Tikslas – apmokyti vieną neuroną spręsti nesudėtingą dviejų klasių uždavinį, atlikti tyrimą.

Duomenys:
Duomenys imti iš https://archive.ics.uci.edu/ml/datasets/iris
Čia kiekvienas irisas turi priskirtą klasę: Setosa, Versicolor ir Virginica

Iš atsisiųstų duomenų reikia pasidaryti 2 duomenų rinkinius:

(1) Vieną klasę sudaro Setosa rūšis (50 duomenų įrašų), kitą klasę – Versicolor ir Virginica rūšys (100 duomenų įrašų).

(2) Vieną klasę sudaro Versicolor rūšis (50 duomenų įrašų), kitą klasę – Virginica rūšys (50 duomenų įrašų).

Sutvarkyti duomenys atrodo taip:

sepl – sepal length – taurėlapio ilgis
sepw – sepal width – taurėlapio plotis
petl – petal length – žiedlapio ilgis
petw – petal width – žiedlapio plotis
class – priskirta klasė 0/1 (abiems duomenims skirtingai priskiriama)
b – bias (visalaik = 1)

Pirmi:

```
one.head()
```

	sepl	sepw	petl	petw	class	b
36	5.5	3.5	1.3	0.2	0	1
47	4.6	3.2	1.4	0.2	0	1
28	5.2	3.4	1.4	0.2	0	1
9	4.9	3.1	1.5	0.1	0	1
13	4.3	3.0	1.1	0.1	0	1

```
one.tail()
```

	sepl	sepw	petl	petw	class	b
93	5.0	2.3	3.3	1.0	1	1
72	6.3	2.5	4.9	1.5	1	1
122	7.7	2.8	6.7	2.0	1	1
65	6.7	3.1	4.4	1.4	1	1
90	5.5	2.6	4.4	1.2	1	1

Antri:

```
two.head()
```

	sepl	sepw	petl	petw	class	b
86	6.7	3.1	4.7	1.5	0	1
97	6.2	2.9	4.3	1.3	0	1
78	6.0	2.9	4.5	1.5	0	1
59	5.2	2.7	3.9	1.4	0	1
63	6.1	2.9	4.7	1.4	0	1

```
two.tail()
```

	sepl	sepw	petl	petw	class	b
122	7.7	2.8	6.7	2.0	1	1
108	6.7	2.5	5.8	1.8	1	1
145	6.7	3.0	5.2	2.3	1	1
115	6.4	3.2	5.3	2.3	1	1
140	6.7	3.1	5.6	2.4	1	1

Kodas

Duomenis reikejo išskaidyt į mokymo ir testavimo aibes, pasirinkau 80/20 santykį.

Duomenų paruošimas

```
data = pd.read_csv('iris.data', sep=",", header=None)
data.columns = ["sepl", "sepw", "petl", "petw","class"]
```

```
one = pd.concat(
    [
        data[data["class"].isin(["Iris-setosa"])].sample(n = 50, random_state=2),
        data[data["class"].isin(["Iris-versicolor", "Iris-virginica"])].sample(n = 100, random_state=2)
    ]
    )

one["class"].replace({"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 1}, inplace=True)
one["b"] = 1

onex_train, onex_test, oney_train, oney_test = train_test_split(
    np.array(one.loc[:, one.columns != "class"]),
    np.array(one.loc[:, one.columns == "class"]),
    test_size = 0.2)
```

```
two = pd.concat(
    [
        data[data["class"].isin(["Iris-versicolor"])].sample(n = 50, random_state=2),
        data[data["class"].isin(["Iris-virginica"])].sample(n = 50, random_state=2)
    ]
    )

two["class"].replace({"Iris-versicolor": 0, "Iris-virginica": 1}, inplace=True)
two["b"] = 1

twox_train, twox_test, twoy_train, twoy_test = train_test_split(
    np.array(two.loc[:, two.columns != "class"]),
    np.array(two.loc[:, two.columns == "class"]),
    test_size = 0.2)
```

Svorius koregavau pagal formulę

$$w_k(t+1) = w_k(t) + \eta(t_i - y_i)x_{ik}$$

Metodai

```
def new_weights(weights, row, prediction, value,lrate):
    # Svorių koregavimas
    new_weights = []
    for i in range(len(weights)):
        new_weights.append(weights[i] + lrate*(value - prediction)*row[i])
    return new_weights
```

Epocha – kai algoritmas visu duomenis pamato 1 kartą

Iteracija – kai algoritmas pamato vieną duomenų eilutę (batch size = 1)

Visus pradinius svorius parinkau 0,5 dėl paprastumo.

Slenkstinės funckijos metodai

```
def binary(row,weights):
    # Slenkstine funkcija
    a = np.dot(row, weights)
    if a>=0:
        return 1
    else:
        return 0

def train_binary(lrate, epochs, xtrain, ytrain):
    # Treniravimo funkcija
    weights = [0.5, 0.5, 0.5, 0.5, 0.5]
    errors = [] #error calculation

    for e in range(epochs):
        errorssum = 0 #error calculation
        for i in range(len(xtrain)):
            y = binary(xtrain[i], weights)
            row = xtrain[i]
            result = ytrain[i]
            weights = new_weights(weights, row, y, result , lrate)

            errorssum += (result - y)**2 #error calculation

        errors.append([e,(0.5*errorssum)[0]]) #error calculation

    plot_error(pd.DataFrame(errors, columns = ["epoch", "error"]), "Binary error graph")
    return (weights, errors[-1][1])

def test_binary(weights, xtest, ytest):
    # Matuoja slenkstinio modelio taikluma
    success = 0
    for i in range(len(xtest)):
        if binary(xtest[i],weights) == ytest[i][0]:
            success +=1
    return success/len(xtest)
```

Sigmoidinės funkcijos metodai

```
def sigmoid(row,weights):
    # Sigmoidine funkcija
    a = np.dot(row, weights)
    return 1/(1+math.exp(-a))

def train_sigmoid(lrate, epochs, xtrain, ytrain):
    # Treniravimo funkcija
    weights = [0.5, 0.5, 0.5, 0.5, 0.5]
    errors = [] #error calculation

    for e in range(epochs):
        errorssum = 0 #error calculation
        for i in range(len(xtrain)):
            y = sigmoid(xtrain[i], weights)
            row = xtrain[i]
            result = ytrain[i]
            weights = new_weights(weights, row, y, result , lrate)

            errorssum += (result - y)**2 #error calculation

        errors.append([e,(0.5*errorssum)[0]]) #error calculation

    plot_error(pd.DataFrame(errors, columns = ["epoch", "error"]), "Sigmoid error graph")
    return (weights, errors[-1][1])

def test_sigmoid(weights, xtest, ytest):
    # Matuoja sigmoidinio modelio taikluma
    success = 0
    for i in range(len(xtest)):
        if round(sigmoid(xtest[i],weights) ,0) == ytest[i][0]:
            success +=1
    return success/len(xtest)
```

Papildoma funkcija paklaidos grafiko braižymui

```
def plot_error(data, title):
    x = data.epoch
    y = data.error
    plt.plot(x,y, alpha = 0.7)
    plt.xlabel('Epochs')
    plt.ylabel('Error')
    plt.title(title)
```

Rezultatai

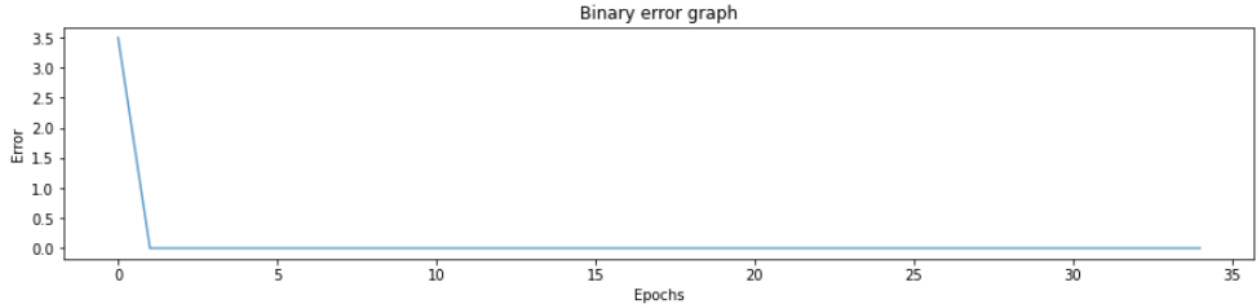
Mokymosi greitis = 1

Epochų kiekis = 35

Pirmi duomenys, slenkstine funkcija

```
onebinary = train_binary(1, 35, onex_train, oney_train)
print("Final weights:", [round(i[0],4) for i in onebinary[0]], " ",
      "Final error:", onebinary[1],
      "Accuracy:", test_binary(onebinary[0], onex_test, oney_test))
```

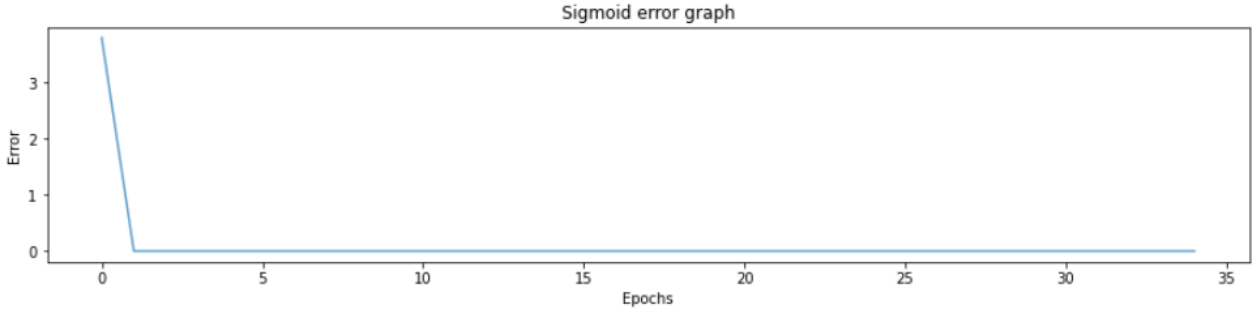
```
Final weights: [-1.8, -4.7, 7.5, 3.7, -0.5]   Final error: 0.0 Accuracy: 1.0
```



Pirmi duomenys, sigmoidine funkcija

```
onesigmoid = train_sigmoid(1, 35, onex_train, oney_train)
print("Final weights:", [round(i[0],4) for i in onesigmoid[0]], " ",
      "Final error:", onesigmoid[1],
      "Accuracy:", test_sigmoid(onesigmoid[0], onex_test, oney_test))
```

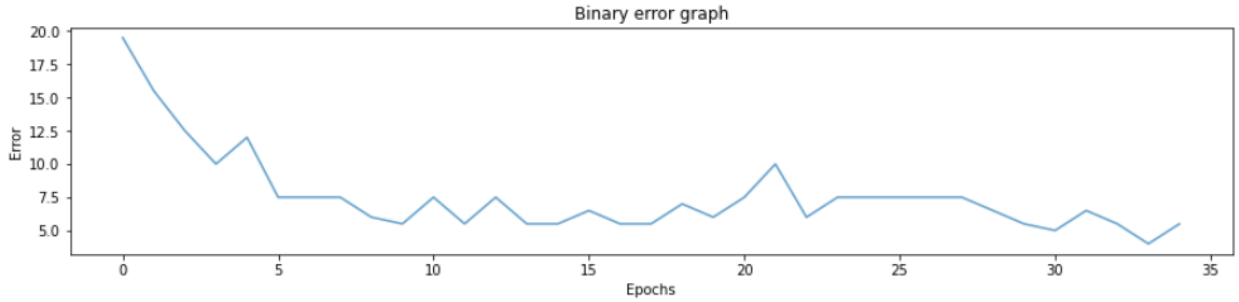
```
Final weights: [-1.8907, -5.1006, 8.8883, 4.5235, -0.6061]   Final error: 8.991662158915849e-08 Accuracy: 1.0
```



Antri duomenys, slenkstine funkcija

```
twobinary = train_binary(1, 35, twox_train, twoy_train)
print("Final weights:", [round(i[0],4) for i in twobinary[0]], " ",
      "Final error:", twobinary[1],
      "Accuracy:", test_binary(twobinary[0], twox_test, twoy_test))
```

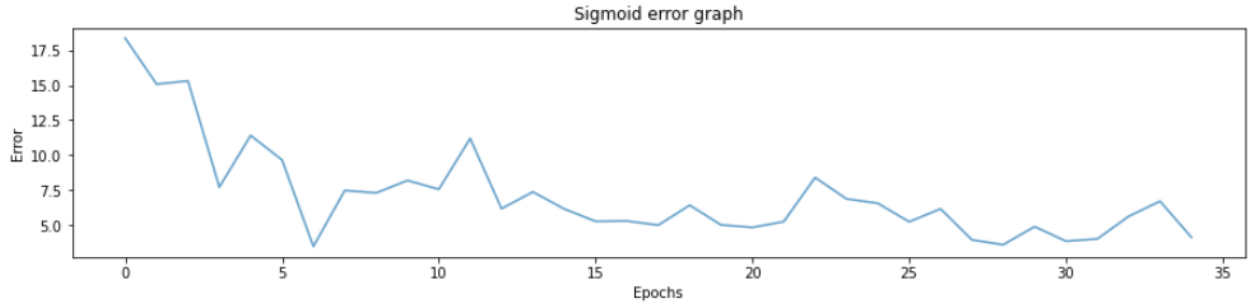
```
Final weights: [-36.0, -50.8, 50.5, 78.9, -37.5]   Final error: 5.5 Accuracy: 0.9
```



Antri duomenys, sigmoidine funkcija

```
twosigmoid = train_sigmoid(1, 35, twox_train, twoy_train)
print("Final weights:", [round(i[0],4) for i in twosigmoid[0]], " ",
      "Final error:", twosigmoid[1],
      "Accuracy:", test_sigmoid(twosigmoid[0], twox_test, twoy_test))
```

```
Final weights: [-29.9236, -51.6058, 41.899, 79.3348, -32.7426]   Final error: 4.128428867098631 Accuracy: 0.9
```



Tyrimas

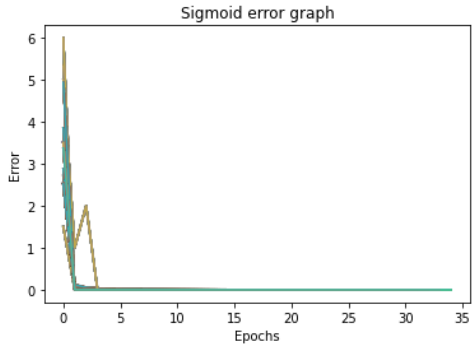
Tyriau slenkstinę ir sigmoidinę funkciją keisdamas mokymosi greičio ir epochų kiekio parametrus:
Mokymosi greitis nuo 0.1 iki 1 su žingsniu 0.1
Epochų skaičius nuo 1 iki 35 su žingsniu 1

1 duomenų tyrimas

```
onetest = []

for lrate in np.arange(0.1, 1.1,0.1):
    for epoch in np.arange(1,36,1):
        model1 = train_binary(lrate, epoch, onex_train, oney_train)
        accuracy1 = test_binary(model1[0], onex_test, oney_test)
        onetest.append(["binary",lrate, epoch, accuracy1])

        model2 = train_sigmoid(lrate, epoch, onex_train, oney_train)
        accuracy2 = test_sigmoid(model2[0], onex_test, oney_test)
        onetest.append(["sigmoid",lrate, epoch, accuracy2])
```



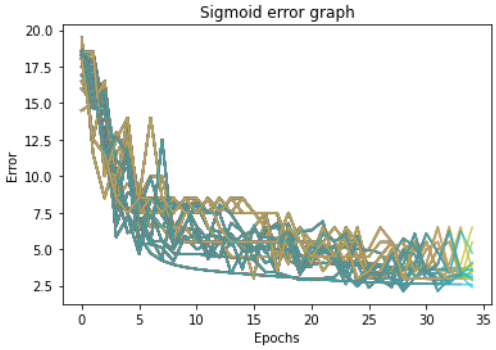

```
pd.DataFrame(onetest, columns = ["function","learning_rate","epochs","accuracy"]).to_csv("testone.csv", index = False)
```

2 duomenų tyrimas

```
twotest = []

for lrate in np.arange(0.1, 1.1,0.1):
    for epoch in np.arange(1,36,1):
        model1 = train_binary(lrate, epoch, twox_train, twoy_train)
        accuracy1 = test_binary(model1[0], twox_test, twoy_test)
        twotest.append(["binary",lrate, epoch, accuracy1])

        model2 = train_sigmoid(lrate, epoch, twox_train, twoy_train)
        accuracy2 = test_sigmoid(model2[0], twox_test, twoy_test)
        twotest.append(["sigmoid",lrate, epoch, accuracy2])
```




```
pd.DataFrame(twotest, columns = ["function","learning_rate","epochs","accuracy"]).to_csv("testtwo.csv", index = False)
```

Kad galėčiau tyrinėti gautus duomenis juos exportavau į CSV formatą ir naudojant Tableau susidariau lenteles.
Pirmi duomenys:

	function / learning rate																			
	binary										sigmoid									
epochs	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
2	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
3	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
4	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
5	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
6	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
7	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
8	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
9	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
10	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
11	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
12	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
13	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
14	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
15	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
16	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
17	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
18	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
19	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
20	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
21	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
22	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
23	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
24	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
25	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
26	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
27	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
28	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
29	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
30	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
31	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
32	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
33	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
34	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
35	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00

Pirmiesiems duomenims rezultatai nepriklausė nuo nei vieno kintamojo, gali būti dėl to nes mokymosi ir testavimo aibės buvo didesnės.
Antri duomenys:

	function / learning rate																			
	binary										sigmoid									
epochs	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
2	0,65	0,65	0,65	0,65	0,65	0,90	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
3	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
4	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
5	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
6	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
7	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
8	0,90	0,65	0,65	0,65	0,65	0,90	0,85	0,65	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,80
9	0,90	0,85	0,90	0,65	0,90	0,65	0,85	0,90	0,85	0,90	0,65	0,65	0,65	0,75	0,65	0,65	0,65	0,65	0,75	0,85
10	0,90	0,90	0,90	0,85	0,85	0,90	0,90	0,90	0,90	0,90	0,65	0,70	0,65	0,65	0,65	0,85	0,65	0,65	0,65	0,65
11	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
12	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,65	0,65	0,75	0,65	0,65	0,65	0,65	0,65	0,65	0,65
13	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,65	0,65	0,65	0,65	0,65	0,90	0,65	0,65	0,65	0,65
14	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
15	0,90	0,90	0,95	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,75	0,65
16	0,95	0,90	0,95	0,90	0,90	0,90	0,90	0,90	0,90	0,95	0,65	0,65	0,65	0,65	0,75	0,65	0,90	0,90	0,65	0,65
17	0,95	0,90	0,95	0,90	0,90	0,90	0,90	0,95	0,90	0,95	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
18	0,95	0,95	0,95	0,95	0,95	0,95	0,65	0,95	0,90	0,95	0,65	0,80	0,65	0,65	0,90	0,65	0,65	0,75	0,65	0,65
19	0,95	0,95	0,95	0,95	0,95	0,95	0,90	0,95	0,65	0,65	0,65	0,65	0,65	0,85	0,65	0,65	0,65	0,90	0,65	0,65
20	0,65	0,95	0,95	0,95	0,95	0,95	0,65	0,95	0,90	0,95	0,65	0,65	0,65	0,85	0,65	0,65	0,65	0,90	0,65	0,65
21	0,95	0,90	0,95	0,65	0,95	0,65	0,95	0,65	0,90	0,90	0,70	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,90
22	0,90	0,65	0,90	0,95	0,90	0,95	0,90	0,95	0,90	0,65	0,70	0,65	0,80	0,65	0,85	0,65	0,90	0,75	0,65	0,85
23	0,90	0,95	0,95	0,90	0,95	0,90	0,90	0,90	0,90	0,95	0,70	0,65	0,65	0,85	0,65	0,65	0,90	0,65	0,90	0,90
24	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,70	0,65	0,65	0,65	0,65	0,65	0,90	0,90	0,65	0,65
25	0,90	0,90	0,90	0,90	0,90	0,90	0,85	0,90	0,90	0,90	0,75	0,65	0,65	0,65	0,85	0,65	0,90	0,65	0,65	0,65
26	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,75	0,70	0,80	0,80	0,65	0,65	0,65	0,65	0,65	0,65
27	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,75	0,70	0,70	0,65	0,65	0,75	0,90	0,65	0,65	0,65
28	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,65	0,90	0,90	0,75	0,70	0,75	0,65	0,65	0,75	0,90	0,65	0,75	0,75
29	0,90	0,85	0,90	0,90	0,90	0,65	0,90	0,95	0,85	0,85	0,75	0,75	0,70	0,85	0,75	0,65	0,90	0,65	0,95	0,75
30	0,90	0,90	0,90	0,90	0,90	0,65	0,90	0,65	0,65	0,65	0,75	0,75	0,75	0,85	0,65	0,65	0,65	0,90	0,90	0,65
31	0,90	0,90	0,70	0,65	0,90	0,90	0,70	0,90	0,90	0,90	0,75	0,75	0,75	0,65	0,85	0,65	0,90	0,90	0,65	0,65
32	0,75	0,75	0,95	0,65	0,90	0,90	0,65	0,90	0,90	0,90	0,75	0,75	0,75	0,80	0,90	0,65	0,90	0,75	0,65	0,95
33	0,95	0,90	0,90	0,90	0,75	0,70	0,90	0,90	0,70	0,75	0,75	0,75	0,75	0,85	0,90	0,65	0,90	0,65	0,90	0,90
34	0,90	0,90	0,75	0,90	0,95	0,95	0,65	0,70	0,90	0,95	0,75	0,75	0,75	0,75	0,90	0,65	0,90	0,90	0,75	0,90
35	0,65	0,65	0,95	0,75	0,90	0,90	0,90	0,95	0,65	0,90	0,75	0,75	0,70	0,75	0,90	0,75	0,90	0,65	0,90	0,90

Su pirmais duomenimis visiškai jokie kintamieji nedarė įtakos rezultatui.

Su antrais duomenimis matosi kad didžiausią itaką darė aktyvacijos funkcijos parinkimas:

Su slenkstine funkcija geriausias epochų skaičius yra tarp 18 ir 20 o mokymosi geriau <0,5.
Su sigmoidine funkcija kuo daugiau epochų tuo geriau, ir mokymosi greiti geriausias 0,9 arba 1,0.

Sprendžiant iš pirmų ir antrų duomenų geriausia kombinacija yra:

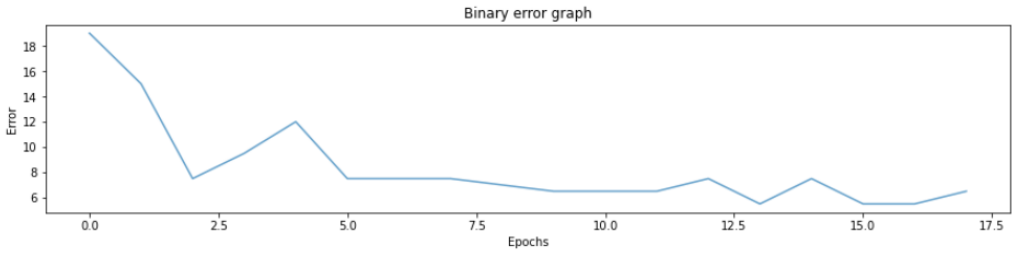
Funkcija – slenkstinė
Mokymosi greitis – 0.3
Epochų skaičius – 18

Geriausias variantas

```
def test_binary_return_data(weights, xtest, ytest):
    # Matuoja slenkstinio modelio taikluma
    success = 0
    guesses = []
    for i in range(len(xtest)):
        guess = binary(xtest[i],weights)
        guesses.append((xtest[i], guess, ytest[i][0]))
        if  guess== ytest[i][0]:
            success +=1
    return (success/len(xtest), guesses)
```

```
best = train_binary(0.3, 18, twox_train, twoy_train)
besttest = test_binary_return_data(best[0], twox_test, twoy_test)
print(" Gauti svoriai:", [round(i[0],4) for i in best[0]], "\n",
      "Paklaida:", best[1], "\n"
      " Tikslumas:", besttest[0])
```

```
 Gauti svoriai: [-5.74, -10.27, 8.36, 15.89, -6.1]
 Paklaida: 6.5
 Tikslumas: 0.95
```




```
for i in besttest[1]:
    print("Įejimas:",i[0], " Spėjimas:", i[1], "Tikra klasė:", i[2])
```

```
Įejimas: [6.9 3.2 5.7 2.3 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.5 2.8 4.6 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.7 3.3 5.7 2.5 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [5.7 2.6 3.5 1.  1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.6 3.  4.5 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.4 3.2 4.5 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.  2.9 4.5 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.6 2.5 3.9 1.1 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.3 2.3 4.4 1.3 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.  2.2 4.  1.  1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.7 3.1 5.6 2.4 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.4 2.7 5.3 1.9 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.9 3.1 5.1 2.3 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.7 3.  5.  1.7 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.7 3.1 4.7 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.7 2.8 4.5 1.3 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.2 2.7 3.9 1.4 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.  2.  3.5 1.  1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.2 3.4 5.4 2.3 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.5 3.2 5.1 2.  1. ]  Spėjimas: 0 Tikra klasė: 1
```

Išvados

Supratau kad kuo daugiau epochų nereiškia kad modelis bus tikslesnis, nes logiškiau mokyti modelį tol kol jo paklaida sumažėja iki norimo lygio.
Taip pat per didelis mokymosi greitis gali reikši kad svoriai šokinėja per dideliais tarpais ir nebus surandami optimalus svoriai, todėl geriau rinktis mažesnį mokymosį greiti jeigu yra tam resursų.
Didžiausia itaką šios užduoties rezultatams darė duomenų rinkinio dydis, nes kaip matėme su pirmais duomenim nesikeičia tikslumas, o su antrais duomenim labai šokinėja, tad pagrindinis kriterijus kuriant neuroninius tinklus yra turėti kuo įmanoma daugiau duomenų.