Deekshith Krishnegowda

deekshithkrishnegowda@yahoo.com

Miniproject3

Inference of neural network

Contents

[MiniProject3: Inference engine for neural network 2](#_Toc529819558)

[Aim: 2](#_Toc529819559)

[What does neural network do: 3](#_Toc529819560)

[Information about this neural net: 4](#_Toc529819561)

[General Information: 4](#_Toc529819562)

[Activation functions: 4](#_Toc529819563)

[ReLu function: 4](#_Toc529819564)

[Sigmoid function: 7](#_Toc529819565)

[Results: 8](#_Toc529819566)

[Python Code: 9](#_Toc529819567)

# MiniProject3: Inference engine for neural network

## Aim:

In this project we are attempting to implement Convolutional Neural Networks. This exercise is previously done in MATLAB. The trained weights are extracted from running the MATLAB code and the inputs are taken from the MATLAB code as well. The neuron activation point is result of sigmoid or ReLu function which is defined in the python code.

## What does neural network do:

These networks have a certain fixed set of input data and they train over them to find out what weights do they need to come to a certain output value. Finding weights is an iterative process and needs higher level languages such as python, or a MATLAB package. Another set of data is kept completely clean from the training set and its used only for testing purposes. These parameters given to us were yielding 95% accuracy which is pretty good. If we made pushed for more accuracy, we might fall into the trap of over fitting.

## Information about this neural net:

### General Information:

The inputs are taken from the MATLAB code which contains around 5000 datasets. The neural back propagation is implemented in the MATLAB code and the trained weights are taken it as well. The input image is 20x20 pixel image which is given to the input neurons of this neural net. There are 400 input neurons, 25 hidden layer neurons and 10 output neurons. There is bias in the hidden layer and input layer.

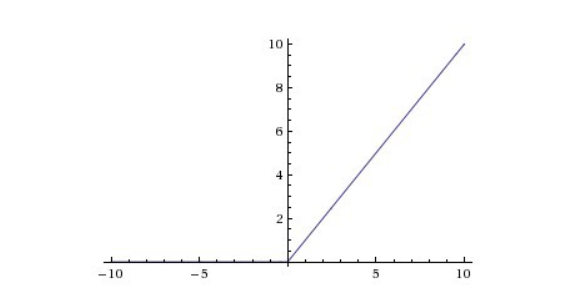


### Activation functions:

### ReLu function:

A(x) = max (0, x)

The ReLu function is as shown above. It gives an output x if x is positive and 0 otherwise.



At first look this would look like having the same problems of linear function, as it is linear in positive axis. First, ReLu is nonlinear in nature. And combinations of ReLu are also nonlinear! (in fact, it is a good approximator. Any function can be approximated with combinations of ReLu). Great, so this means we can stack layers. It is not bound though. The range of ReLu is [0, inf). This means it can blow up the activation.

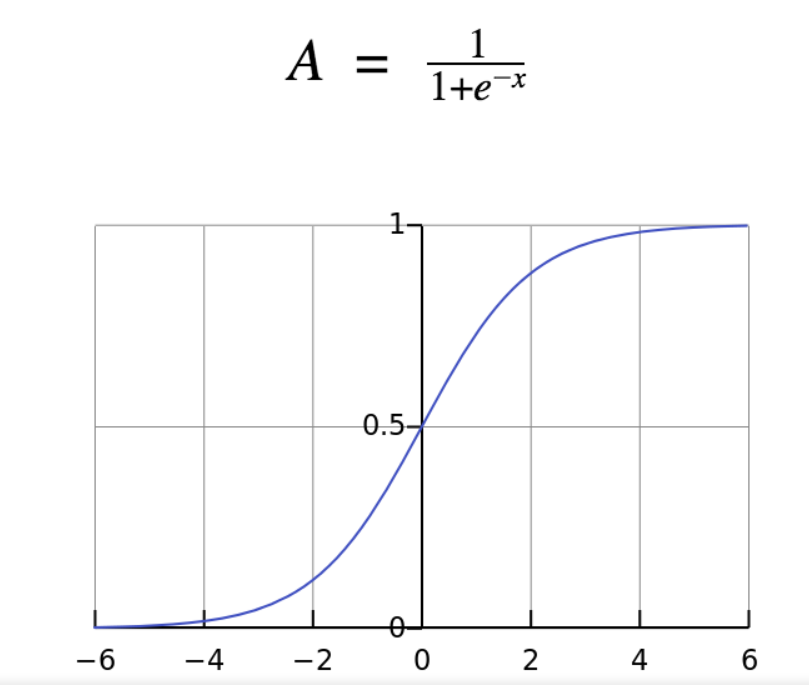
Another point that I would like to discuss here is the sparsity of the activation. Imagine a big neural network with a lot of neurons. Using a sigmoid or tanh will cause almost all neurons to fire in an analog way. That means almost all activations will be processed to describe the output of a network. In other words, the activation is dense. This is costly. We would ideally want a few neurons in the network to not activate and thereby making the activations sparse and efficient.

ReLu give us this benefit. Imagine a network with random initialized weights ( or normalized ) and almost 50% of the network yields 0 activation because of the characteristic of ReLu ( output 0 for negative values of x ). This means a fewer neurons are firing (sparse activation) and the network is lighter.

Because of the horizontal line in ReLu(for negative X ), the gradient can go towards 0. For activations in that region of ReLu, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). This is called dying ReLu problem. This problem can cause several neurons to just die and not respond making a substantial part of the network passive. There are variations in ReLu to mitigate this issue by simply making the horizontal line into non-horizontal component. for example, y = 0.01x for x<0 will make it a slightly inclined line rather than horizontal line. This is leaky ReLu. There are other variations too. The main idea is to let the gradient be non zero and recover during training eventually.

ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

### Sigmoid function:



The sigmoid function gives an ‘S’ shaped curve

This curve has a finite limit of:

‘0’ as x approaches −∞

‘1’ as x approaches +∞

The output of sigmoid function when x=0 is 0.5

## Results:

The final output activation point of neural network is the maximum function of 10 neurons activation point in the output layer. Printing all 10 activation points for 5000 datasets will take a lot of time and space. Therefore, the results are stored in the excel sheets provided in the submission directory. There are two result files, one for ReLu function and another for Sigmoid function. The result is maximum function of output neurons.

## Python Code:

'''

This file reads the input pixel values, trained weights from the excel file

and will do MAC to get activation point of each neuron. The MAC is followed by

a ReLu fuction or sigmoid function. The trained weights and input pixels are obtained

from the MATLAB training code.

There are 401 input neurons along with 1 bias. 26 neurons in the middle layer along with

1 bias and 10 output neurons.

'''

import csv

import re

import numpy as np

import pandas as pd

from math import exp

list1\_80=[]

list2\_80=[]

list4\_80=[]

#reading the first layer weights from excel file

with open('Theta1.csv', encoding='utf-8-sig') as file1:

reader=csv.reader(file1)

for row in reader:

list1\_80.append(row)

#file1.close()

#reading the output layer weights from excel file

with open('Theta2.csv', encoding='utf-8-sig') as file3:

reader=csv.reader(file3)

for row in reader:

list4\_80.append(row)

#file3.close()

#reading the inputs

with open('X\_value.csv', encoding='utf-8-sig') as file2:

reader=csv.reader(file2)

for row in reader:

list2\_80.append(row)

#file2.close()

#sigmoid function

def sigmoid(num\_80):

num\_80=float(num\_80)

deno\_80=1+exp(-num\_80)

ans\_80=1/deno\_80

return ans\_80

count=0

for i in list2\_80:

i.insert(0,1)

list3\_80=[]

#relu function

def reLu(num\_80):

if num\_80<0:

return 0

else:

return num\_80

#MAC for hidden layer

for m in list2\_80:

temp\_list\_80=[]

for n in list1\_80:

c=[float(a)\*float(b) for a,b in zip(n,m)]

sum=0

#print(c)

for i in c:

sum=sum+i

# print(sum)

ans\_80=sigmoid(sum)

temp\_list\_80.append(sum)

#print(temp\_list\_80)

temp\_list\_80.insert(0,1)

#print(temp\_list\_80)

list3\_80.append(temp\_list\_80)

# print(len(list3\_80))

# print(len(list3\_80[0]))

#MAC for output layer

list5\_80=[]

for m in list3\_80:

temp\_list\_80=[]

for n in list4\_80:

c=[float(a)\*float(b) for a,b in zip(n,m)]

sum=0

for i in c:

sum=sum+i

# print(sum)

ans\_80=sigmoid(sum)

temp\_list\_80.append(sum)

list5\_80.append(temp\_list\_80)

# #print('we ARE HERE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

# with open('reLu.csv', 'w',newline='') as f1:

# for rows in list5\_80:

# writer = csv.writer(f1)

# writer.writerow(rows)

# #print('write completed')

# f1.close()

#print((list5\_80[0]))

final\_list\_80=[]

for i in list5\_80:

t\_list\_80=i.copy()

sorted=False

while not sorted:

sorted=True

for j in range(len(t\_list\_80)-1):

if (t\_list\_80[j]>t\_list\_80[j+1]):

sorted=False

temp=t\_list\_80[j]

t\_list\_80[j]=t\_list\_80[j+1]

t\_list\_80[j+1]=temp

c=t\_list\_80[-1]

for k in range(len(i)):

if i[k]==c:

final\_list\_80.append(k)

break

with open('sigmoid.csv', 'w',newline='') as f1:

writer = csv.writer(f1)

writer.writerow(final\_list\_80)

f1.close()

print(final\_list\_80)

print(len(final\_list\_80))