Практическое задание № 4

ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ

Цель работы: получить практику обучения нейронной сети прямого распространения.

Содержание задания

1. Общие сведения

1. Ознакомиться с материалами лекции № 4.

2. Установить необходимое программное обеспечение. При выполнении задания наверняка понадобятся **Python 3**, **NumPy**, **SciPy**, и **Matplotlib**.

3. Ознакомиться с содержимым папки с заданием, которая включает в себя файлы, представленные ниже.

`main.py` – «основной» модуль, необходимый для выполнения задания, который поможет выполнить его поэтапно. Настоящий программный код не требует какой-либо коррекции!

`data.mat` – база данных для выполнения задания.

`displayData.py` – модуль, содержащий функцию `displayData`, которая необходима для визуализации данных. Данный модуль не требует коррекции!

`computeCost.py` – модуль, содержащий функцию `computeCost`, которая необходима для вычисления значения стоимостной функции трехслойной нейронной сети прямого распространения.

`randInitializeWeights.py` – модуль, содержащий функцию `randInitializeWeights`, которая необходима для выполнения случайной инициализации весов, расположенных на связях между смежными слоями нейронной сети прямого распространения.

`sigmoid.py` – модуль, содержащий функцию `sigmoid`, которая позволяет вычислить значение сигмоидной функции. Данный модуль не требует коррекции!

`sigmoidGradient.py` – модуль, содержащий функцию `sigmoidGradient`, которая позволяет вычислить значение производной сигмоидной функции.

`gradientDescent.py` – модуль, содержащий функцию `gradientDescent`, которая необходима для выполнения градиентного спуска с целью поиска параметров модели трехслойной нейронной сети прямого распространения.

predictNN.py – модуль, содержащий функцию predictNN, которая необходима для предсказания метки класса при решении задачи многоклассовой классификации с использованием трехслойной нейронной сети прямого распространения. Данный модуль не требует коррекции!

weights.mat – файл, содержащий обученные параметры модели нейронной сети прямого распространения.

4. Поэтапно выполнить задание, связанное с реализацией и исследованием нейронной сети прямого распространения.

5. Ответить на вопросы, необходимые для составления отчета по данному практическому заданию. Отчет сдается на проверку в печатной или письменной форме в указанные сроки.

2. Трехслойная нейронная сеть прямого распространения

При выполнении задания требуется заполнить пустые места программного кода в блоках с комментарием «Ваш код здесь». Данную процедуру необходимо выполнить для следующих функций: computeCost, sigmoidGradient, randInitializeWeights, gradientDescent.

1. При решении любой задачи с использованием инструментов машинного обучения важным является понимание структуры анализируемых данных и их визуализация в случае возможности. В настоящем задании предлагается использовать базу данных из файла **data.mat**. Данные представляют собой множество объектов, описываемых 400 признаками (нормализованные значения яркостей пикселей, описывающих рукописные цифры от 0 до 9) и меткой (от 0 до 9). Необходимо обратить внимание на то, что база данных в настоящем задании размечена, а метка принимает дискретный набор из десяти значений (0 для цифры «0», 1 для цифры «1» и т. д.). Поэтому в рамках настоящего задания рассматривается решение задачи многоклассовой классификации. За основу при формировании базы данных в задании была взята база рукописных цифр MNIST, описанная в лекции № 3. С оригинальной базой данных MNIST можно ознакомиться здесь: http://www.cs.nyu.edu/~roweis/data.html. При формировании файла **data.mat** база MNIST была усечена до 5000 объектов, разрешения изображений цифр уменьшены до 20x20 пикселей, что определяет длину вектора признаков в 400 значений, проведена нормализация признаков, поэтому при обучении использование отдельной функции нормализации не требуется. Визуализацию данных в настоящем задании можно выполнить с использованием функции displayData, расположенной в модуле **displayData.py**. Для выполнения визуализации данных,

а также для проверки правильности работы завершенных кодов, интерпретируйте файл **main_lr.py**. Результат визуализации представлен на рис. 1.

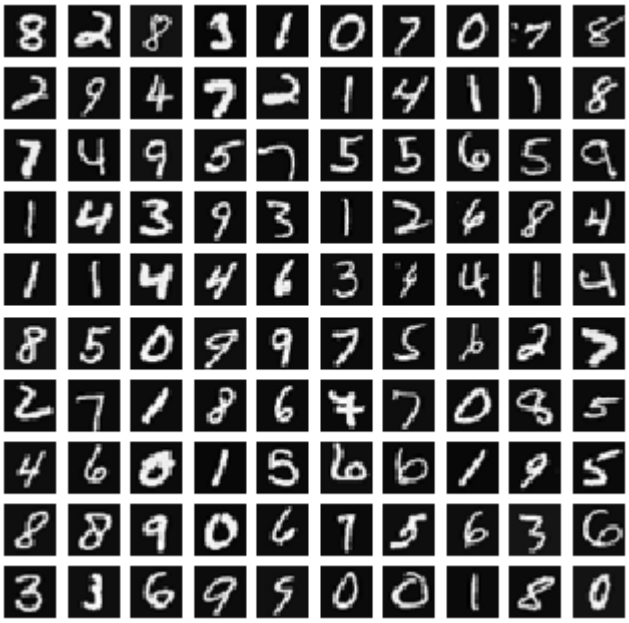

Рис. 1. Пример 100 случайно отобранных изображений цифр
из рассматриваемой в задании базы данных

2. Завершите программный код в модуле **computeCost.py**, который позволит вычислить значение стоимостной функции для трехслойной нейронной сети прямого распространения без учета и с учетом регуляризации. Формулы, описывающие ее вычисление, представлены в лекции № 4. При выполнении данной части задания могут понадобиться функции из библиотеки **NumPy**, представленные ниже.

`dot` – позволяет вычислить матричное произведение для двумерных массивов и скалярное произведение для одномерных массивов (без комплексного сопряжения).

`sum` – позволяет вычислить сумму элементов вдоль определенной размерности двумерного массива и сумму всех элементов для одномерного массива.

`log` – позволяет вычислить натуральный логарифм от элементов массива.

`transpose` – позволяет выполнить транспонирование массива. Для одномерного массива данная функция не оказывает никакого действия, а для двумерного массива использование функции соответствует обычному матричному транспонированию.

`concatenate` – выполняет объединение последовательности массивов вдоль определенной размерности.

`zeros` – позволяет сформировать массив заданной формы и типа, состоящий из нулевых значений.

`where` – позволяет возвратить номера элементов массива, удовлетворяющие определенному условию.

Так же совершенно будет необходима функция `sigmoid`, реализованная в модуле **`sigmoid.py`**, который находится в папке с заданием.

Необходимо отметить, что при выполнении настоящей части задания потребуется реализовать алгоритм прямого распространения сигнала через нейронную сеть для того, чтобы вычислить значение гипотезы (предсказание), сделанное нейронной сетью для некоторого примера из базы данных.

3. Завершите программный код в модуле **`sigmoidGradient.py`**, который позволит вычислить значение производной сигмоидной функции. При реализации можно воспользоваться функцией `sigmoid`, реализованной в модуле **`sigmoid.py`**.

4. Завершить программный код в модуле, выполняющем случайную инициализацию весов, **`randInitializeWeights.py`**. Веса должны быть инициализированы случайными значениями, распределенными по равномерному закону в интервале от -ε до ε. Значение ε рекомендуется выбрать равным 0.12. Пример визуализации весов, определенных на связях между входным и скрытым слоем при подобной случайной инициализации, представлен на рис. 2 (слева).

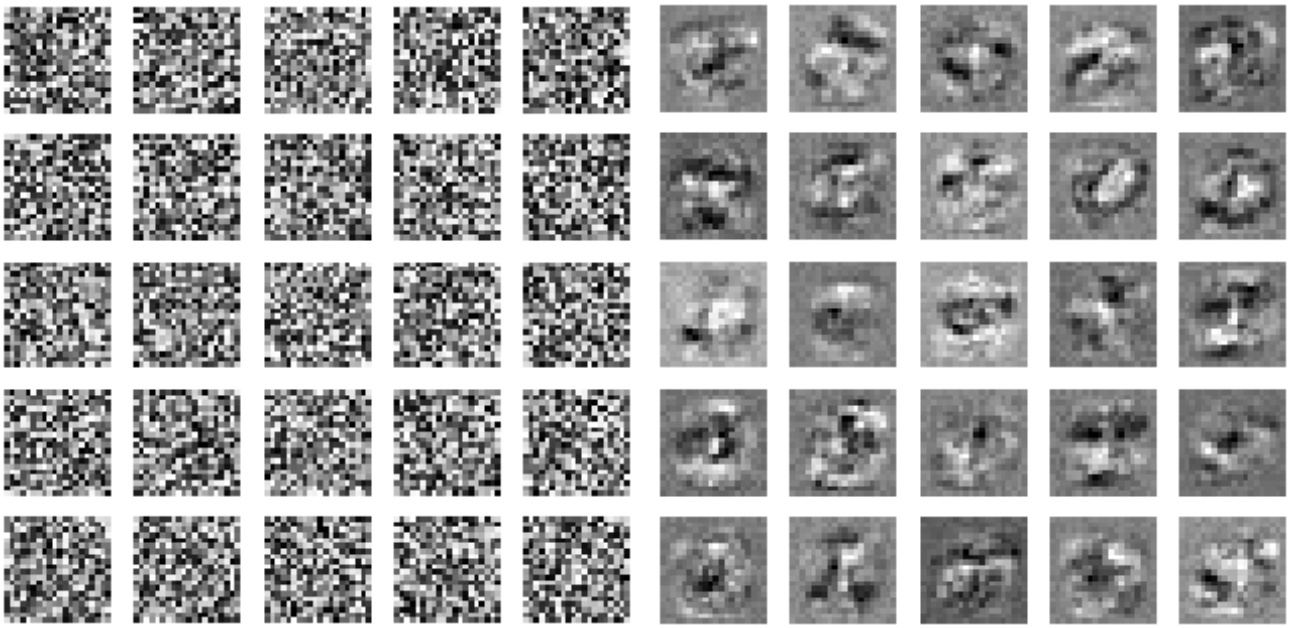

Рис. 2. Веса трехслойной нейронной сети прямого распространения, определенные на связях между входным и скрытым слоем до обучения (слева) и после (справа)

При выполнении данной части задания может понадобиться функция `rand` из библиотеки **NumPy**.

`rand` – позволяет сгенерировать массив заданной формы, состоящий из случайных значений, распределенных по равномерному закону в интервале от 0 до 1.

5. Завершите программный код в модуле **`gradientDescent.py`**, который позволит выполнить алгоритм градиентного спуска с целью обучения параметров модели трехслойной нейронной сети прямого распространения с учетом регуляризации. Формулы, описывающие реализацию градиентного спуска для данного случая, представлены в лекции № 4. При выполнении данной части задания могут понадобиться следующие функции из библиотеки **NumPy**: `dot`, `transpose`, `concatenate` и `ones`.

`ones` – позволяет сформировать массив заданной формы и типа, состоящий из единичных значений.

Так же совершенно будут необходимы функция `sigmoid` и `sigmoidGradient`, реализованные в модулях **`sigmoid.py`** и **`sigmoidGradient.py`**, которые находятся в папке с заданием.

```
for i in range(num_iters):

    print('Эпоха обучения №', i + 1)

    # ==================== Ваш код здесь ====================
    # Инструкция: выполнить алгоритм обратного распространения ошибки
    # с целью поиска частных производных от стоимостной функции по
    # параметрам модели

    Delta1 = np.zeros(Theta1.shape)
    Delta2 = np.zeros(Theta2.shape)

    for i in range(m):
        a1 = X[i:i + 1, :]
        a2 = sigmoid(np.dot(a1, Theta1.transpose()))
        a2 = np.concatenate((np.ones((1, 1)), a2), axis = 1)
        a3 = sigmoid(np.dot(a2, Theta2.transpose()))
        h  = a3

        delta3 = (h - Y[i:i + 1]).transpose()
        delta2 = np.dot(Theta2.transpose(), delta3)
        delta2 = delta2[1:, :] * sigmoidGradient((np.dot(a1, Theta1.transpose()))).transpose())

        Delta1 = Delta1 + np.dot(delta2, a1)
        Delta2 = Delta2 + np.dot(delta3, a2)

    Temp1 = np.copy(Theta1); Temp1[:, 0] = 0
    Temp2 = np.copy(Theta2); Temp2[:, 0] = 0
    Theta1_grad = Delta1 / m + Temp1 * lam / m
    Theta2_grad = Delta2 / m + Temp2 * lam / m
    # =========================================================

    Theta1 = Theta1 - alpha * Theta1_grad
    Theta2 = Theta2 - alpha * Theta2_grad

    J_history.append(computeCost(X, y, num_labels, Theta1, Theta2, lam)) # сохранение значений стоимостной функции
                                                                         # на каждой итерации

return Theta1, Theta2, J_history
```

Рис. 3. Завершенный блок программного кода для функции `gradientDescent`

Необходимо отметить, что при реализации градиентного спуска требуется вычисление частных производных от стоимостной функции. Для выполнения последнего при обучении нейронной сети прямого распространения потребуется реализовать алгоритм обратного распространения ошибки, рассмотренный в лекции № 4. В свою очередь реализация данного алгоритма потребует написания программного кода для прямого распространения сигнала через нейронную сеть. Подсказка реализации алгоритма обратного распространения ошибки представлена на рис. 3.

6. После завершения предыдущих пунктов выполните обучение параметров модели трехслойной нейронной сети прямого распространения с настройками градиентного спуска, заданными по умолчанию в файле **`main.py`**, и определите долю правильных ответов. График зависимости значения стоимостной функции нейронной сети от числа итераций градиентного спуска должен выглядеть примерно так, как представлено на рис. 4.

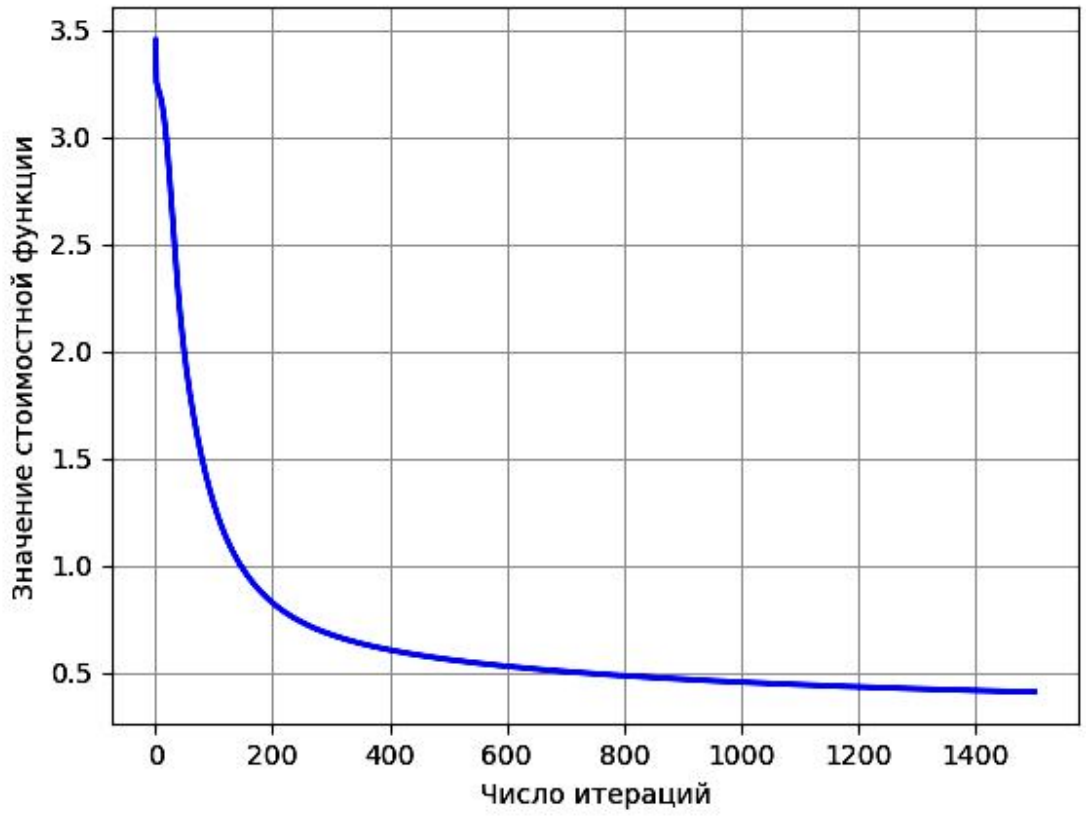

Рис. 4. Пример сходимости градиентного спуска для удачно подобранной скорости сходимости

На этапе начального тестирования кодов, выполняющих процедуру обучения нейронной сети, рекомендуется выбирать число шагов градиентного спуска (эпох обучения нейронной сети) малым, например,

10, 50, 100. Убедившись, что алгоритм сходится, можно запустить полную процедуру обучения для большого числа итераций. Пример весов, определенных на связях между входным и скрытым слоем, после обучения представлен на рис. 2 (справа).

3. Вопросы для составления отчета

1. Чему равно значение стоимостной функции трехслойной нейронной сети прямого распространения, обученные веса которой содержатся в файле `weights.mat`, для параметра регуляризации равного 0 (**20 баллов**)?
2. Чему равно значение стоимостной функции трехслойной нейронной сети прямого распространения, обученные веса которой содержатся в файле `weights.mat`, для параметра регуляризации равного 1 (**20 баллов**)?
3. Чему равны значения производной сигмоидной функции в точках 1, -0.5, 0, 0.5, 1 (**10 баллов**)?
4. Чему равна доля правильных ответов обученной (с использованием метода обратного распространения ошибки) трехслойной нейронной сети прямого распространения? Параметр сходимости градиентного спуска выставить равным 1, число итераций – 1500, а значение параметра регуляризации – 1 (**50 баллов**).