|  |  |  |  |
| --- | --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | | |
| Факультет | Компьютерных сетей и систем | | |
| Кафедра | Информатики  Дисциплина: Конструирование те технологии электронных вычислительных средств | | |
|  |  | | |
| ЛАБОРАТОРНАЯ РАБОТА №4  «Нейронные сети» | | | |
| Магистрант:  гр. 858341  Жишкевич С.А. | |  | Проверил:  Стержанов М. В. |
| Минск, 2019 | | | |

# 

ХОД РАБОТЫ

**Задание.**

Набор данных ex4data1.mat (такой же, как в лабораторной работе №2) представляет собой файл формата \*.mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 20x20 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 400 элементов. В результате загрузки набора данных должна быть получена матрица 5000x400. Далее расположены метки классов изображений от 1 до 9 (соответствуют цифрам от 1 до 9), а также 10 (соответствует цифре 0).

1. Загрузите данные ex4data1.mat из файла.
2. Загрузите веса нейронной сети из файла ex4weights.mat, который содержит две матрицы Θ(1) (25, 401) и Θ(2) (10, 26). Какова структура полученной нейронной сети?
3. Реализуйте функцию прямого распространения с сигмоидом в качестве функции активации.
4. Вычислите процент правильных классификаций на обучающей выборке. Сравните полученный результат с логистической регрессией.
5. Перекодируйте исходные метки классов по схеме one-hot.
6. Реализуйте функцию стоимости для данной нейронной сети.
7. Добавьте L2-регуляризацию в функцию стоимости.
8. Реализуйте функцию вычисления производной для функции активации.
9. Инициализируйте веса небольшими случайными числами.
10. Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.
11. Для того, чтобы удостоверится в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром ε = 10-4.
12. Добавьте L2-регуляризацию в процесс вычисления градиентов.
13. Проверьте полученные значения градиента.
14. Обучите нейронную сеть с использованием градиентного спуска или других более эффективных методов оптимизации.
15. Вычислите процент правильных классификаций на обучающей выборке.
16. Визуализируйте скрытый слой обученной сети.
17. Подберите параметр регуляризации. Как меняются изображения на скрытом слое в зависимости от данного параметра?
18. Ответы на вопросы представьте в виде отчета.

**Результат выполнения:**

1. Загрузите данные ex4data1.mat из файла.

1. img\_data = scipy.io.loadmat('ex4data1.mat')
3. X, Y = img\_data['X'], img\_data['y']
5. **print**(f'X.shape = {X.shape}')
6. **print**(f'Y.shape = {Y.shape}')
7. X.shape = (5000, 400)
8. Y.shape = (5000, 1)

2. Загрузите веса нейронной сети из файла ex4weights.mat, который содержит две матрицы Θ(1) (25, 401) и Θ(2) (10, 26). Какова структура полученной нейронной сети?

Загрузка весов:

1. weights\_data = scipy.io.loadmat('ex4weights.mat')
2. theta1 = weights\_data['Theta1']
3. theta2 = weights\_data['Theta2']
5. **print**(f'theta1.shape = {theta1.shape}')
6. **print**(f'theta2.shape = {theta2.shape}')
7. theta1.shape = (25, 401)
8. theta2.shape = (10, 26)

Структура нейросети состоит из 3х слоев:

* входной слой - 400 нейронов + 1 bias:
* скрытый слой – 25 нейронов + 1 bias;
* выходной слой – 10 нейронов.

Количество нейронов выходного слоя равно количеству классов.

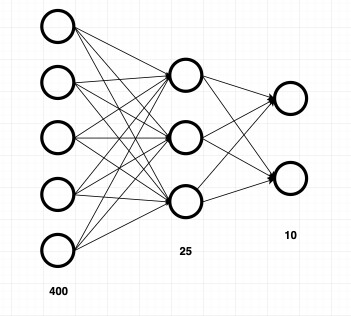


Рисунок 1 – структура нейросети.

3. Реализуйте функцию прямого распространения с сигмоидом в качестве функции активации..

Функция сигмоида:

1. **def** sigmoid(z):
2. **return** 1.0 / (1 + np.exp(-z))

Вспомогательные функции:

1. **def** add\_bias\_vec(a):
2. **return** np.insert(a,0,1,axis=1)
4. **def** rm\_bias(input):
5. **return** input[:, 1:]

Функцию прямого распространения:

1. **def** forward\_prop\_vec\_all(thetas, X):
2. a1 = add\_bias\_vec(X)
3. z2 = np.dot(a1, thetas[0].T)
4. a2 = sigmoid(z2)
6. a2 = add\_bias\_vec(a2)
7. z3 = np.dot(a2, thetas[1].T)
8. a3 = sigmoid(z3)
10. **return** {'a1': a1, 'z2': z2, 'a2': a2, 'z3': z3, 'a3': a3}

13. **def** forward\_prop\_vec(thetas, X):
14. **return** forward\_prop\_vec\_all(thetas, X)['a3']

4. Вычислите процент правильных классификаций на обучающей выборке. Сравните полученный результат с логистической регрессией..

Функция предсказания:

1. **def** predict(thetas, x):
2. **if** x.ndim == 1:
3. x = x.reshape(1, -1)
5. fp\_res = forward\_prop\_vec(thetas, x)
7. **return** np.argmax(fp\_res[0]) + 1

Функция подсчета “accuracy”:

1. **def** calc\_accuracy(thetas, X, Y):
2. m = X.shape[0]
3. correct = 0
5. **for** i **in** range(m):
6. **if** predict(thetas, X[i]) == Y[i]:
7. correct += 1
9. **return** correct / m

Accuracy: 99.0%

Нейросеть дала точность - 99.0%, что на несколько процентов больше, чем логистическая регрессия, которая дала - 97.2%.

5. Перекодируйте исходные метки классов по схеме one-hot.

Функция кодирования в «one-hot»:

1. **def** one\_hot(labels):
2. m = len(labels)
3. uniq\_labels = np.unique(labels)
5. **return** (labels == uniq\_labels).astype(int)

Результат работы:

1. Y\_oh = one\_hot(Y)
2. **print**(Y\_oh[0])
3. array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])

6. Реализуйте функцию стоимости для данной нейронной сети.

Функция стоимости:

1. **def** J(thetas, X, Y\_one\_hot, lmb=0.):
2. m = len(X)
3. h = forward\_prop\_vec(thetas, X)
5. e1 = np.multiply(Y\_one\_hot, np.log(h))
6. e2 = np.multiply((1 - Y\_one\_hot), np.log(1 - h))
8. regularization = 0
9. cost = (-1 / m) \* np.sum(e1 + e2)
11. # Не помню, нужно ли в регуляризации учитывать баес? В Лог регрессии - не нужно было
12. **if** lmb != 0:
13. reg\_sum = np.sum(np.power(rm\_bias(thetas[0]), 2)) + np.sum(np.power(rm\_bias(thetas[1]), 2))
14. regularization = (lmb / (2 \* m)) \* reg\_sum
16. **return** cost + regularization

Результат работы функции стоимости:

0.2876291651613189

7. Добавьте L2-регуляризацию в функцию стоимости.

Функция стоимости с регуляризацией L2:

0.3260854427331608

8. Реализуйте функцию вычисления производной для функции активации.

Функция производная от функции активации:

1. **def** der\_sigmoid(a):
2. **return** np.multiply(a, 1 - a)
4. **print**(der\_sigmoid(np.array([1, 2, 3])))
5. array([ 0, -2, -6])

9. Инициализируйте веса небольшими случайными числами..

Функция инициализации весов случайными числами:

1. **def** gen\_thetas(eps=0.1):
2. t1 = np.random.rand(nn\_params['layer\_1\_output'], nn\_params['layer\_1\_input']) \* (2 \* eps) - eps
3. t2 = np.random.rand(nn\_params['layer\_2\_output'], nn\_params['layer\_2\_input']) \* (2 \* eps) - eps
5. **return** np.array([t1, t2])

10. Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.

Функция обратного распространения ошибки:

1. **def** back\_prop\_vec(unrolled\_thetas, X, Y\_one\_hot, lmb=0.):
2. # print('call - back\_prop\_vec')
3. m = len(X)
4. thetas = rehape\_thetas(unrolled\_thetas)
6. delta\_1 = np.zeros(thetas[0].shape)
7. delta\_2 = np.zeros(thetas[1].shape)
9. **for** i **in** range(m):
10. fp\_data = forward\_prop\_vec\_all(thetas, X[i].reshape(1, -1))
12. d3 = fp\_data['a3'] - Y\_one\_hot[i]  # слой выхода
14. d2 = rm\_bias(np.multiply(np.dot(d3, thetas[1]), der\_sigmoid(fp\_data['a2'])))
16. delta\_1 += np.dot(d2.T, fp\_data['a1'])
17. delta\_2 += np.dot(d3.T, fp\_data['a2'])
19. delta\_1 /= m
20. delta\_2 /= m
22. **if** lmb != 0:
23. lmb\_mult = (lmb / m)
24. delta\_1[:, 1:] += lmb\_mult \* rm\_bias(thetas[0])
25. delta\_2[:, 1:] += lmb\_mult \* rm\_bias(thetas[1])
27. **return** unroll\_thetas(np.array([delta\_1, delta\_2]))

11. Для того, чтобы удостоверится в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром ε = 10-4.

Функция проверки градиента:

1. **def** gd\_check(experiments\_count, unrolled\_thetas, back\_prop\_thetas, X, Y\_one\_hot, lmb=0.):
2. eps = 0.0001
3. theta\_count = len(unrolled\_thetas)
5. **for** i **in** range(experiments\_count):
6. idx = int(np.random.rand() \* theta\_count)
8. experiment\_thetas = np.copy(unrolled\_thetas)
9. orig\_val = experiment\_thetas[idx]
11. experiment\_thetas[idx] = orig\_val + eps
12. cost\_plus = J\_unroll(experiment\_thetas, X, Y\_one\_hot, lmb)
14. experiment\_thetas[idx] = orig\_val - eps
15. cost\_minus = J\_unroll(experiment\_thetas, X, Y\_one\_hot, lmb)
17. calc\_g = (cost\_plus - cost\_minus) / (2 \* eps)
19. **print**(f'Idx: {idx} check gradient: {calc\_g:f}, BP gradient: {back\_prop\_thetas[idx]:f}')

Пример работы функции градиента:

gd\_check(5, unroll\_thetas(thetas), back\_prop\_thetas, X, Y\_oh, 0)

Idx: 4801 check gradient: 0.000002, BP gradient: 0.000002

Idx: 2015 check gradient: -0.000000, BP gradient: -0.000000

Idx: 2598 check gradient: -0.000219, BP gradient: -0.000219

Idx: 715 check gradient: -0.000083, BP gradient: -0.000083

Idx: 3469 check gradient: -0.000000, BP gradient: -0.000000

12. Добавьте L2-регуляризацию в процесс вычисления градиентов.

Функция вычисления градиента с L2-регуляризацией:

back\_prop\_thetas\_reg = back\_prop\_vec(unroll\_thetas(thetas), X, Y\_oh, 0.5)

13. Проверьте полученные значения градиента.

Результат проверки градиента с L2-регуляризацией:

gd\_check(5, unroll\_thetas(thetas), back\_prop\_thetas\_reg, X, Y\_oh, 0.5)

Idx: 5652 check gradient: -0.000001, BP gradient: -0.000001 Idx: 5867 check gradient: 0.000180, BP gradient: 0.000180 Idx: 2974 check gradient: 0.000173, BP gradient: 0.000173 Idx: 7214 check gradient: 0.000000, BP gradient: 0.000000 Idx: 6494 check gradient: -0.000074, BP gradient: -0.000074

14. Обучите нейронную сеть с использованием градиентного спуска или других более эффективных методов оптимизации.

Функция обучения нейросети:

1. **def** fit(X, Y, lmb=0., maxiter=30):
2. rand\_thetas = gen\_thetas()
3. unrolled\_thetas = unroll\_thetas(rand\_thetas)
5. Y\_one\_hot = one\_hot(Y)
7. # back\_prop\_vec(unroll\_thetas(thetas), X, Y\_oh, 0.5)
8. # result = optimize.minimize(J\_unroll, jac=back\_prop\_vec, x0=unrolled\_thetas, args=(X, Y\_one\_hot, lmb), method='BFGS', options={ 'maxiter': maxiter , 'disp': True})
9. # out\_thetas = rehape\_thetas(result.x)
10. result = optimize.fmin\_cg(maxiter=maxiter, f=J\_unroll, x0=unrolled\_thetas, fprime=back\_prop\_vec,
11. args=(X, Y\_one\_hot, lmb))
12. out\_thetas = rehape\_thetas(result)
14. **return** out\_thetas

15. Вычислите процент правильных классификаций на обучающей выборке.

Результат:

**print**("NN accuracy: %0.1f%%"%(100\*calc\_accuracy(fitted\_thetas, X, Y)))

Accuracy - 99.0% при 75 итерациях.

16. Визуализируйте скрытый слой обученной сети.



Рисунок 2 – визуализация скрытого слоя нейросети.

17. Подберите параметр регуляризации. Как меняются изображения на скрытом слое в зависимости от данного параметра?



Рисунок 3 – зависимость точности от параметра lambda.

Про рисунку 3 видно, что наибольшая точность получается при параметре lambda = 0.01, отсутствие регуляризации, или использование других коэффициентов lambda понижает точность модели.



Рисунок 4 – визуализация скрытого слоя при lambda=0.001.



Рисунок 5 – визуализация скрытого слоя при lambda=1.



Рисунок 6 – визуализация скрытого слоя при lambda=30.

Если сравнить визуализацию внутреннего слоя при разных lambda, можно заметить, что наиболее четкое изображение получается при lambda = 0.001. Если увеличить коэффициент, то изображение начинает смазываться и размываться. Пример тому визуализация скрытого слоя при lambda=30.

**Вывод.**

В ходе выполнения лабораторной работы я изучил принципы работы нейронной сети, реализовал алгоритмы: “forward propagation”, “back propagation” и алгоритм проверки корректности градиента.

Применив для классификации рукописных цифр нейронную сеть вместо логистической регрессии удалость повысить параметр accuracy на 1,8% и теперь он составляет 99.0%.