Лабораторная работа 2. Введение в нейронные сети с Python

Имя: Борисов Максим

Номер в ИСУ: 225169

Группа: R41341C

Задание

Реализовать нейронную сеть, которая способна распознавать рукописные цифры на изображении. Использовать датасет MNIST.

Необходимо

1. Импортировать библиотеки
2. Реализовать функцию инициализации параметров сети
3. Реализовать функцию инициализации весов сети
4. Реализовать функцию расчёта выходного значения сети
5. Реализовать функцию обучения нейронной сети
6. Реализовать функцию валидации нейронной сети
7. Реализовать функцию вывода изображений из датасета
8. Обучить сеть и провести валидацию

Замечание В работе для удобства нейронная сеть реализована через класс, соответственно представленный код по структуре и исполнению отличается от предложенного в материалах для подготовки.

1. Импортирование библиотек

```
1  import csv
2  from typing import List
3
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from scipy.special import expit
```

Библиотека `typing` необходима для указания сигнатур функций для удобства работы в IDE.

Библиотека `csv` используется для чтения данных из датасета.

2. Реализация функции инициализации параметров сети

Функция для получения параметров сети. Имеет флаг `default` при котором возвращает значения параметров по умолчанию без необходимости вводить их с клавиатуры.

```
74 def get_params(*, default: bool = False):
75     DEFAULT_INPUT_NODES = 784
76     DEFAULT_HIDDEN_NODES = 100
77     DEFAULT_OUTPUT_NODES = 10
78     DEFAULT_LEARNING_RATE = 0.5
79
80     if default:
81         in_nodes = DEFAULT_INPUT_NODES
82         hid_nodes = DEFAULT_HIDDEN_NODES
83         out_nodes = DEFAULT_OUTPUT_NODES
84         alpha = DEFAULT_LEARNING_RATE
85     else:
86         in_nodes = int(input('Enter number of nodes in input
            ↪  layer: ') or DEFAULT_INPUT_NODES)
87         hid_nodes = int(input('Enter number of nodes in hidden
            ↪  layer: ') or DEFAULT_HIDDEN_NODES)
88         out_nodes = int(input('Enter number of output nodes: ')
            ↪  or DEFAULT_OUTPUT_NODES)
89         alpha = float(input('Enter learning rate: ') or
            ↪  DEFAULT_LEARNING_RATE)
90
91     return in_nodes, hid_nodes, out_nodes, alpha
```

Эти данные передаются в конструктор класса `NetworkMNIST`, который также инициализирует веса случайными значениями. Однако ничто не мешает их переопределить.

```
 9 class NetworkMNIST:
10     def __init__(
11             self,
12             input_nodes: int,
13             hidden_nodes: int,
14             output_nodes: int,
15             activation_func=expit,
16             alpha: float = .5
17             ):
18         self.input_nodes = input_nodes
19         self.hidden_nodes = hidden_nodes
20         self.output_nodes = output_nodes
21         self.alpha = alpha
22         self.activation_func = activation_func
23         self._init_weights()
```

3. Реализация функции инициализации весов

```
25 def _init_weights(self):
26     self.in_hid_weights = np.random.uniform(-.5, .5,
        ↪  (self.hidden_nodes, self.input_nodes))
27     self.hid_out_weights = np.random.uniform(-.5, .5,
        ↪  (self.output_nodes, self.hidden_nodes))
```

Инициализировать веса одинаковыми значениями нет смысла, в таком случае сеть будет плохо обучаться. В случае со всеми весами равными 0.5 данная сеть в принципе всегда выдаёт на выходе единицы и не обучается вовсе (веса не изменяются).

Однако если инициализировать веса случайным образом (с зерном 42) и попробовать без обучения предсказать число для изображения номер 6 в датасете, то ответ будет 1.

4. Реализация функции расчёта выходного значения сети

```
29 def calculate(self, input: np.ndarray):
30     hid_in = self.in_hid_weights @ input
31     hid_out = self.activation_func(hid_in)
32     out_in = self.hid_out_weights @ hid_out
33     out_out = self.activation_func(out_in)
34     return out_out, hid_out
```

5. Реализация функции обучения сети

```
36  def train(self, training_data: List):
37      for input_image, label in training_data:
38          self.update_weights(input_image, label)
39
40  def update_weights(self, input_image: np.ndarray, label:
    ↪  np.ndarray):
41      out, hid_out = self.calculate(input_image)
42      out_epsilon = label - out
43      hid_epsilon = self.hid_out_weights.T @ out_epsilon
44      self.hid_out_weights += self.alpha * (out_epsilon * out * (1
        ↪  - out)) @ hid_out.T
45      self.in_hid_weights += self.alpha * (hid_epsilon * hid_out *
        ↪  (1 - hid_out)) @ input_image.T
```

6. Реализация функции валидации сети

```
47  def validate(self, validation_data: List):
48      results = []
49      for validation_image, label in validation_data:
50          results.append(self.validate_image(validation_image,
            ↪  label))
51      results = np.array(results, dtype=int)
52      print(f'Accuracy {np.mean(results) * 100}%')
53
54  def validate_image(self, validation_image: np.ndarray, label:
    ↪  np.ndarray):
55      return np.argmax(label) ==
        ↪  np.argmax(self.calculate(validation_image)[0])
```

7. Реализация функции вывода изображений из датасета

```
94  def display_and_save_image(image, label):
95      f = plt.figure()
96      plt.imshow(image.reshape((28, 28)), cmap='gray')
97      f.suptitle(f'Image of digit "{np.argmax(label)}"')
98      f.savefig(img_path + 'random_image.png')
```

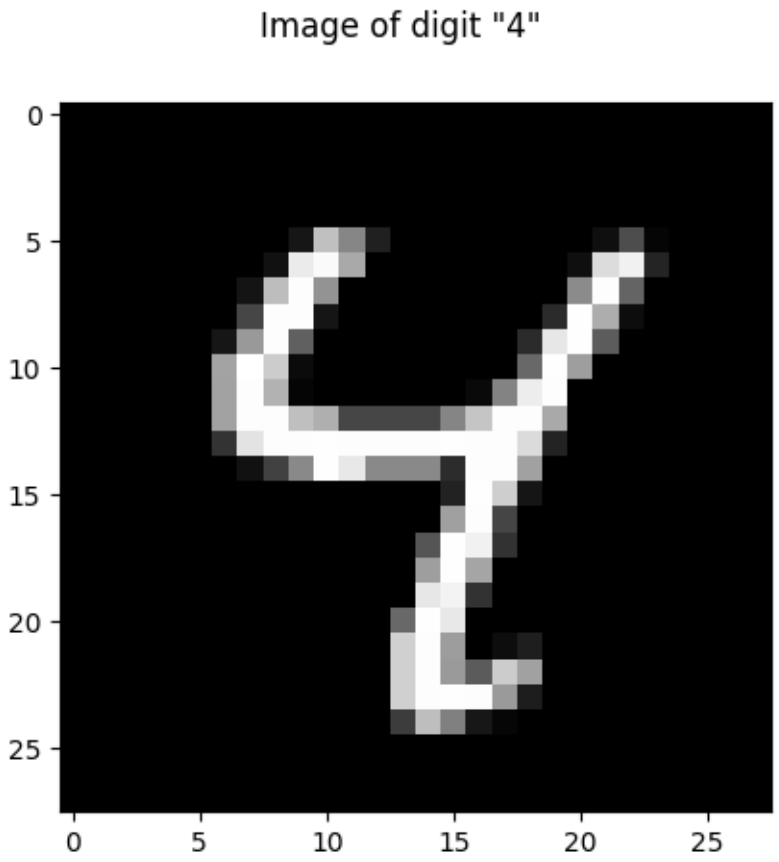

Рис. 1: Изображение из датасета соответствующее варианту

8. Обучение и валидация сети

```
120 # initialize net and feed data to it
121 nn = NetworkMNIST(in_nodes, hid_nodes, out_nodes, alpha=alpha)
122 nn.train(train_data)
123 nn.validate(test_data)
124
125 # a couple more training iterations
126 for _ in range(3):
127     print(f'Additional training iteration #{_ + 1}')
128     nn.train(train_data)
129     nn.validate(test_data)
```

Цифра соответствующая шестому (по номеру варианта), считая от нуля, элементу датасета это 4.

Полный код

```
import csv
from typing import List

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit


class NetworkMNIST:
    def __init__(
            self,
            input_nodes: int,
            hidden_nodes: int,
            output_nodes: int,
            activation_func=expit,
            alpha: float = .5
            ):
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes
        self.alpha = alpha
        self.activation_func = activation_func
        self._init_weights()

    def _init_weights(self):
        self.in_hid_weights = np.random.uniform(-.5, .5,
         ↪  (self.hidden_nodes, self.input_nodes))
        self.hid_out_weights = np.random.uniform(-.5, .5,
         ↪  (self.output_nodes, self.hidden_nodes))

    def calculate(self, input: np.ndarray):
        hid_in = self.in_hid_weights @ input
        hid_out = self.activation_func(hid_in)
        out_in = self.hid_out_weights @ hid_out
        out_out = self.activation_func(out_in)
        return out_out, hid_out

    def train(self, training_data: List):
        for input_image, label in training_data:
            self.update_weights(input_image, label)

    def update_weights(self, input_image: np.ndarray, label:
     ↪  np.ndarray):
        out, hid_out = self.calculate(input_image)
        out_epsilon = label - out
        hid_epsilon = self.hid_out_weights.T @ out_epsilon
```

```
44          self.hid_out_weights += self.alpha * (out_epsilon * out
             ↪  * (1 - out)) @ hid_out.T
45          self.in_hid_weights += self.alpha * (hid_epsilon *
             ↪  hid_out * (1 - hid_out)) @ input_image.T
46
47      def validate(self, validation_data: List):
48          results = []
49          for validation_image, label in validation_data:
50              results.append(self.validate_image(validation_image,
                 ↪  label))
51          results = np.array(results, dtype=int)
52          print(f'Accuracy {np.mean(results) * 100}%')
53
54      def validate_image(self, validation_image: np.ndarray,
         ↪  label: np.ndarray):
55          return np.argmax(label) ==
             ↪  np.argmax(self.calculate(validation_image)[0])
56
57
58  def data_loader(fname):
59      with open(fname, 'r') as file:
60          data = list(csv.reader(file))
61          data = np.array(data, dtype=int)
62      return data
63
64
65  def data_preprocessor(raw_data: np.ndarray):
66      processed_data = []
67      for row in raw_data:
68          image = np.array(row[1:] / 255, ndmin=2).T
69          label_vector = np.insert(np.zeros(9), row[0],
             ↪  1).reshape((10, 1))
70          processed_data.append((image, label_vector))
71      return processed_data
72
73
74  def get_params(*, default: bool = False):
75      DEFAULT_INPUT_NODES = 784
76      DEFAULT_HIDDEN_NODES = 100
77      DEFAULT_OUTPUT_NODES = 10
78      DEFAULT_LEARNING_RATE = 0.5
79
80      if default:
81          in_nodes = DEFAULT_INPUT_NODES
82          hid_nodes = DEFAULT_HIDDEN_NODES
83          out_nodes = DEFAULT_OUTPUT_NODES
84          alpha = DEFAULT_LEARNING_RATE
85      else:
```

```
 86          in_nodes = int(input('Enter number of nodes in input
             ↪  layer: ') or DEFAULT_INPUT_NODES)
 87          hid_nodes = int(input('Enter number of nodes in hidden
             ↪  layer: ') or DEFAULT_HIDDEN_NODES)
 88          out_nodes = int(input('Enter number of output nodes: ')
             ↪  or DEFAULT_OUTPUT_NODES)
 89          alpha = float(input('Enter learning rate: ') or
             ↪  DEFAULT_LEARNING_RATE)
 90
 91      return in_nodes, hid_nodes, out_nodes, alpha
 92
 93
 94  def display_and_save_image(image, label):
 95      f = plt.figure()
 96      plt.imshow(image.reshape((28, 28)), cmap='gray')
 97      f.suptitle(f'Image of digit "{np.argmax(label)}"')
 98      f.savefig(img_path + 'random_image.png')
 99
100
101  if __name__ == '__main__':
102      np.random.seed(42)  # to get consistent results
103
104      VARIANT = 6
105
106      img_path = '../img/'
107
108      train_data_path = '../data/mnist_train.csv'
109      test_data_path = '../data/mnist_test.csv'
110
111      # load and prepare data. define parameters
112      raw_train_data = data_loader(train_data_path)
113      raw_test_data = data_loader(test_data_path)
114
115      train_data = data_preprocessor(raw_train_data)  # [ (image,
         ↪  label_vector), ... ]
116      test_data = data_preprocessor(raw_test_data)  # [ (image,
         ↪  label_vector), ... ]
117
118      in_nodes, hid_nodes, out_nodes, alpha =
         ↪  get_params(default=True)
119
120      # initialize net and feed data to it
121      nn = NetworkMNIST(in_nodes, hid_nodes, out_nodes,
         ↪  alpha=alpha)
122      nn.train(train_data)
123      nn.validate(test_data)
124
125      # a couple more training iterations
```

```
126     for _ in range(3):
127         print(f'Additional training iteration #{_ + 1}')
128         nn.train(train_data)
129         nn.validate(test_data)
130 
131     # display image from set
132     image, label = test_data[VARIANT]
133     display_and_save_image(image, label)
```