07.01.2021

BİM470 NEURAL NETWORK PROJECT

Subject: Creating A Convolutional Neural Network Architecture

MEHMET ZAHİT ANGİ

The CNN code contains one convolution layer, one maxpooling layer, a flattening layer and one fully connected layer. The transfer function of the output layer is softmax function.

There are five different filters.

The fundamentals of CNN architecture are below the pdf. Other details added to code document as comment lines.

FIRST PART

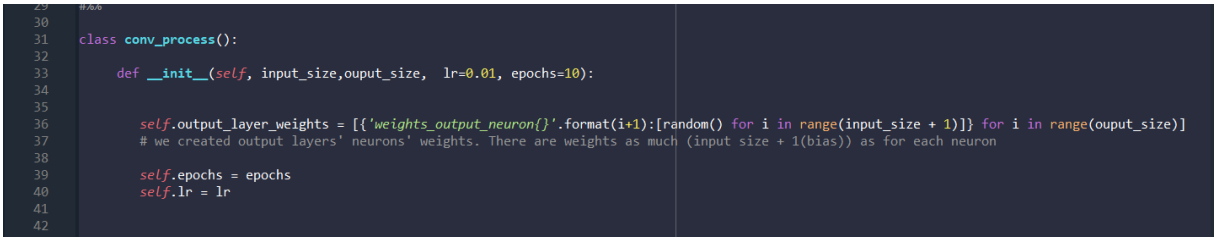
Read and Split The Data

```
# -*- coding: utf-8 -*-
"""
Created on Sun Dec 27 17:49:43 2020

@author: mehme
"""

import numpy as np
from numpy import unravel_index

from numpy import genfromtxt
my_data = genfromtxt('train.csv', delimiter=',') # gets first row that is empty of csv
#%%  pictures has 28*28 Dimension
from random import random
data = np.delete(my_data,0,0) # deleted the get empty row, now there are just datas

rrr = data[20000:21000]


train_data = data[0:100]
train_data_y = train_data[:,0]  # data[row][coloumn] or data[row,column]
train_data_features = train_data[:,1:]


test_data = data[101:111]
test_data_y = test_data[:,0]
test_data_features = test_data[:,1:]
```

On the top added libraries.

I took the first 100 data for train_data and took just 10 data for test_data. That can be change inside the code. I took less data for easy computing.

All forwarding processes doing in “class conv_process” but backward processes is not in this class except flatten_to_maxpooling and filter_updated functions. Also train and prediction parts are in this class.

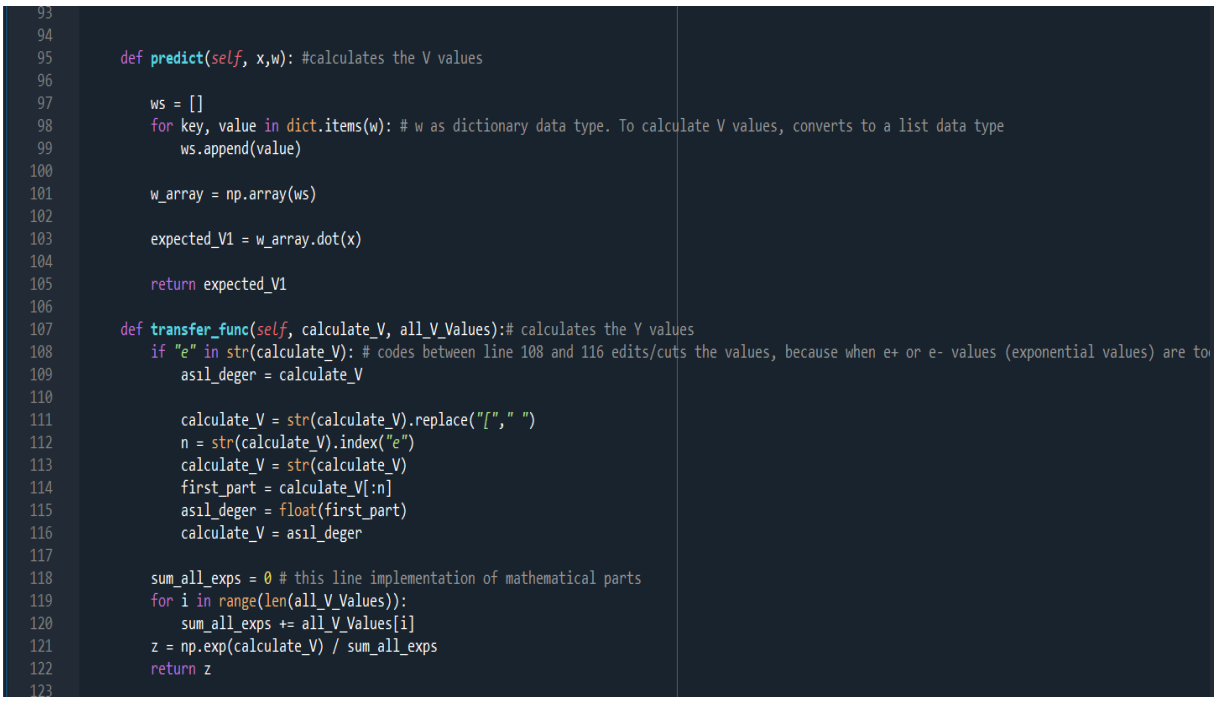


```
#%%

class conv_process():

    def __init__(self, input_size,ouput_size,  lr=0.01, epochs=10):


        self.output_layer_weights = [{'weights_output_neuron{}'.format(i+1):[random() for i in range(input_size + 1)]} for i in range(ouput_size)]
        # we created output layers' neurons' weights. There are weights as much (input size + 1(bias)) as for each neuron

        self.epochs = epochs
        self.lr = lr


```

On init function, the program takes some necessary inputs that will use after processes. Those are input size, output size, learning rate and epochs number.

Def function details are in code document because of this part is too long and screen shots are too small. In shorty, all calculate processes are in this funciton like find V values, y values, update weights, local gradients and update flatten layer.

Predict V values and predict y values using transfer function.(Details on code documents as comment lines)



```


    def predict(self, x,w): #calculates the V values

        ws = []
        for key, value in dict.items(w): # w as dictionary data type. To calculate V values, converts to a list data type
            ws.append(value)

        w_array = np.array(ws)

        expected_V1 = w_array.dot(x)

        return expected_V1

    def transfer_func(self, calculate_V, all_V_Values):# calculates the Y values
        if "e" in str(calculate_V): # codes between line 108 and 116 edits/cuts the values, because when e+ or e- values (exponential values) are to
            asıl_deger = calculate_V

            calculate_V = str(calculate_V).replace("[","")
            n = str(calculate_V).index("e")
            calculate_V = str(calculate_V)
            first_part = calculate_V[:n]
            asıl_deger = float(first_part)
            calculate_V = asıl_deger

        sum_all_exps = 0 # this line implementation of mathematical parts
        for i in range(len(all_V_Values)):
            sum_all_exps += all_V_Values[i]
        z = np.exp(calculate_V) / sum_all_exps
        return z


```

Derivetive_softmax calculates the derivatives of softmax function with an matrix and Local_gradients function calculates the local gradients.

```
    def derivative_softmax(self,s):
        matris_array = []
        for i in range(len(s)):
            matris_array.append(float(s[i]))

        jacobian_m = np.diag(matris_array) # normalde deneme değil s

        for i in range(len(jacobian_m)): # created a matrix and calculate softmax derivatives
            for j in range(len(jacobian_m)):

                if i == j:
                    jacobian_m[i][j] = s[i] * (1 - s[i])
                else:
                    jacobian_m[i][j] = -s[i] * s[j]
        return jacobian_m


    def local_gradients(self,y_correct,derivatives): # calculates the local gradients
        first_part = -1/y_correct
        second_part = round(derivatives,2)
        local_gradient = first_part * second_part
        return local_gradient

```

Some backwards function

```
148         def update_flatten(self,output_layer_weights,local_gradients):# updating flatten layer on backward
149             flatten_updated = []
150 
151             for i in range(flatten_layer.shape[0]):
152                 summ = 0
153                 for j in range(len(self.output_layer_weights)):
154                     temporal_array = []
155                     for k in range(len(self.output_layer_weights)):
156                         for key, value in dict.items(self.output_layer_weights[k]):
157                             temporal_array.append(value)
158 
159                     temporal_array = np.array(temporal_array)
160 
161                     summ += temporal_array[j][i]*local_gradients[j]
162 
163                 flatten_updated.append(summ)
164 
165             return flatten_updated
166 
167 
168         def flatten_to_maxpooling(flatten_layer): # This function assign flatten layer values to max poolings
169             pooling_groups = []
170 
171             as_numpy = np.array(flatten_layer)
172             as_numpy = np.split(as_numpy, 5) # 5 is filter number
173             pooling_groups.append(as_numpy)
174 
175         def filter_updated(self,filter_name,input_normal,gradient_conv):
176 
177             stride_length = len(gradient_conv)
178             h2, w2 = input_normal.shape
179 
180             backward_convaluated_input = np.empty([3, 3]) # delta_filter
181 
182             for i in range(h2 - stride_length):
183                for j in range(w2 - stride_length):
184                     region = input_normal[i:(i + stride_length), j:(j + stride_length)]
185                     #delta_filter = region*gradient_conv
186                     #delta_filter = np.append (input_normal[i, j], region*gradient_conv)  #region*gradient_conv
187                     np.append (backward_convaluated_input[i, j], region*gradient_conv)
188 
189                     #np.append (conv_filter1[i, j], region*gradient_conv)
190 
191             filter_name = filter_name + self.lr*backward_convaluated_input#delta filter
192             return filter_name
193 
```

Other backwards processes is in codes as linearly. They are in main class.

Test_func has same parts with fit function. I just wrote necessary parts for test.

```
193 
194     def test_func(self,flatten_layer):
195         for i in range(1):
196                x = np.insert(flatten_layer, 0, 1)
197                x.reshape(len(flatten_layer)+1,1)
198 
199                ys = []
200 
201                Vs = []
202 
203                for i in range(len(self.output_layer_weights)):
204 
205                    Vs.append(self.predict(x,self.output_layer_weights[i]) )
206 
207                for i in range(len(self.output_layer_weights)):
208 
209                    ys.append(self.transfer_func(Vs[i],Vs))
210         return ys
211 
212 
```

MAIN PARTS

The initialize filters that I used :

```
sharpen = np.array([[0, -1, 0],[-1, 5, -1],[0, -1, 0]])
blur = np.array([[1,1,1],[1,1,1],[1,1,1,]])
edge_enhance = np.array([[0,0,0],[-1,1,0],[0,0,0]])
edge_detect = np.array([[0, 1, 0],[1,-4,1],[0,1,0]])
emboss = np.array([[-2, -1, 0],[-1,1,1],[0,1,2]])
```

I created a for loop to training. That is have range of lenght of train data set.

All convolution and max pooling steps are done inside this for loop.

All other parts using functions that above the PDF , runs in the loop.

And as same as the codes that make test are in main class.

Codes are too lenght in main class, so I didn't add images of this parts.

DATASET

For this report, I used first 100 data for training and 10 data betweern row 101 and row 111 for test because of time of computing. This numbers can be chaned on code documents. All the way, all processes will work correctyl.

And test results in other page.

For row 101 .The program predicted the results is 5. (Highest predicted probabilty is 5.)

Predict results : [array([0.33711337]), array([0.42983617]), array([0.4111792]), array([0.20435373]), array([0.21279256]), array([0.43571578]), array([0.32135268]), array([0.33407145]), array([0.19271308]), array([0.40269653])]

For row 102 .The program predicted the results is 3 . (Highest predicted probabilty is 3.)

Predict results : [array([0.3145761]), array([0.36272015]), array([0.21433088]), array([0.52716663]), array([0.36111603]), array([0.36924994]), array([0.30459617]), array([0.26269291]), array([0.37773103]), array([0.26876836])]

For row 103 .The program predicted the results is 3 . (Highest predicted probabilty is 3.)

Predict results : [array([0.36977316]), array([0.20637809]), array([0.40106578]), array([0.42616807]), array([0.27738413]), array([0.21089154]), array([0.36262613]), array([0.32718189]), array([0.23322606]), array([0.30353903])]

For row 104 .The program predicted the results is 1 . (Highest predicted probabilty is 1.)

Predict results : [array([0.28592825]), array([0.59452225]), array([0.28191037]), array([0.54384635]), array([0.30492153]), array([0.47124672]), array([0.51081872]), array([0.37912766]), array([0.42773589]), array([0.27206879])]

For row 105 .The program predicted the results is 9 . (Highest predicted probabilty is 9.)

Predict results : [array([0.51829253]), array([0.24700232]), array([0.39936159]), array([0.22601015]), array([0.26896002]), array([0.25276244]), array([0.28754791]), array([0.33335537]), array([0.51671838]), array([0.52519315])]

For row 106 .The program predicted the results is 0 . (Highest predicted probabilty is 0.)

Predict results : [array([0.4677589]), array([0.21085336]), array([0.32555227]), array([0.27638038]), array([0.45550465]), array([0.41024774]), array([0.23881372]), array([2932.19561954]), array([0.30361763]), array([0.37362654])]

For row 107 .The program predicted the results is 9 . (Highest predicted probabilty is 9.)

Predict results : [array([0.36194853]), array([0.22778814]), array([0.51889577]), array([0.22134049]), array([0.38973539]), array([0.41665691]), array([0.22823251]), array([0.32667911]), array([0.38904794]), array([0.49499668])]

For row 108 .The program predicted the results is 8 . (Highest predicted probabilty is 8.)

Predict results : [array([0.50725264]), array([0.35958277]), array([0.30364066]), array([0.41823627]), array([0.23025073]), array([0.24000868]), array([0.19785229]), array([0.49332649]), array([0.52874853]), array([0.20719495])]

For row 109 .The program predicted the results is 1 . (Highest predicted probabilty is 1.)

Predict results : [array([0.37772485]), array([0.38731187]), array([0.36283816]), array([0.30585527]), array([0.16682728]), array([0.36441087]), array([0.32717949]), array([0.37818769]), array([0.21222747]), array([0.17739323])]

For row 110 .The program predicted the results is 7 . (Highest predicted probabilty is 7.)

Predict results : [array([0.43354776]), array([0.25741666]), array([0.59379469]), array([0.32012358]), array([0.26679964]), array([0.34302307]), array([0.24905005]), array([0.57491263]), array([0.29914141]), array([0.47787453])]