Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 1

по курсу «Теория искуственных нейронных сетей»

«Реализация однослойного перцептрона»

Студент группы ИУ9-71Б Окутин Д. А.

Преподаватель Каганов Ю. Т.

Москва 2024

1 Цель

Цель данной лабораторной работы:

1. Реализовать на языке высокого уровня однослойный персептрон и проверить его работоспособность на примере искусственных данных типа цифр от 0 до 9 и букв русского алфавита. Размер поля 5x4.

2. Исследовать работу персептрона на основе использования различных функций активации. (Линейной, сигмоиды, гиперболического тангенса, ReLu).

2 Задание

1. Сгенерировать датасет.

2. Реализовать однослойный персептрон, способный классифицировать цифры и буквы(A-J).

3. Протестировать персептрон, используя разные функции активации, построить графики.

3 Реализация

Исходный код представлен в листинге 1 - 4.

Листинг 1: Датасет

```
dataset = [
  ('0',[1, 1, 1, 1,
        1, 0, 0, 1,
        1, 0, 0, 1,
        1, 0, 0, 1,
        1, 1, 1, 1]),

  ('1',[0, 1, 1, 0,
        1, 1, 1, 0,
        0, 1, 1, 0,
        0, 1, 1, 0,
        1, 1, 1, 1]),

  ('2',[1, 1, 1, 1,
        0, 0, 0, 1,
        1, 1, 1, 1,
        1, 0, 0, 0,
```

```
18              1, 1, 1, 1]),
19
20      ('3',[1, 1, 1, 1,
21            0, 0, 0, 1,
22            1, 1, 1, 1,
23            0, 0, 0, 1,
24            1, 1, 1, 1]),
25
26      ('4',[1, 0, 0, 1,
27            1, 0, 0, 1,
28            1, 1, 1, 1,
29            0, 0, 0, 1,
30            0, 0, 0, 1]),
31
32      ('5',[1, 1, 1, 1,
33            1, 0, 0, 0,
34            1, 1, 1, 1,
35            0, 0, 0, 1,
36            1, 1, 1, 1]),
37      ('6',[1, 1, 1, 1,
38            1, 0, 0, 0,
39            1, 1, 1, 1,
40            1, 0, 0, 1,
41            1, 1, 1, 1]),
42
43      ('7',[1, 1, 1, 1,
44            0, 0, 0, 1,
45            0, 0, 1, 0,
46            0, 1, 0, 0,
47            1, 0, 0, 0]),
48
49      ('8',[1, 1, 1, 1,
50            1, 0, 0, 1,
51            1, 1, 1, 1,
52            1, 0, 0, 1,
53            1, 1, 1, 1]),
54
55      ('9',[1, 1, 1, 1,
56            1, 0, 0, 1,
57            1, 1, 1, 1,
58            0, 0, 0, 1,
59            1, 1, 1, 1]),
60
61      ('a', [
62          0, 1, 1, 0,
63          1, 0, 0, 1,
```

```
64          1, 1, 1, 1,
65          1, 0, 0, 1,
66          1, 0, 0, 1
67      ]),
68      ('b', [
69          1, 1, 1, 0,
70          1, 0, 0, 1,
71          1, 1, 1, 0,
72          1, 0, 0, 1,
73          1, 1, 1, 0
74      ]),
75      ('c', [
76          0, 1, 1, 1,
77          1, 0, 0, 0,
78          1, 0, 0, 0,
79          1, 0, 0, 0,
80          0, 1, 1, 1
81      ]),
82      ('d', [
83          1, 1, 1, 0,
84          1, 0, 0, 1,
85          1, 0, 0, 1,
86          1, 0, 0, 1,
87          1, 1, 1, 0
88      ]),
89      ('e', [
90          1, 1, 1, 1,
91          1, 0, 0, 0,
92          1, 1, 1, 0,
93          1, 0, 0, 0,
94          1, 1, 1, 1
95      ]),
96
97          ('f', [
98          1, 1, 1, 1,
99          1, 0, 0, 0,
100         1, 1, 1, 0,
101         1, 0, 0, 0,
102         1, 0, 0, 0
103     ]),
104     ('g', [
105         0, 1, 1, 1,
106         1, 0, 0, 0,
107         1, 0, 1, 1,
108         1, 0, 0, 1,
109         0, 1, 1, 1
```

```
    ]),
    ('h', [
        1, 0, 0, 1,
        1, 0, 0, 1,
        1, 1, 1, 1,
        1, 0, 0, 1,
        1, 0, 0, 1
    ]),
    ('i', [
        0, 1, 1, 1,
        0, 0, 1, 0,
        0, 0, 1, 0,
        0, 0, 1, 0,
        0, 1, 1, 1
    ]),
    ('j', [
        0, 1, 1, 1,
        0, 0, 0, 1,
        0, 0, 0, 1,
        1, 0, 0, 1,
        0, 1, 1, 0
    ]),

    ('0',[1, 1, 1, 1,
          1, 0, 0, 1,
          1, 0, 1, 1,
          1, 0, 0, 1,
          1, 1, 1, 1]),

    ('1',[0, 1, 1, 0,
          1, 1, 1, 0,
          0, 0, 1, 0,
          0, 1, 1, 0,
          1, 1, 1, 1]),

    ('2',[1, 1, 1, 1,
          0, 0, 1, 1,
          1, 1, 1, 1,
          1, 1, 0, 0,
          1, 1, 1, 1]),

    ('3',[1, 1, 1, 1,
          0, 0, 1, 1,
          1, 1, 1, 1,
          0, 0, 1, 1,
          1, 1, 1, 1]),
```

```
156
157     ('4',[0, 0, 0, 0,
158           1, 0, 0, 1,
159           1, 1, 1, 1,
160           0, 0, 0, 1,
161           0, 0, 0, 1]),
162
163     ('5',[1, 1, 1, 1,
164           1, 1, 0, 0,
165           1, 1, 1, 1,
166           0, 0, 1, 1,
167           1, 1, 1, 1]),
168     ('6',[1, 1, 0, 0,
169           1, 0, 0, 0,
170           1, 1, 1, 1,
171           1, 0, 0, 1,
172           1, 1, 1, 1]),
173
174     ('7',[0, 1, 1, 1,
175           0, 0, 1, 1,
176           0, 0, 1, 0,
177           0, 1, 0, 0,
178           1, 0, 0, 0]),
179
180     ('8',[1, 1, 1, 1,
181           1, 0, 0, 1,
182           0, 1, 1, 0,
183           1, 0, 0, 1,
184           1, 1, 1, 1]),
185
186     ('9',[1, 1, 1, 1,
187           1, 1, 0, 1,
188           1, 1, 1, 1,
189           0, 0, 1, 1,
190           0, 1, 1, 1]),
191
192     ('a', [
193         1, 1, 1, 1,
194         1, 0, 0, 1,
195         1, 1, 1, 1,
196         1, 0, 0, 1,
197         1, 0, 0, 1
198     ]),
199     ('b', [
200         1, 1, 1, 1,
201         1, 0, 0, 1,
```

```
            1, 1, 1, 0,
            1, 0, 0, 1,
            1, 1, 1, 1
        ]),
        ('c', [
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 0, 0, 0,
            1, 0, 0, 0,
            1, 1, 1, 0
        ]),
        ('d', [
            1, 1, 0, 0,
            1, 0, 1, 0,
            1, 0, 1, 0,
            1, 0, 1, 0,
            1, 1, 0, 0
        ]),
        ('e', [
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 1, 1, 0
        ]),

            ('f', [
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 1, 0, 0,
            1, 0, 0, 0,
            1, 0, 0, 0
        ]),
        ('g', [
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 0, 0, 0,
            1, 0, 1, 1,
            0, 1, 1, 1
        ]),
        ('h', [
            1, 0, 1, 0,
            1, 0, 1, 0,
            1, 1, 1, 0,
            1, 0, 1, 0,
            1, 0, 1, 0
```

```
    ]),
    ('i', [
        1, 1, 1, 1,
        0, 1, 1, 0,
        0, 1, 1, 0,
        0, 1, 1, 0,
        1, 1, 1, 1
    ]),
    ('j', [
        1, 1, 1, 1,
        0, 0, 0, 1,
        1, 0, 0, 1,
        1, 0, 0, 1,
        0, 1, 1, 1
    ]),

    ('0',[0, 0, 0, 0,
          0, 1, 1, 1,
          0, 1, 0, 1,
          0, 1, 0, 1,
          0, 1, 1, 1]),

    ('1',[0, 0, 0, 1,
          0, 0, 1, 1,
          0, 0, 0, 1,
          0, 0, 0, 1,
          0, 0, 1, 1]),

    ('2',[0, 1, 1, 1,
          0, 0, 0, 1,
          0, 1, 1, 0,
          1, 0, 0, 0,
          1, 1, 1, 0]),

    ('3',[0, 1, 1, 1,
          0, 0, 0, 1,
          0, 1, 1, 1,
          0, 0, 0, 1,
          0, 1, 1, 1]),

    ('4',[0, 1, 0, 1,
          0, 1, 0, 1,
          0, 1, 1, 1,
          0, 0, 0, 1,
          0, 0, 0, 1]),

```

```
294     ('5',[1, 1, 1, 0,
295           1, 0, 0, 0,
296           1, 1, 1, 0,
297           0, 0, 1, 0,
298           1, 1, 1, 0]),
299     ('6',[1, 1, 1, 0,
300           1, 0, 0, 0,
301           1, 1, 1, 0,
302           1, 0, 1, 0,
303           1, 1, 1, 0]),
304
305     ('7',[0, 1, 1, 1,
306           0, 0, 0, 1,
307           0, 0, 1, 0,
308           0, 1, 0, 0,
309           0, 0, 0, 0]),
310
311     ('8',[1, 1, 1, 0,
312           1, 0, 1, 0,
313           1, 1, 1, 0,
314           1, 0, 1, 0,
315           1, 1, 1, 0]),
316
317     ('9',[1, 1, 1, 0,
318           1, 0, 1, 0,
319           1, 1, 1, 0,
320           0, 0, 1, 0,
321           1, 1, 1, 0]),
322
323     ('0',[1, 1, 1, 1,
324           1, 0, 1, 1,
325           1, 0, 0, 1,
326           1, 0, 0, 1,
327           1, 1, 1, 1]),
328
329     ('1',[0, 1, 1, 0,
330           1, 1, 1, 0,
331           0, 1, 1, 0,
332           0, 1, 1, 0,
333           0, 1, 1, 0]),
334
335     ('2',[1, 0, 1, 1,
336           0, 0, 0, 1,
337           1, 1, 1, 1,
338           1, 0, 0, 0,
339           1, 1, 0, 1]),
```

```
340
341     ('3',[1, 1, 1, 0,
342           0, 0, 0, 1,
343           1, 1, 1, 1,
344           0, 0, 0, 1,
345           1, 1, 1, 0]),
346
347     ('4',[1, 0, 0, 1,
348           1, 0, 0, 1,
349           1, 1, 1, 1,
350           0, 0, 1, 1,
351           0, 0, 1, 1]),
352
353     ('5',[1, 1, 1, 1,
354           1, 0, 0, 0,
355           0, 1, 1, 1,
356           0, 0, 0, 1,
357           1, 1, 1, 1]),
358     ('6',[1, 1, 1, 1,
359           1, 0, 0, 0,
360           1, 0, 1, 1,
361           1, 0, 0, 1,
362           1, 1, 1, 1]),
363
364     ('7',[1, 1, 1, 1,
365           0, 0, 1, 0,
366           0, 0, 1, 0,
367           0, 1, 0, 0,
368           1, 0, 0, 0]),
369
370     ('8',[1, 1, 1, 0,
371           1, 0, 1, 0,
372           1, 1, 0, 0,
373           1, 0, 1, 0,
374           1, 1, 1, 0]),
375
376     ('9',[1, 1, 1, 1,
377           1, 0, 1, 1,
378           0, 1, 1, 1,
379           0, 0, 0, 1,
380           1, 1, 1, 1]),
381
382     ('a', [
383         1, 1, 1, 1,
384         1, 0, 0, 1,
385         1, 1, 1, 1,
```

```
            1, 0, 0, 1,
            0, 0, 0, 0
        ]),
        ('b', [
            0, 1, 1, 1,
            1, 0, 0, 1,
            1, 1, 1, 0,
            1, 0, 0, 1,
            0, 1, 1, 1
        ]),
        ('c', [
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 0, 0, 0,
            1, 0, 0, 0,
            1, 1, 1, 1
        ]),
        ('d', [
            1, 1, 1, 0,
            1, 0, 1, 1,
            1, 0, 1, 1,
            1, 0, 1, 1,
            1, 1, 1, 0
        ]),
        ('e', [
            1, 1, 1, 1,
            1, 1, 0, 0,
            1, 1, 1, 0,
            1, 1, 0, 0,
            1, 1, 1, 1
        ]),

        ('f', [
            1, 1, 1, 1,
            1, 1, 0, 0,
            1, 1, 1, 0,
            1, 0, 0, 0,
            1, 0, 0, 0
        ]),
        ('g', [
            1, 1, 1, 1,
            1, 0, 0, 0,
            1, 0, 1, 0,
            1, 0, 1, 1,
            1, 1, 1, 1
        ]),
```

```
    ('h', [
        1, 0, 0, 1,
        1, 0, 0, 1,
        0, 1, 1, 0,
        1, 0, 0, 1,
        1, 0, 1, 1
    ]),
    ('i', [
        1, 1, 1, 1,
        0, 0, 1, 0,
        0, 1, 1, 0,
        0, 1, 0, 0,
        1, 1, 1, 1
    ]),
    ('j', [
        1, 1, 1, 1,
        0, 0, 0, 1,
        0, 0, 0, 1,
        1, 1, 0, 1,
        1, 1, 1, 1
    ]),
]

X = []
y = []
for i in dataset:
    a,b = i
    X.append(b)
    y1=[0]*20
    if a in ['0','1','2','3','4','5','6','7','8','9','10']:
        y1[ord(a)-ord('0')]=1
    else:
        y1[10+ord(a)-ord('a')]=1
    y.append(y1)
```

Листинг 2: Функции активации

```
import math

def sigmoid(x):
    return 1/(1+math.e**(-x))
def sigmoid_der(x):
    return sigmoid(x)*(1-sigmoid(x))

def tanh(x):
    return (math.e**(2*x) - 1) / (math.e**(2*x) + 1)
def tanh_der(x):
```

```
11        return 1 - tanh(x)**2
12
13    def relu(x):
14        return x if x>0 else 0
15    def relu_der(x):
16        return 1 if x > 0 else 0
17
18    def softmax(arr):
19        exp_arr = [(math.e**x) for x in arr]
20        sum_exp = sum(exp_arr)
21        softmax_arr = [exp / sum_exp for exp in exp_arr]
22        return softmax_arr
23
24    def softmax_der(arr):
25        return [a*b for a,b in zip(softmax(arr),([1 - x for x in softmax(
      arr)]))]
26
27    def linear(x):
28        return x
29    def linear_der(x):
30        return 1
```

Листинг 3: Функции потерь

```
1    def mse_loss(predictions, labels):
2    errors = [(label-prediction) ** 2 for label, prediction in zip(labels,
       predictions)]
3    return sum(errors) / len(labels)
4
5
6  def der_mse(predictions, labels):
7    res = [(label-prediction) * (-1) for label, prediction in zip(labels,
       predictions)]
8    return res
9
10
11 def cross_entropy(target, pred):
12   #
13   pred = [max(min(p, 1 - 1e-8), 1e-8) for p in pred]
14
15   #                                    -
16   return -sum(t * math.log(p) for t, p in zip(target, pred)) / len(
       target)
17
18 def der_cross_entropy(target, pred):
19   #
20   pred = [max(min(p, 1 - 1e-8), 1e-8) for p in pred]
```

```
21
22  #                                                    -
23  res = [p - t for p, t in zip(pred, target)]
24
25  #
26  norm = math.sqrt(sum(r**2 for r in res))
27
28  #
29  return [r / norm for r in res]
```

Листинг 4: Однослойный персептрон

```
1   from tqdm import tqdm
2   from matplotlib import pyplot as plt
3   import random
4
5
6   class Perceptron:
7       def __init__(self, input_size, output_size,loss=mse_loss,
      activation=linear,activation_der=linear_der,epochs=1000,
      learning_rate=0.1):
8           self.weights = [[random.uniform(-1, 1) for _ in range(
      input_size)] for _ in range(output_size)]
9           self.biases = [random.uniform(-1, 1) for _ in range(
      output_size)]
10          self.learning_rate = learning_rate
11          self.activation = activation
12          self.activation_der = activation_der
13          self.loss=loss
14          self.total_loss = []
15          self.total_acc = []
16          self.epochs = epochs
17
18      def predict(self, inputs):
19          outputs = []
20          for weights, bias in zip(self.weights, self.biases):
21              z = sum(w * i for w, i in zip(weights, inputs)) + bias
22              outputs.append(self.activation(z))
23
24
25          return softmax(outputs)
26
27      def train(self, training_inputs, labels):
28          average_losses=[]
29          accuracy = []
30          for _ in tqdm(range(self.epochs)):
```

```
31                total_loss = 0
32                total_accuracy = 0
33                for inputs, label in zip(training_inputs, labels):
34                    predictions = self.predict(inputs)
35
36                    loss = self.loss(predictions, label)
37                    total_loss += loss
38                    total_accuracy += get_index_of_max_element(predictions
    ) == get_index_of_max_element(label)
39
40                    # errors = [(lb-prediction) for lb, prediction in zip(
    label, predictions)]
41                    errors = der_mse(predictions, label)
42
43                    predictions = softmax_der(predictions)
44                    for i in range(len(self.weights)):
45                        for j in range(len(self.weights[i])):
46                            gradient = errors[i] * self.activation_der(
    predictions[i])
47                            self.weights[i][j] -= self.learning_rate *
    gradient * inputs[j]
48
49                accuracy.append(total_accuracy/len(training_inputs))
50                average_losses.append(total_loss / len(training_inputs))
51
52            self.total_loss.append(average_losses)
53            self.total_acc.append(accuracy)
54        def show_graphics(self):
55            plt.plot(list(range(1, self.epochs+1)), self.total_loss[0],
    label="Train Loss")
56            plt.plot(list(range(1, self.epochs+1)), self.total_loss[1],
    label="Test Loss")
57            plt.grid()
58            plt.legend()
59            plt.show()
60
61            plt.plot(list(range(1, self.epochs+1)), self.total_acc[0],
    label="Train Acc")
62            plt.plot(list(range(1, self.epochs+1)), self.total_acc[1],
    label="Test Acc")
63            plt.grid()
64            plt.legend()
65            plt.show()
66
67
68    def get_index_of_max_element(arr):
```

```
69          max_value = max(arr)  #

70          max_index = arr.index(max_value)  #

71          return max_index
72
73      print(len(y))
74      X_train = X[:50]
75      y_train = y[:50]
76
77      X_test = X[50:]
78      y_test = y[50:]
79
80      perceptron = Perceptron(20, 20, epochs=500, loss=cross_entropy,
          activation=linear,activation_der=linear_der, learning_rate=0.01)
81      perceptron.train(X_train, y_train)
82
83      perceptron.train(X_test, y_test)
84
85      perceptron.show_graphics()
```

4 Результаты

Результат представлен на рисунках 1 - 8.

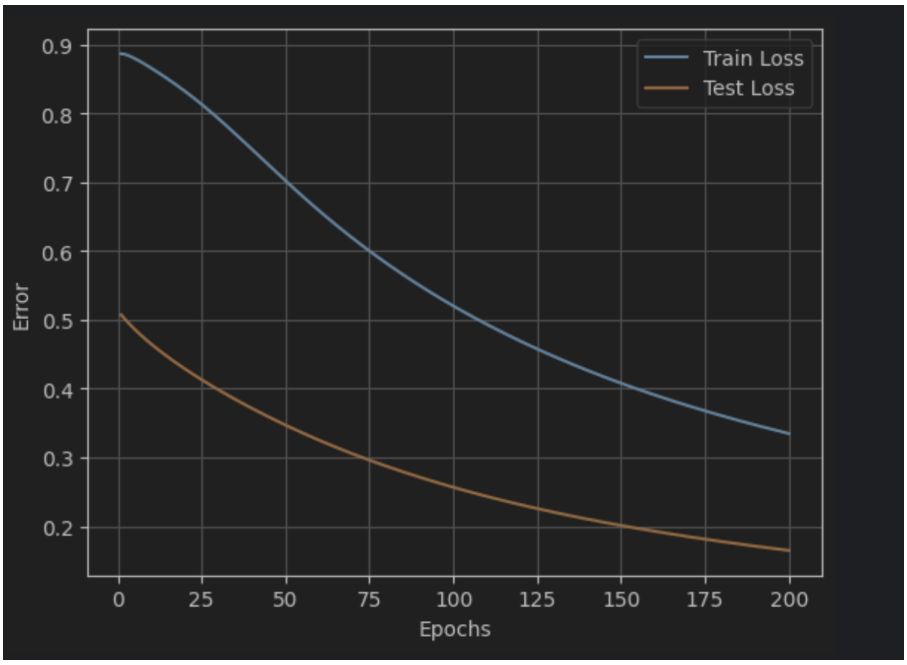

Рис. 1 — Зависимость ошибки от кол-ва эпох. Линейная функция активации

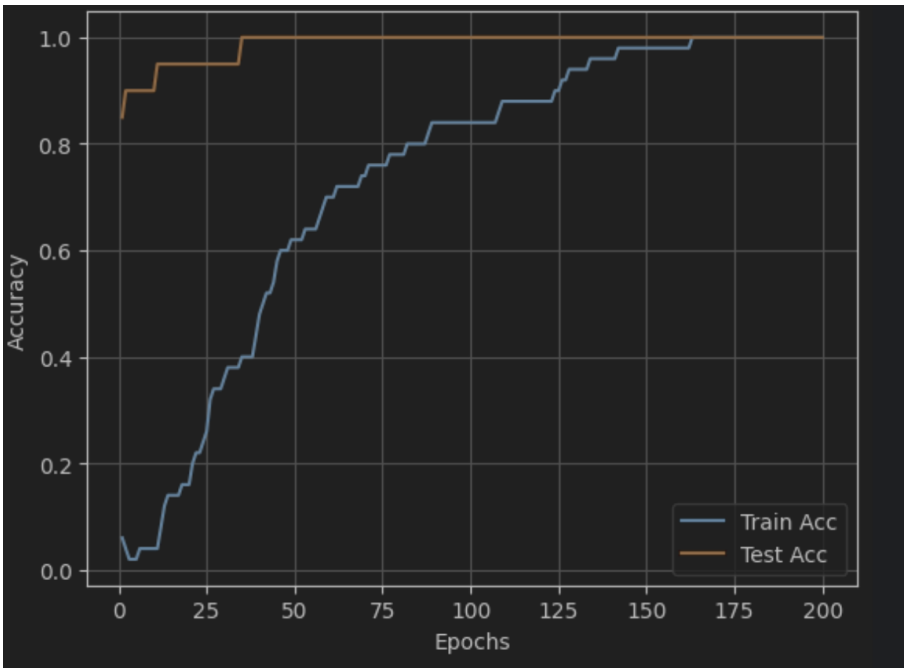

Рис. 2 — Зависимость точности предсказаний от кол-ва эпох. Линейная функция активации

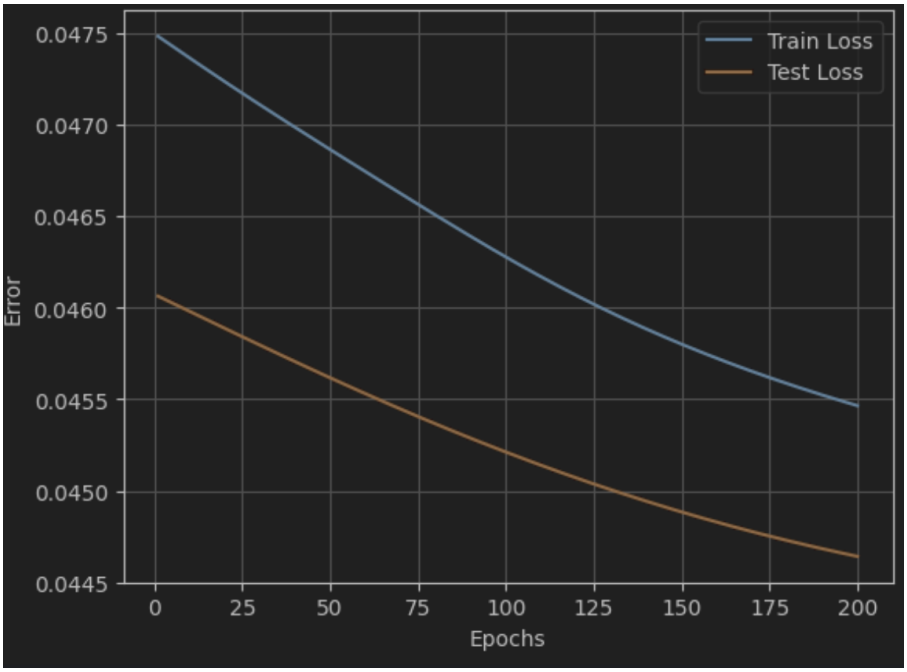

Рис. 3 — Зависимость ошибки от кол-ва эпох. Сигмоида

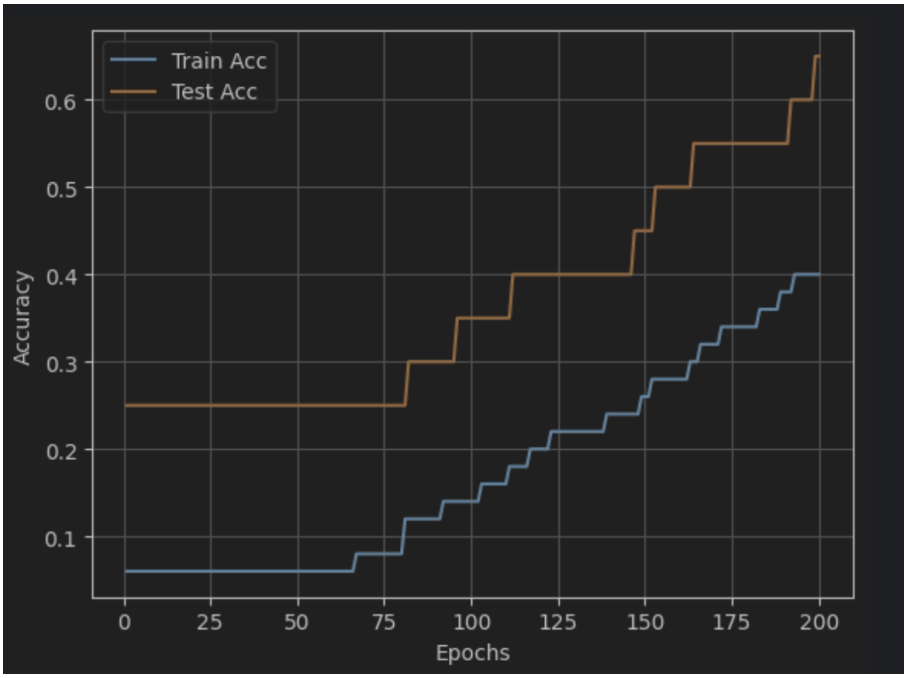

Рис. 4 — Зависимость точности предсказаний от кол-ва эпох. Сигмоида

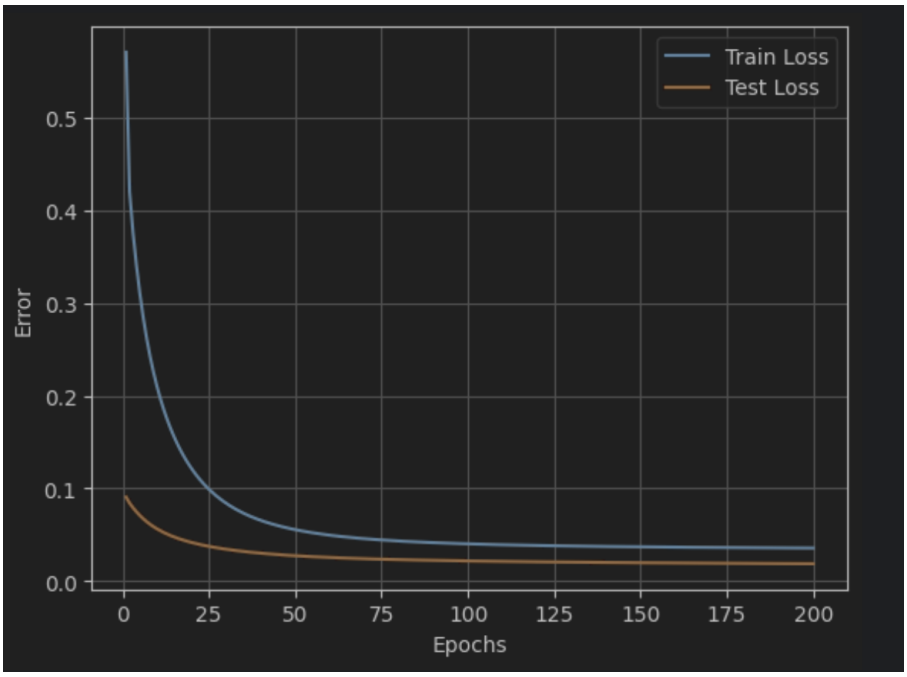

Рис. 5 — Зависимость ошибки от кол-ва эпох. Тангенс гиперболический

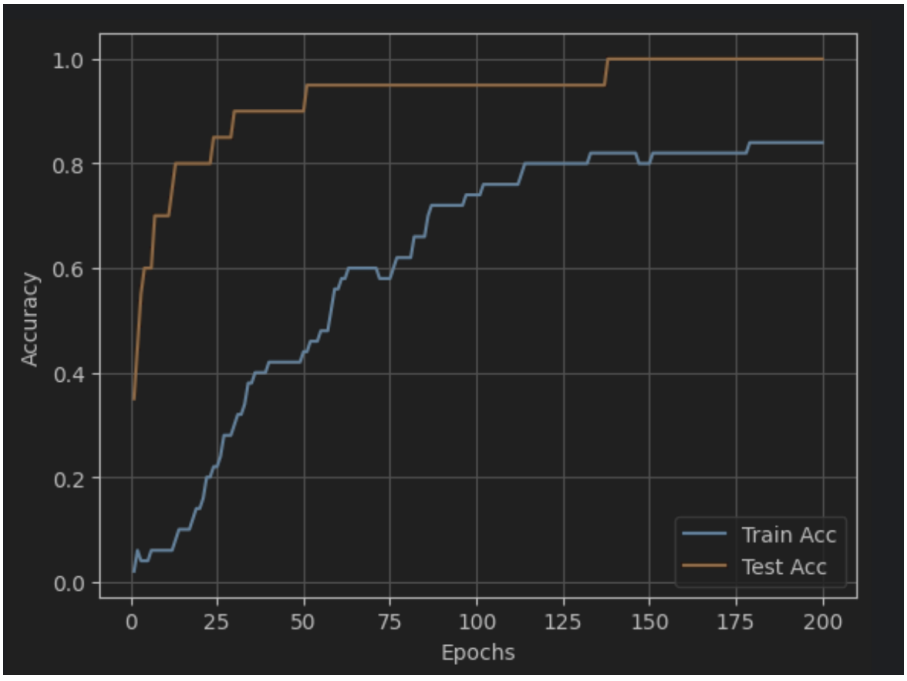

Рис. 6 — Зависимость точности предсказаний от кол-ва эпох. Тангенс гиперболический

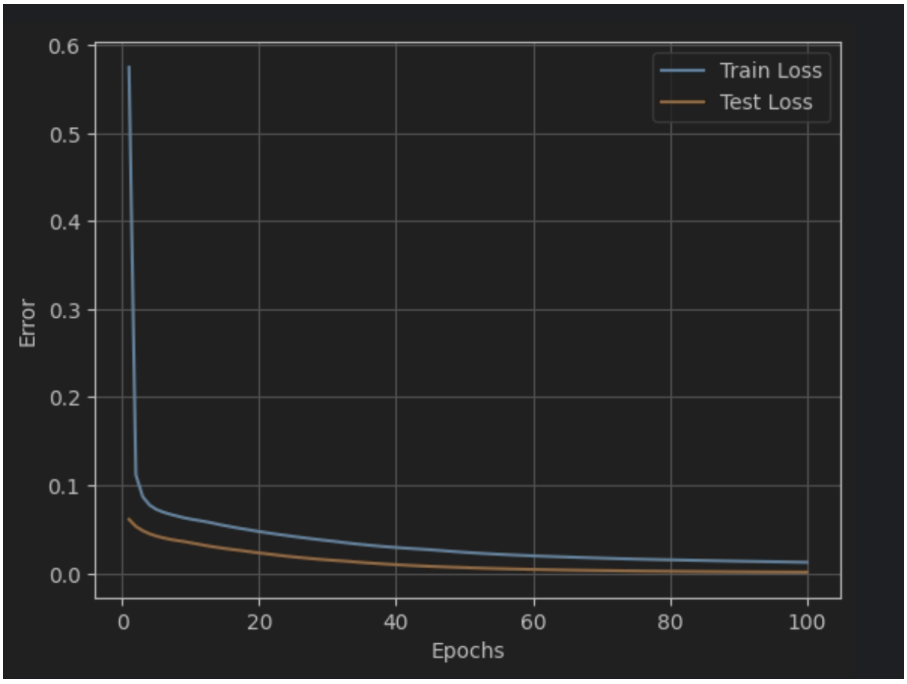

Рис. 7 — Зависимость ошибки от кол-ва эпох. RELU

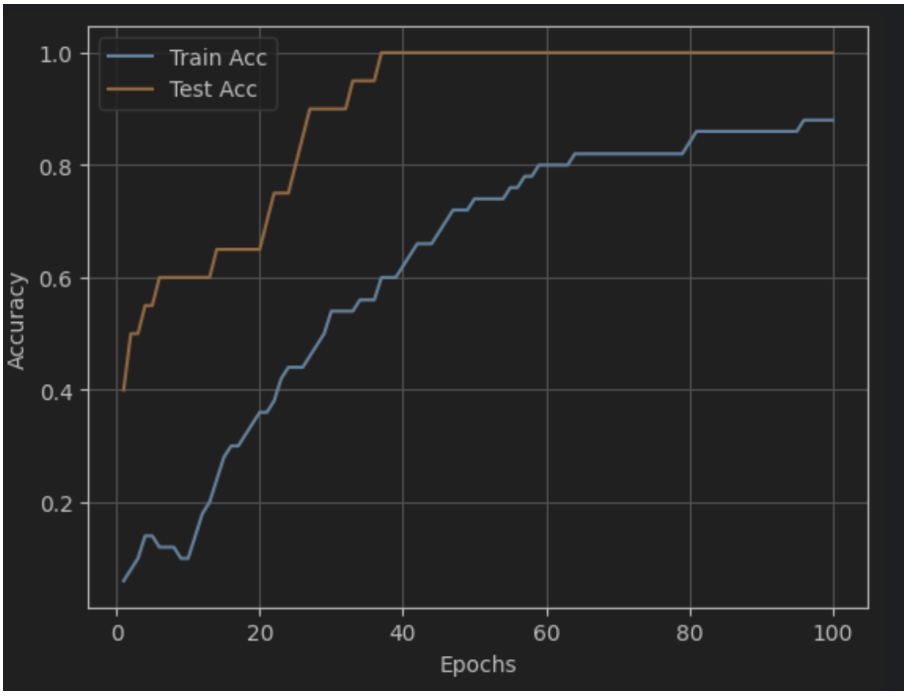

Рис. 8 — Зависимость точности предсказаний от кол-ва эпох. RELU

5 Выводы

Анализируя полученные результаты, можно видеть, что, используя линейную функцию активации и RELU нейросеть быстрее достигает 100 процентной точности предсказаний на тестовой выборке. Это можно объяснить тем, что для каждого нейрона в данной задаче используется бинарная классификация (т.к каждый нейрон отвечает за классификацию одной буквы или цифры), с данным типом классификации отлично справляются линейные функции активации. При увеличении количества эпох, ошибка, закономерно, уменьшается.