Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

**СИСТЕМЫ ОБРАБОТКИ МЕДИАДАННЫХ**

Лабораторная работа № 8

Распознавание рукописных цифр

Вариант № 14

группа № 850702

| Выполнил: | Турко В.Д. |
| --- | --- |
| Проверил | Рыбенков Е.В. |

Минск 2021

**1 ЦЕЛЬ РАБОТЫ**

Принципы использования искусственных нейронных сетей для распознавания образов.

**2 ЗАДАНИЕ**

1. Загрузить набор данных и инициализировать веса.
2. Провести обучение сети на 100 эпохах.
3. Выполнить проверку обученной сети на тестовом наборе данных.

**3 ХОД РАБОТЫ**

**3.1 Структура сети**

Реализация класса Dense, представляющего собой один слой сети:

class Dense:

def \_\_init\_\_(self, in\_dim, neurons\_count, h, h\_derive, lr):

np.random.seed(0)

self.weights = 2 \* \

np.random.random((neurons\_count, in\_dim + 1)) / \

np.sqrt(neurons\_count + in\_dim + 1)

self.cache = None

self.activation = h

self.activation\_derive = h\_derive

self.learning\_rate = lr

def forward(self, x):

x\_ext = np.vstack([x, np.ones((1, x.shape[1]))])

output = self.activation(self.weights @ x\_ext)

self.cache = (x\_ext, output)

return output

def backward(self, dE):

input = self.cache[0]

output = self.cache[1]

deriv\_act = dE

if self.activation\_derive is not None:

deriv\_act \*= self.activation\_derive(output)

deriv\_act = (deriv\_act / 1000) @ input.T

dE\_next = (dE.T @ self.weights).T

dE\_next = dE\_next[:dE\_next.shape[0] - 1]

self.weights -= deriv\_act \* self.learning\_rate

return dE\_next

Реализации функции активации “сигмоида” и её производной:

def sigmoid(i):

return 1/(1 + np.exp(-i))

def sigmoid\_derive(i):

return i \* (1 - i)

Реализация функции активации “softmax”:

def softmax(i):

e = np.exp(i - np.max(i, axis=0, keepdims=True))

e\_sum = np.sum(e, axis = 0, keepdims=True)

return e / e\_sum

Реализация функции для оценки ошибки и её производной:

def logloss(target, result):

return (-1 / target.shape[1]) \* np.sum(target \* np.log(result + 1e-5))

def logloss\_derive(target, result):

return result - target

**3.2 Подготовка сети**

Тестовый набор получается путем разделения датасета в пропорции 80/20, где 80 - набор для обучения, а 20 - тестовый набор. Далее представлена реализация функции обучения на 100 эпохах и проверки результатов на тестовом наборе, которая понадобится в дальнейшем.

def run\_lab(lr, first\_layer\_size: int = 256, show\_plot: bool = True):

m = scipy.io.loadmat('lab8/dataset.mat')

inputs = m['inputs']

targets = np.array([int\_to\_onehot(number[0]) for number in m['targets'].T]).T

bigger\_part = int(inputs.shape[1] \* 0.8)

training\_inputs = inputs[..., :bigger\_part]

training\_targets = targets[..., :bigger\_part]

test\_inputs = inputs[..., bigger\_part:]

test\_targets = targets[..., bigger\_part:]

layers =

[Dense(training\_inputs.shape[0], first\_layer\_size, sigmoid, sigmoid\_derive, lr),

Dense(first\_layer\_size, training\_targets.shape[0], softmax, None, lr)]

epoch\_errors = []

for i in range(100):

result = training\_inputs

for layer in layers:

result = layer.forward(result)

loss = logloss(training\_targets, result)

epoch\_errors.append(loss)

dE = logloss\_derive(training\_targets, result)

for layer in list(reversed(layers)):

dE = layer.backward(dE)

if (show\_plot):

plt.plot(epoch\_errors)

plt.show()

result = test\_inputs

for layer in layers:

result = layer.forward(result)

digits = np.argmax(test\_targets, axis=0)

predicted\_digits = np.argmax(result, axis=0)

prediction\_accuracy = np.sum(digits == predicted\_digits) / test\_inputs.shape[1]

return prediction\_accuracy

**3.3 Подбор конфигурации для получения наилучшего результата**

layer\_size\_range = np.linspace(10, 256, num=7)

lr\_range = np.linspace(0.1, 1, num=19)

best\_result = 0

best\_params = (layer\_size\_range[0], lr\_range[0])

for i, layer\_size in enumerate(layer\_size\_range):

ls = int(layer\_size)

layer\_result = []

for j, lr in enumerate(lr\_range):

result = run\_lab(lr=lr, first\_layer\_size=ls, show\_plot=False)

layer\_result.append(result)

if result > best\_result:

best\_result = result

best\_params = (ls, lr)

sub = plt.subplot(1, layer\_size\_range.size, i + 1)

sub.set\_title(str(i))

sub.stem(lr\_range, layer\_result)

print('Best accuracy:', best\_result, 'with params:', best\_params)

plt.show()

# Best accuracy: 0.9309090909090909 with params: (10, 0.4)

Подбор конфигурации осуществлялся простым перебором - запуском обучения и тестирования сети с различными значениями learning rate (от 0.1 до 1 c шагом в 0.05) и размера первого слоя сети (7 значений от 10 до 256). Результаты показаны на рисунке 1.1.

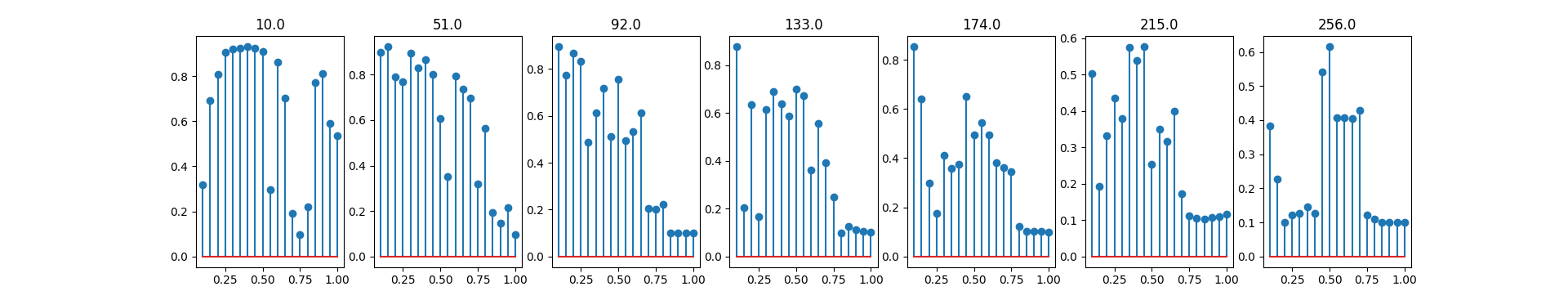


Рисунок 1 - Результаты работы сети при различных конфигурациях

**3.4 Тест сети**

Тестовый набор данных представляет собой 2200 примеров, аналогичных набору для обучения. По итогам теста сети с количеством нейронов на первом слое равным 10 и learning rate равным 0.4 было распознано 93% примеров.

**4 ВЫВОД**

В результате данной лабораторной работы была реализована двухслойная нейросеть, позволяющая распознавать рукописные цифры с точностью около 93%.