy: 0.9850
Epoch 86/100
4/4 [==============================] - 0s 12ms/step - loss: 0.2010 - accuracy: 0.9769 - val_loss: 0.1882 - val_accuracy: 0.9850
Epoch 87/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1996 - accuracy: 0.9781 - val_loss: 0.1861 - val_accuracy: 0.9900
Epoch 88/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1979 - accuracy: 0.9775 - val_loss: 0.1855 - val_accuracy: 0.9850
Epoch 89/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1964 - accuracy: 0.9781 - val_loss: 0.1837 - val_accuracy: 0.9850
Epoch 90/100
4/4 [==============================] - 0s 13ms/step - loss: 0.1950 - accuracy: 0.9787 - val_loss: 0.1820 - val_accuracy: 0.9900
Epoch 91/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1935 - accuracy: 0.9800 - val_loss: 0.1807 - val_accuracy: 0.9850
Epoch 92/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1921 - accuracy: 0.9794 - val_loss: 0.1794 - val_accuracy: 0.9900
Epoch 93/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1907 - accuracy: 0.9812 - val_loss: 0.1776 - val_accuracy: 0.9900
Epoch 94/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1894 - accuracy: 0.9806 - val_loss: 0.1759 - val_accuracy: 0.9900
Epoch 95/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1881 - accuracy: 0.9794 - val_loss: 0.1753 - val_accuracy: 0.9950
Epoch 96/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1867 - accuracy: 0.9819 - val_loss: 0.1735 - val_accuracy: 1.0000
Epoch 97/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1855 - accuracy: 0.9800 - val_loss: 0.1726 - val_accuracy: 0.9950
Epoch 98/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1840 - accuracy: 0.9825 - val_loss: 0.1710 - val_accuracy: 1.0000
Epoch 99/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1830 - accuracy: 0.9812 - val_loss: 0.1695 - val_accuracy: 1.0000
Epoch 100/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1816 - accuracy: 0.9819 - val_loss: 0.1683 - val_accuracy: 1.0000
```

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

```
 0.3047181963920593,
 0.3012007176876068,
 0.2972562313079834,
 0.293799489736557,
 0.29037564992904663,
 0.2872477173805237,
```

```
 0.19352253409881592,
 0.19207939505577087,
 0.19073787331581116,
 0.18940868973731995,
 0.18814758956432343,
 0.18665599822998047,
 0.1854790449142456,
 0.18401916325092316,
 0.18295152485370636,
 0.18155828118324281
```

```
val_loss = h.history['val_loss']
val_loss
```

```
     0.1966799795627594,
     0.1953277587890625,
     0.19288483262062073,
     0.19122423231601715,
     0.1898297518491745,
     0.1881985068321228,
     0.18610839545726776,
     0.18549294769763947,
     0.18374523520469666,
     0.18197081983089447,
     0.18066704273223877,
     0.1793949455022812,
     0.17758993804454803,
     0.17585183680057526,
     0.17526482045650482,
     0.17347538471221924,
     0.17261318862438202,
     0.17100811004638672,
     0.1694786101579666,
     0.16832588613033295]
```

```
val_accuracy = h.history['val_accuracy']
accuracy = h.history['accuracy']
```

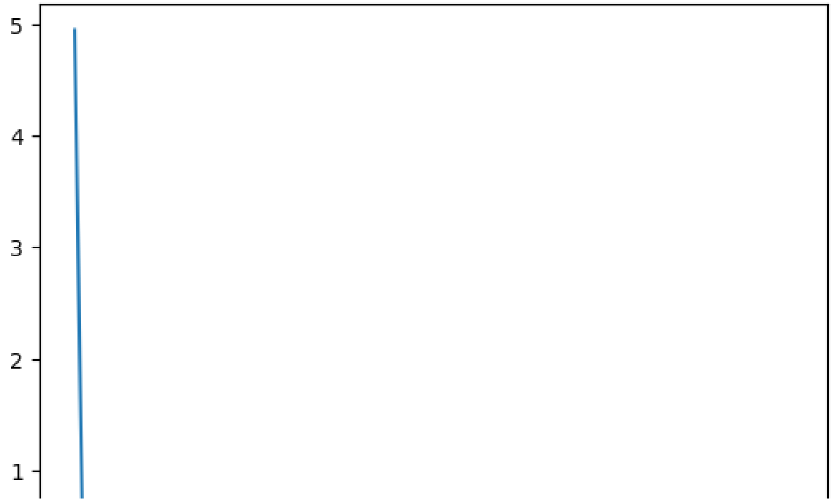
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()
```

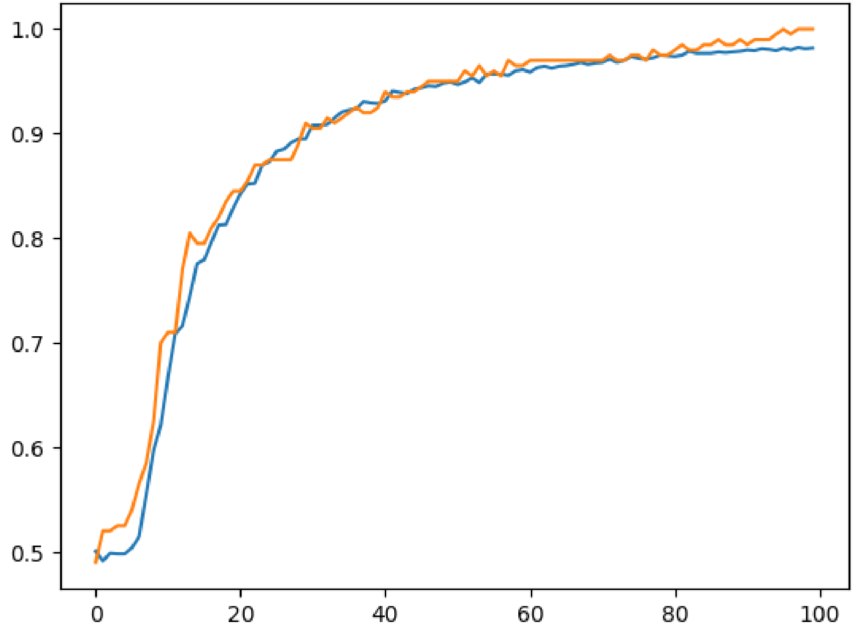
```
print(weights[0])
print(weights[1])    #bias
```

```
[[0.21406241]
 [0.7659361 ]]
[-3.657632]
```

```
plt.plot(Loss)
plt.plot(val_loss)

plt.show()
```




```
plt.plot(accuracy)
plt.plot(val_accuracy)
plt.show()
```



Model.evaluate for test data

```
results = model.evaluate(data_points_test,label_test)
print("test loss, test acc:", results)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.1793 - accuracy: 0.9850
test loss, test acc: [0.17931680381298065, 0.9850000143051147]
```

Model.predict for test dataset

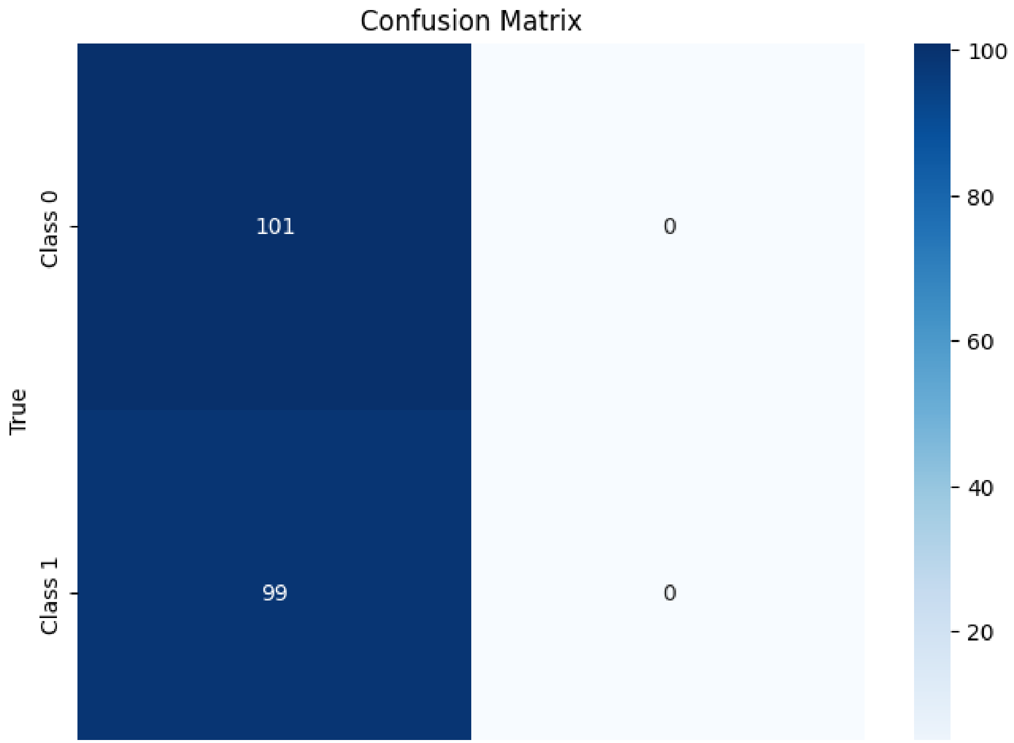
```
predictions = model.predict(data_points_test)
```

```
7/7 [==============================] - 0s 2ms/step
```

```
predictions
```

```
       [0.9191581 ],
       [0.9498681 ],
       [0.8756479 ],
       [0.96368283],
       [0.32990745],
       [0.4930959 ],
       [0.96790355],
       [0.8198292 ],
       [0.9342449 ],
       [0.13312134],
       [0.10245708],
       [0.08154522],
       [0.7994891 ],
       [0.9579017 ],
       [0.26032454],
       [0.9604467 ],
       [0.93202925],
       [0.32018277],
       [0.15687034],
       [0.17212923],
       [0.8615817 ],
       [0.9535125 ],
       [0.161577  ],
       [0.93483543]], dtype=float32)
```

```
y_true = np.array(label_test, dtype=int)
y_pred = np.array(predictions, dtype=int)

# Convert continuous predictions to class labels (binary classification example)
y_pred = (y_pred > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(label_test, y_pred)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



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

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


10