Artificial Intelligence Miniproject
Perceptron learning

Jannick Drews

May 3, 2020

1 Dataset & modifications

The dataset used in this project is from the Kaggle website: `https://www.kaggle.com/spscientist/students-performance-in-exams`
I used this dataset, converted it into 4 different categories which I feed into my network. These 4 categories are;

- Gender
- Test preparation
- Parents educational degree (> associates degree)
- If the student ate lunch

The student would need a score > 65 out of a 100 to pass.

2 code

```
1  import numpy as np
2  import csv
3
4
5  gender = [] #0
6  race = [] #1
7  parental = [] #2
8  lunch = [] #3
9  test_prep = [] #4
10 math_score = [] #5
11 read_score = [] #6
12 write_score = [] #7
13
14 gender_dict = ["female", "male"]
15 test_prep_dict = ["none", "completed"]
16
17 with open('StudentsPerformance.csv') as csvfile:
18     csvread = csv.reader(csvfile)
19     for row in csvread:
20         gender.append(row[0])
21         if not row[0] in gender_dict:
```

```
            gender_dict.append(row[0])
        race.append(row[1])
        parental.append(row[2])
        lunch.append(row[3])
        test_prep.append(row[4])
        if not row[4] in test_prep_dict:
            test_prep_dict.append(row[4])
        math_score.append(row[5])
        read_score.append(row[6])
        write_score.append(row[7])

def parent_dec(arg):
    return (
            arg == "bachelor's degree"
            or arg == "master's degree"
            or arg == "associate's degree")

def gender_dec(arg):
    return arg=="female"

def score_dec(arg):
    return int(arg) > 65

def lunch_dec(arg):
    return arg == "standard"

train_set = np.zeros((1000, 4), dtype=int)
train_result = np.zeros((1000, 1), dtype=int)

for i in range(0, 1000):
    train_set[i][0] = int(gender_dict.index(gender[i]))
    train_set[i][1] = test_prep_dict.index(test_prep[i])
    train_set[i][2] = parent_dec(parental[i])
    train_set[i][3] = lunch_dec(lunch[i])
    train_result[i] = score_dec(math_score[i])

#DATA HANDLING DONE


# PERCEPTRON LEARNING
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_der(x):
    return sigmoid(x) * (1 - sigmoid(x))

weights = np.random.rand(4,1)
bias = np.random.rand(1)
learnrate = 0.0005

for epoch in range(2500):
    inputs = train_set

    # feed
    sum = np.dot(train_set, weights) + bias
    sig_sum = sigmoid(sum)

    # backprop
    err_marg = sig_sum - train_result
    if (epoch % 500 == 1):
        print(err_marg.sum())

    sigged = err_marg * sigmoid_der(sig_sum)
    weights -= learnrate * np.dot(train_set.T, sigged)

    for x in sigged:
        bias -= learnrate * x

```

```
90  # TESTING
91
92
93  while(1):
94      userinput = np.array([0,0,0,0])
95      userinput[0] = input("Gender: 1 = Female\n")
96      userinput[1] = input("Did the student prepare? 1 = Yes\n")
97      userinput[2] = input("Does the students parents have educational degree?
            1 = Yes\n")
98      userinput[3] = input("Did the student eat lunch? 1 = Yes\n")
99
100     result = sigmoid(np.dot(userinput, weights) + bias)
101     print("\nStudent probability of passing: ", result[0]*100, "%\n\n")
```

Up until line 60, the code is merely adjusting the needed data for the AI to handle. This includes reading from the CSV file and appending it properly to the needed input data. Getting tthe data for whether the student have had lunch, parental education, if they prepared and their gender.

Two functions are declared at line 62 and 65. These are the sigmoid and sigmoid derivative functions used for the feedforward and backpropagation.
The weights are initialized randomly as well as the bias at lines 68 and 69.
Lines 72 to 88 contains the learning for the network. This is done in a loop to 2500 for learning with epochs.
Lines 76 to 77 contains the feedforward instructions. With taking the sum of the dot products of the set with its weights, with the bias added afterwards. This is then put into the sigmoid function for the output.
Lines 80 to 84 contains the backpropagation. Finding the error margin compared to the real result to see how far off the guess was. Lines 85 and 88 then adjusts the bias and weights of the network before we try again in the next loop.