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 № 4

по курсу «Теория искусственных нейронных сетей»

«Dropout. L2-регуляризация»

Студент группы ИУ9-71Б Баев Д.А

Преподаватель Каганов Ю. Т.

Москва 2023

1 Задание

1. Реализовать Dropout в многослойном перцептроне.

2. Реализовать L2-регуляризацию в многослойном перцептроне.

3. Подготовить отчет с распечаткой текста программы, графиками результатов исследования и анализом результатов.

2 Исходный код

Исходный код программы представлен в листингах 1- 6

Листинг 1: Подготовка датасета

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torchvision
from tqdm import tqdm

MNIST_train = torchvision.datasets.MNIST('./', download=True, train=True
    )
MNIST_test = torchvision.datasets.MNIST('./', download=True, train=False
    )

count = 480
count_test = 80

train_X = MNIST_train.data.numpy()[:count]
train_Y = MNIST_train.targets.numpy()[:count]
test_X = MNIST_test.data.numpy()[:count_test]
test_Y = MNIST_test.targets.numpy()[:count_test]

train_X = np.array(list(map(lambda x: x.flatten() / 256, train_X)))
train_Y = np.array([np.array([int(i == x) for i in range(10)]) for x in
    train_Y])
test_X = np.array(list(map(lambda x: x.flatten() / 256, test_X)))
test_Y = np.array([np.array([int(i == x) for i in range(10)]) for x in
    test_Y])
```

Листинг 2: Определение функций активации и функций ошибки

```
def softmax(x):
    if np.linalg.norm(x) < 0.001:
        return np.zeros(len(x))
    x = x / np.linalg.norm(x)
    return np.exp(x)/(np.exp(x)).sum() if (np.exp(x)).sum() > 0.01 else
    np.zeros(len(x))

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return np.where(x > 0, 1, 0)

```

```
def mse(y_true, y_pred):
    return np.sum((y_true - y_pred) ** 2) / len(y_true)

def mse_derivative(y_true, y_pred):
    return 2 * (y_pred - y_true) / len(y_true)

def sigmoid(x): return 1/(1+np.exp(-x))
def sigmoid_derivative(x): return sigmoid(x)*(1-sigmoid(x))

def cross_entropy(y_true, y_pred):
    y_pred = np.clip(y_pred, 1e-8, 1 - 1e-8)
    return -np.mean(y_true * np.log(y_pred))

def cross_entropy_derivative(y_true, y_pred):
    y_pred = np.clip(y_pred, 1e-8, 1 - 1e-8)
    res = y_pred - y_true
    return res / np.linalg.norm(res)

def kl_divergence(y_true, y_pred):
    y_true = np.clip(y_true, 1e-8, 1- 1e-8)
    y_pred = np.clip(y_pred, 1e-8, 1-1e-8)
    return np.mean(y_true * np.log(y_true / y_pred))

def kl_divergence_derivative(y_true, y_pred):
    y_pred = np.clip(y_pred, 1e-8, 1 - 1e-8)
    res = y_pred - y_true
    return res / np.linalg.norm(res)
```

Листинг 3: Класс линейного слоя (с реализацией дропаута и L2-регуляризации)

```
class LinearLayer:
    def __init__(self, input_size, output_size, optimizer, model):
        self.adam_m = None
        self.adam_v = None
        self.adam_t = None
        self.inputs = None
        self.dropout_mask = None
        self.weights = np.random.rand(input_size + 1, output_size) - 0.5
        self.optimizer = optimizer
        self.model = model

    def forward(self, inputs, train=True):
        self.dropout_mask = np.random.choice(a=[0, 1], size=self.weights
    .shape, p=[self.model.d_chance, 1 - self.model.d_chance])
        inputs = np.append([1], inputs)

        if train:
```

```
            self.inputs = inputs
        return inputs @ (self.weights*self.dropout_mask)

    def calculate(self):
        return self.model.calculate(self, self.inputs[:-1])

    def backward(self, grad, train=True):
        if not train:
            return (grad @ self.weights.T)[1:]
        accum_grad = (grad @ (self.weights * self.dropout_mask).T)[1:]
        step_grad = np.outer(self.inputs, grad)
        step_grad *= self.dropout_mask
        if np.linalg.norm(step_grad) != 0:
            step_grad /= np.linalg.norm(step_grad)
        step = None
        match self.optimizer:
            case 0:
                # SGD
                step = step_grad
            case 1:
                # Adam
                if self.adam_t is None:
                    self.adam_t = 0
                    self.adam_m = np.zeros(self.weights.shape)
                    self.adam_v = np.zeros(self.weights.shape)
                self.adam_t += 1
                self.adam_m = self.model.beta1 * self.adam_m + (1 - self
    .model.beta1) * step_grad
                self.adam_v = self.model.beta2 * self.adam_v + (1 - self
    .model.beta2) * step_grad**2
                m = self.adam_m / (1 - self.model.beta1**self.adam_t)
                v = self.adam_v / (1 - self.model.beta2**self.adam_t)
                step = self.model.lr * m / np.sqrt(v + 10e-8)


        self.weights -= step

        if self.model.alpha is not None:
            l2_reg = self.model.alpha * self.model.lr * self.
    dropout_mask
            self.weights -= l2_reg
        return accum_grad
```

Листинг 4: Класс слоя активации

```
class ActivationLayer:
    def __init__(self, activation, activation_derivative):
```

```
        self.inputs = None
        self.activation = activation
        self.activation_derivative = activation_derivative
    def forward(self, inputs, train=True):
        self.inputs = inputs
        return self.activation(inputs)
    def backward(self, grad):
        return grad * self.activation_derivative(self.inputs)
```

Листинг 5: Класс перцептрона

```
class Perceptron:
    def __init__(self, input_size, sizes, loss, loss_derivative,
    optimizer, lr, d_chance, beta1, beta2, alpha):
        self.last_true = None
        self.layers = []
        self.beta1 = beta1
        self.beta2 = beta2
        self.d_chance = d_chance
        self.alpha = alpha
        prev_size = input_size
        for size in sizes:
            self.layers.append(LinearLayer(prev_size, size, optimizer,
    self))
            self.layers.append(ActivationLayer(sigmoid,
    sigmoid_derivative))
            prev_size = size
        self.layers.append(LinearLayer(prev_size, 10, optimizer, self))
        self.layers.append(ActivationLayer(softmax, lambda x: softmax(x)
     * (1 - softmax(x))))
        self.loss = loss
        self.loss_derivative = loss_derivative
        self.lr = lr

    def forward(self, inputs, train=True):
        result = inputs
        for layer in self.layers:
            result = layer.forward(result, train)
        return result

    def backward(self, y_true, y_pred):
        grad = self.loss_derivative(y_true, y_pred)
        for layer in self.layers[::-1]:
            grad = layer.backward(grad)

    def fit(self, inputs, y_true):
        self.last_true = y_true
```

```
33          y_pred = self.forward(inputs, True)
34          loss = self.loss(y_true, y_pred)
35          self.backward(y_true, y_pred)
36          return y_pred, loss
37
38      def calculate(self, layer, inputs):
39          place = self.layers.index(layer)
40          result = inputs
41          for layer in self.layers[place:]:
42              result = layer.forward(result, False)
43          return self.loss(self.last_true, result)
44
45      def nag_helper(self, layer, inputs):
46          place = self.layers.index(layer)
47          result = inputs
48          for layer in self.layers[place:]:
49              result = layer.forward(result, False)
50          grad = self.loss_derivative(self.last_true, result)
51          for layer in self.layers[place + 1:][::-1]:
52              grad = layer.backward(grad, False)
53          return grad
54
55      def train(self, epochs):
56          accuracy = []
57          loss_arr = []
58          for _ in tqdm(range(epochs)):
59              running_accuracy = 0
60              running_loss = 0
61              for inputs, y_true in zip(train_X, train_Y):
62                  y_pred, loss = self.fit(inputs, y_true)
63                  pred = np.argmax(y_pred)
64                  running_loss += loss
65                  running_accuracy += (np.argmax(y_true) == pred)
66              accuracy.append(running_accuracy / len(train_X))
67              loss_arr.append(running_loss / len(train_X))
68          return accuracy, loss_arr
69
70      def validate(self):
71          running_accuracy = 0
72          running_loss = 0
73          for inputs, y_true in zip(test_X, test_Y):
74              y_pred = self.forward(inputs, False)
75              loss = self.loss(y_true, y_pred)
76              pred = np.argmax(y_pred)
77              running_loss += loss
78              running_accuracy += (np.argmax(y_true) == pred)
```

```
79            return running_loss / len(test_X), running_accuracy / len(test_X
    )
```

Листинг 6: Функция эксперимента

```
1  def experiment(learning_rate, layer_count, layer_neurons, epochs,
       optimizer, loss, loss_der, d_chance, beta1, beta2, alpha):
2      perceptron = Perceptron(28 * 28, [layer_neurons for _ in range(
       layer_count)], loss, loss_der, optimizer, learning_rate, d_chance,
       beta1, beta2, alpha)
3
4      accuracy, loss = perceptron.train(epochs)
5
6
7      plt.plot(np.arange(len(accuracy)), accuracy)
8      plt.title("Accuracy")
9      plt.show()
10     plt.plot(np.arange(len(loss)), loss)
11     plt.title("Loss")
12     plt.show()
13
14     print(perceptron.validate())
```

3 Результаты

В качестве начального эксперимента были выбраны следующие параметры: learning rate - 0.0065, количество скрытых слоев (без учета входного и выходного слоев) - 1, количество нейронов в скрытом слое - 64, количество эпох - 40, функция потерь - перекрестная энтропия, оптимизатор - SGD. Дропаута и регуляризации пока нет.

График точности этой модели от количества эпох приведен на рисунке 1

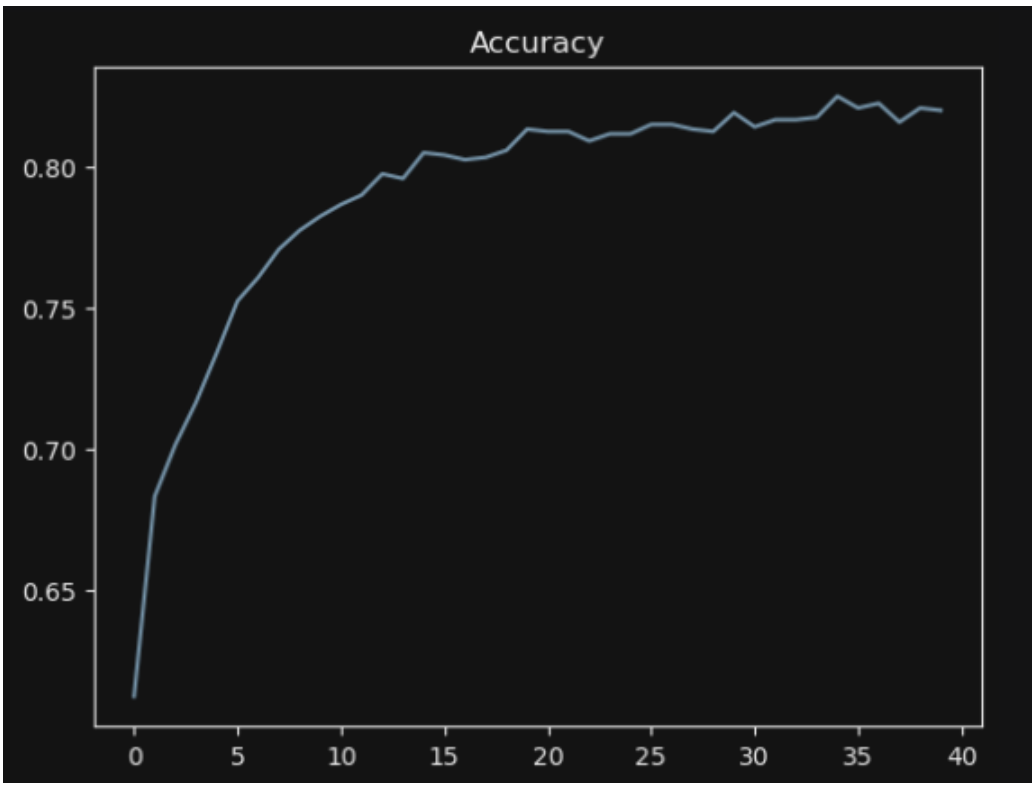

Рис. 1

Теперь добавляется дропаут с шансом 0.05. Видно серьезное улучшение результата.

График точности этой модели от количества эпох приведен на рисунке 2

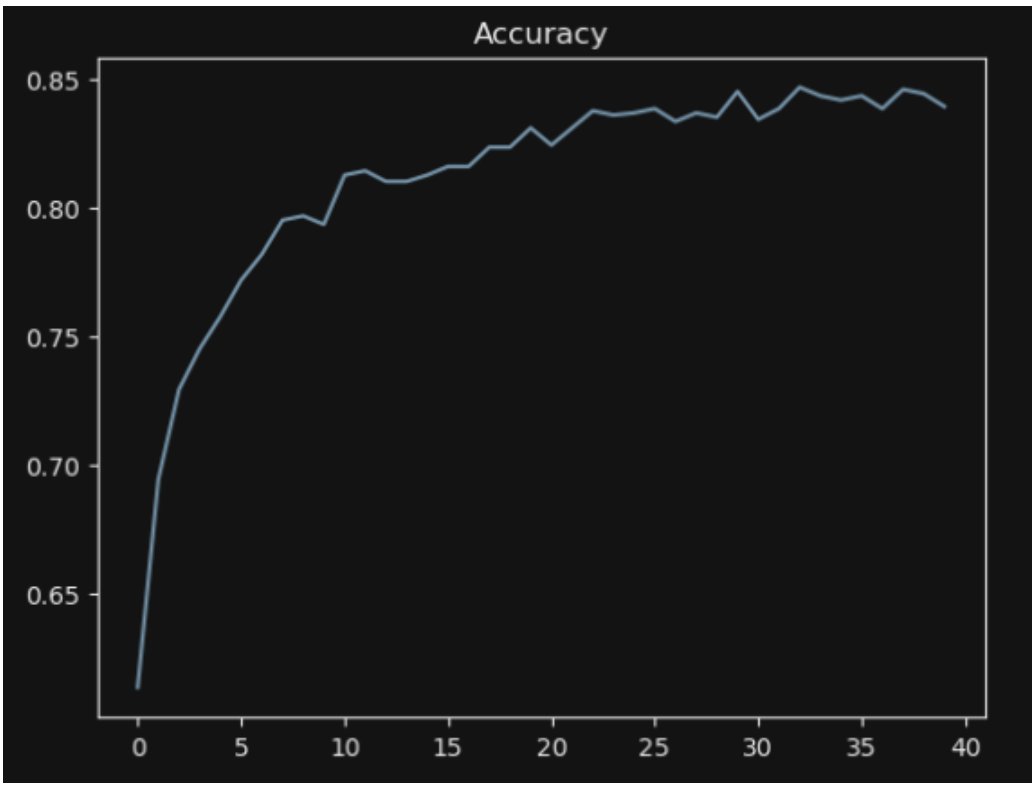

Рис. 2

Теперь шанс дропаута повышен до 0.1. Видна деградация результата до результатов первого эксперимента.

График точности этой модели от количества эпох приведен на рисунке 3

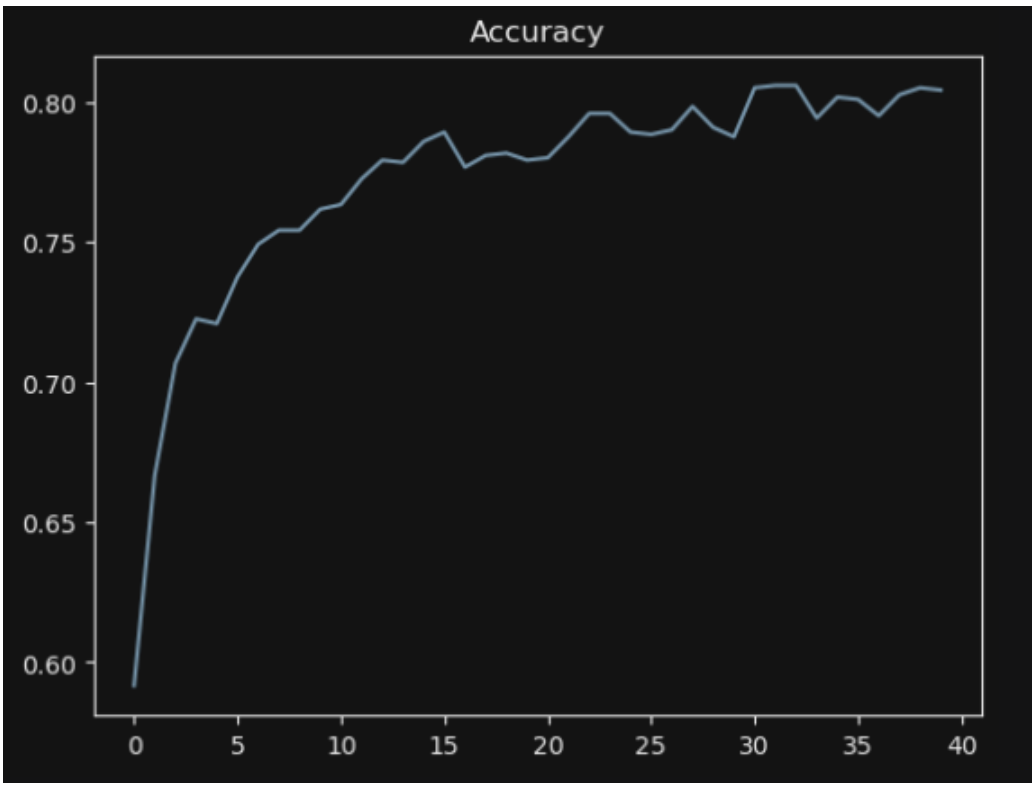

Рис. 3

Теперь шанс дропаута еще повышен до 0.2. Видно дальшейшее ухудшение результата.

График точности этой модели от количества эпох приведен на рисунке 4

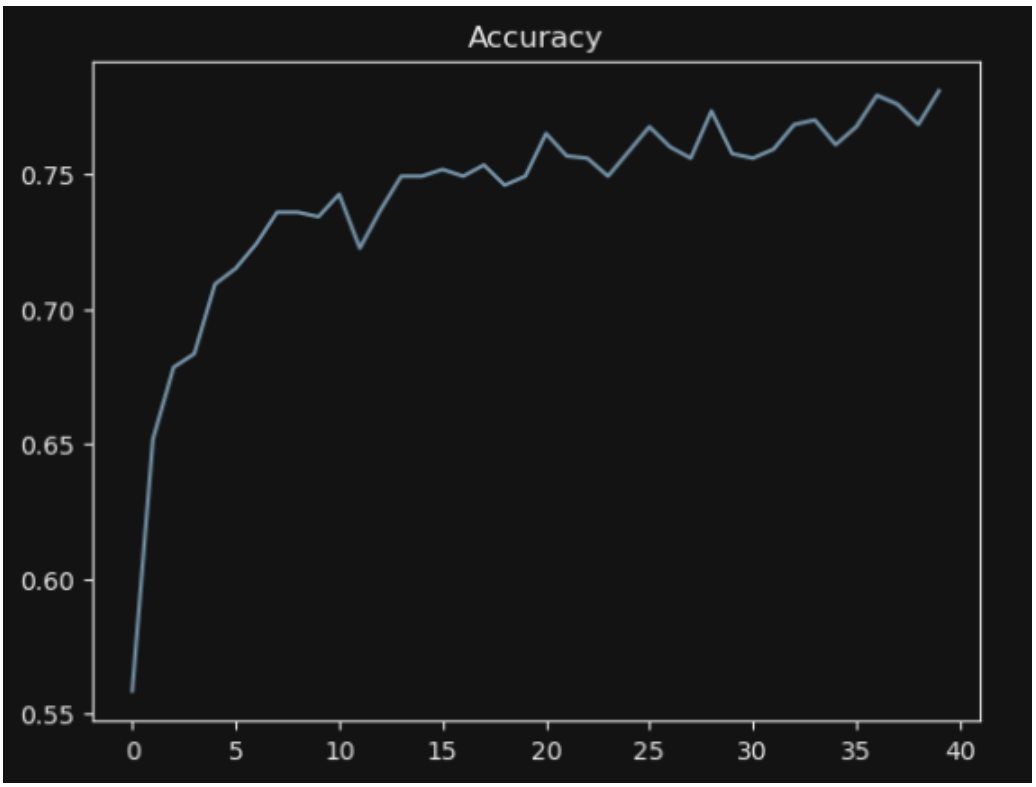

Рис. 4

Теперь добавляется L2-регуляризация (шанс дропаута пока что равен 0). Значение параметра альфа - 0.0005. Видно серьезное улучшение результата.

График точности этой модели от количества эпох приведен на рисунке 5

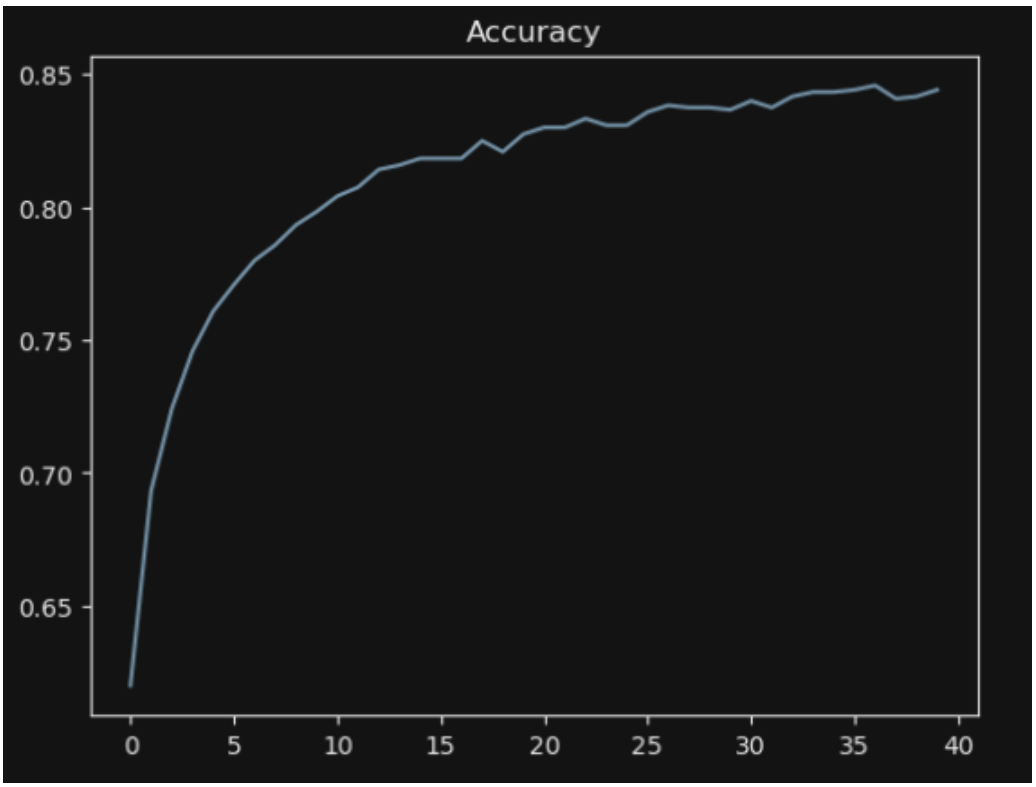

Рис. 5

Теперь параметр альфа равен 0.05. Видно, что результат стал чуть хуже по сравнению с предыдущим значением параметра.

График точности этой модели от количества эпох приведен на рисунке 6

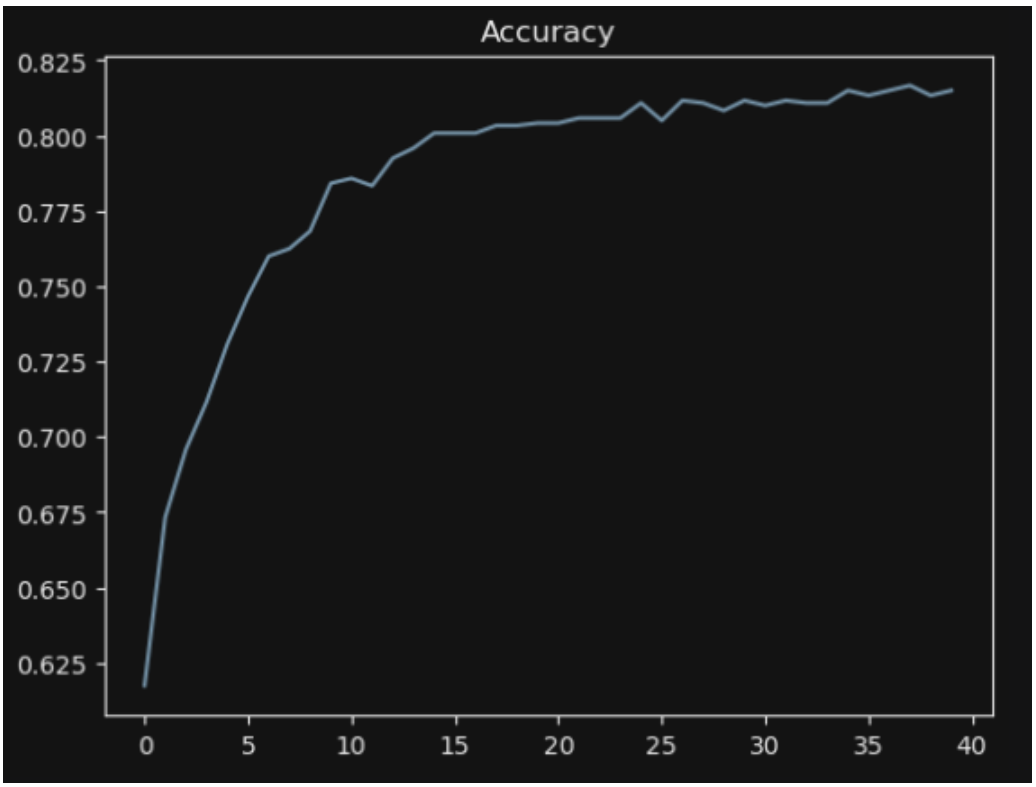

Рис. 6

Теперь одновременно применяются дропаут (шанс - 0.05) и L2-регуляризация (альфа - 0.0005). Этот эксперимент показывает, что в случае данной модели улучшения от дропаута и L2-регуляризации не складываются.

График точности этой модели от количества эпох приведен на рисунке 7

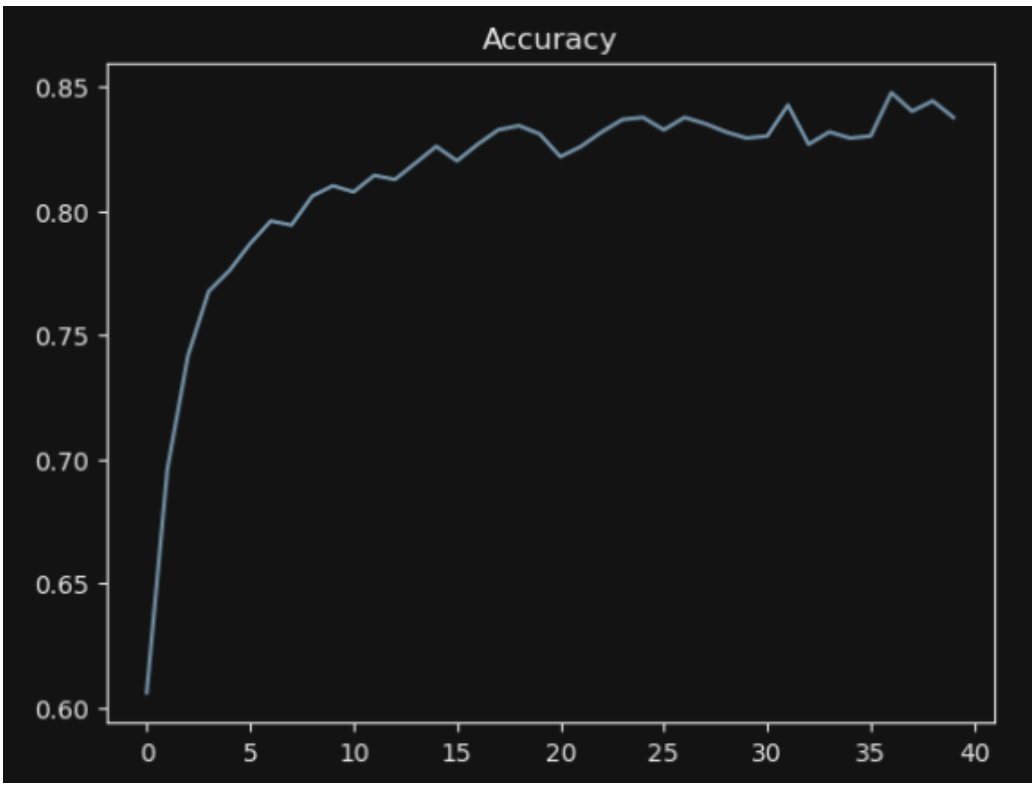

Рис. 7

4 Выводы

В рамках данной лабораторной работы были реализованы dropout и L2-регуляризация в многослойном перцептроне. В результате экспериментов удалось установить, что для решаемой задачи малый шанс дропаута и малое значение параметра альфа L2-регуляризации помогают повысить точность модели.