Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №6**

«Применение свёрточных нейронных сетей (многоклассовая классификация)»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2020

ХОД РАБОТЫ

**Задание.**

Набор данных для распознавания языка жестов, который состоит из изображений размерности 28x28 в оттенках серого (значение пикселя от 0 до 255). Каждое из изображений обозначает букву латинского алфавита, обозначенную с помощью жеста, как показано на рисунке ниже (рисунок цветной, а изображения в наборе данных в оттенках серого). Обучающая выборка включает в себя 27,455 изображений, а контрольная выборка содержит 7172 изображения. Данные в виде csv-файлов можно скачать на сайте Kaggle -> https://www.kaggle.com/datamunge/sign-language-mnist

1. Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.
2. Реализуйте глубокую нейронную сеть со сверточными слоями. Какое качество классификации получено? Какая архитектура сети была использована?
3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?
4. Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Можно ли было обойтись без него? Какой максимальный результат удалось получить на контрольной выборке?

**Результат выполнения:**

1. Код реализации:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex1data1.csv')

data\_frames = pd.read\_csv(file\_path)

x = data\_frames['population']

y = data\_frames['profit']

x = list(x) # np.array(x)

y = list(y)

2. Код реализации:

def compute\_cost(X, Y, theta):

m = len(X)

diff = []

for i in range(0, m):

val = pow(h0x(X[i], theta) - Y[i], 2)

diff.append(val)

cost = (1 / (2 \* m)) \* sum(diff)

return cost

3. Код реализации:

def gradient\_descent(X, Y, theta, iterations, alpha):

"""

From Andrew Ng implementation: without ones vector in X

"""

m = len(X)

J = []

for i in range(iterations):

val = np.zeros(len(theta))

for j in range(0, m):

val[0] += h0x(X[j], theta) - Y[j]

for k in range(1, len(theta)):

val[k] += (h0x(X[j], theta) - Y[j]) \* X[j]

for z in range(0, len(theta)):

theta[z] = theta[z] - (alpha / m) \* val[z]

J.append(compute\_cost(X, Y, theta))

return [theta, J]

4. Код реализации:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex1data2.csv')

data = pd.read\_csv(file\_path)

**Программный код:**

from \_\_future\_\_ import division

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import os

import time

print("BSUIR: Machine Learning, L1")

def h0x(x\_value, theta):

return theta[0] + theta[1] \* x\_value

def compute\_cost(X, Y, theta):

m = len(X)

diff = []

for i in range(0, m):

val = pow(h0x(X[i], theta) - Y[i], 2)

diff.append(val)

cost = (1 / (2 \* m)) \* sum(diff)

return cost

def gradient\_descent(X, Y, theta, iterations, alpha):

"""

From Andrew Ng implementation: without ones vector in X

"""

m = len(X)

J = []

for i in range(iterations):

val = np.zeros(len(theta))

for j in range(0, m):

val[0] += h0x(X[j], theta) - Y[j]

for k in range(1, len(theta)):

val[k] += (h0x(X[j], theta) - Y[j]) \* X[j]

for z in range(0, len(theta)):

theta[z] = theta[z] - (alpha / m) \* val[z]

J.append(compute\_cost(X, Y, theta))

return [theta, J]

def gradient\_descent\_not\_vectorized(X, Y, theta, iterations, alpha):

"""

For reducing impact of compute\_cost\_vectorized vs compute\_cost

"""

m = len(X)

J = []

for i in range(iterations):

val = np.zeros(len(theta))

for j in range(0, m):

for k in range(0, len(theta)):

val[k] += (h0x\_vectorized(X[j], theta) - Y[j]) \* X[j][k]

for z in range(0, len(theta)):

theta[z] = theta[z] - (alpha / m) \* val[z]

J.append(compute\_cost\_vectorized(X, Y, theta))

return [theta, J]

def h0x\_vectorized(X, theta):

return X.dot(theta)

def compute\_cost\_vectorized(X, Y, theta):

# J = (1 / (2 \* m)) \* (X \* theta - y)' \* (X \* theta - y); % equally (sum(power(X, 2)))

m = len(X)

temp = (h0x\_vectorized(X, theta) - Y)

return (1 / (2 \* m)) \* np.dot(temp.T, temp)[0][0]

def gradient\_descent\_vectorized(X, Y, theta, iterations, alpha):

m = len(Y)

J\_history = []

for i in range(iterations):

# theta = theta - alpha \* (1/m) \* (((X\*theta) - y)' \* X)'; % Vectorized

h0x = (h0x\_vectorized(X, theta) - Y).T

dt = np.dot(h0x, X).T

a = alpha \* (1 / m) \* dt

theta = theta - a

J\_history.append(compute\_cost\_vectorized(X, Y, theta))

return [theta, J\_history]

def feature\_normalization(X):

X = X.T

for i in range(1, len(X)):

mu = np.mean(X[i])

s = np.std(X[i], ddof=1) # TODO: learn more about ddof

X[i] = (X[i] - mu) / s

return X.T

def normal\_eqn(X, Y):

# theta = pinv(X' \* X) \* (X' \* y); % Vectorized

return np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T, Y))

if \_\_name\_\_ == '\_\_main\_\_':

# 1

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex1data1.csv')

data\_frames = pd.read\_csv(file\_path)

x = data\_frames['population']

y = data\_frames['profit']

x = list(x) # np.array(x)

y = list(y)

# 2

fig, ax = plt.subplots()

ax.scatter(x, y)

plt.show()

# 3

theta = [0, 0]

print('With theta = [0 ; 0]\nCost computed: ', compute\_cost(x, y, theta))

print('Expected cost value (approx) 32.07\n')

theta = [-1, 2]

print('\nWith theta = [-1 ; 2]\nCost computed: ', compute\_cost(x, y, theta))

print('Expected cost value (approx) 54.24\n')

# 4

iterations = 1500

alpha = 0.01

print('Running Gradient Descent ...\n')

# run gradient descent

theta = [0, 0]

[theta, J1] = gradient\_descent(x, y, theta, iterations, alpha)

print('Theta found by gradient descent:', theta)

print("Cost: ", compute\_cost(x, y, theta))

print('Expected theta values (approx): -3.6303 1.1664\n\n')

ax.plot([4, 23], [h0x(0, theta), h0x(23, theta)], 'red')

plt.show()

# 5

u = np.arange(-5, 5, 0.1)

v = np.arange(-5, 5, 0.1)

z = np.zeros((len(u), len(v)))

for i in range(len(u)):

for j in range(len(u)):

z[i][j] = compute\_cost(x, y, [u[i], v[j]])

u, v = np.meshgrid(u, v)

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(u, v, z, linewidth=0, antialiased=False)

plt.show()

fig, ax = plt.subplots()

plt.contour(u, v, z, np.logspace(-2, 3, 20))

plt.show()

# 6

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex1data2.csv')

data = pd.read\_csv(file\_path)

# 7-8

X = data.iloc[:, 0:2] # read first two columns into X

Y = data.iloc[:, 2] # read the third column into y

m = len(Y)

ones = np.ones((m, 1))

X = np.hstack((ones, X)) # [x1, x2] => [1, x1, x2]

theta = np.zeros((3, 1))

Y = Y[:, np.newaxis] # convert to a matrix

print('With theta = [0; 0; 0]\nCost computed: ', compute\_cost\_vectorized(X, Y, theta))

print('Expected cost value (approx) 65591548106.45744\n')

print('Without normalization: \n')

# Can not be calculated since numbers are too large

# print(gradient\_descent\_vectorized(X, Y, theta, iterations, alpha)[0])

print('With normalization: \n')

X = feature\_normalization(X)

start1 = time.time()

[theta, J1] = gradient\_descent\_not\_vectorized(X, Y, theta, iterations, alpha)

end1 = time.time()

print('Time gradient not vectorized: ', end1 - start1, theta)

start2 = time.time()

[gdv, J] = gradient\_descent\_vectorized(X, Y, np.zeros((3, 1)), iterations, alpha)

end2 = time.time()

print('Vectorized time: ', end2 - start2)

print('Solution: ')

print(gdv)

# 9

objects = ('Vectorized', 'Not-Vectorized')

y\_pos = np.arange(len(objects))

performance = [end2-start2, end1-start1]

plt.bar(y\_pos, performance, align='center', alpha=0.5)

plt.xticks(y\_pos, objects)

plt.ylabel('Time (s)')

plt.title('Performance')

plt.show()

# 10

year = range(0, iterations)

plt.plot(year, J, color='orange')

plt.xlabel('Iterations')

plt.ylabel('Cost J')

plt.title('Cost by iterations')

plt.show()

# 11

theta = normal\_eqn(X, Y)

print('Normal equation: \n')

print(theta)