Project Report

Project 1: **Sorting Neural Network**

By *Josias Moukpe* and *Michael Hon*

Introduction to AI

CSE 5290/4301

2-26-2019

GitHub> https://github.com/ERUD1T3/Sorting-Perceptron

Description

Given the task to sort 10 single digits in range of 0 - 10, we decided to architect the neural network according to the following diagram:

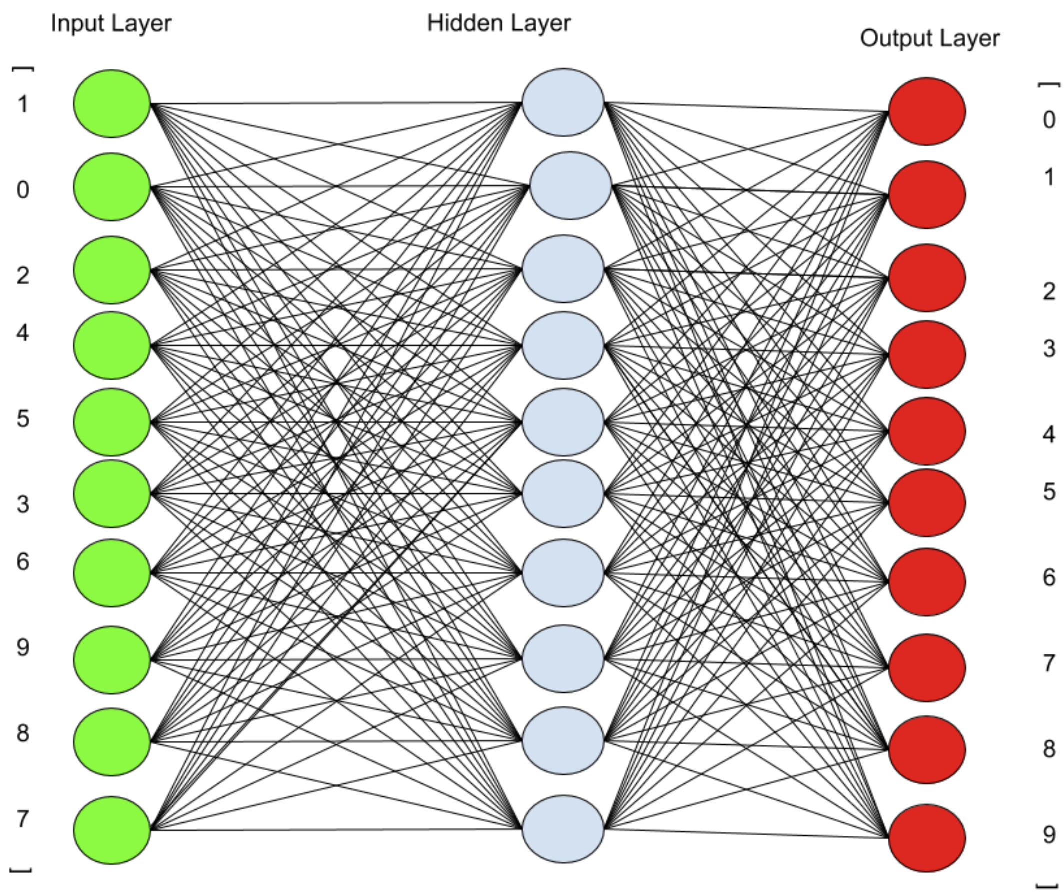


Fig 1. Diagram of the neural network with 10 neurons per layers and 3 layers

Each input digit in the unsorted list was mapped to the input neurons with random weights associated with them on the input layer. The input layer is then fully connected to the hidden layer which possesses 10 neurons as well.

Architecture

Following is the source code for gathering the data from the dataset, building the neural network and predicting on a sample data from the dataset.
The loss function used is Mean Square Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

The activation function used is Logistic Sigmoid, since its derivative is straightforward to compute.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

$$\begin{aligned}
g'_{\text{logistic}}(z) &= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)\\
&= \frac{e^{-z}}{(1+e^{-z})^2}(\text{chain rule})\\
&= \frac{1+e^{-z}-1}{(1+e^{-z})^2}\\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - \left(\frac{1}{1+e^{-z}}\right)^2\\
&= \frac{1}{(1+e^{-z})} - \left(\frac{1}{1+e^{-z}}\right)^2\\
&= g_{\text{logistic}}(z) - g_{\text{logistic}}(z)^2\\
&= g_{\text{logistic}}(z)(1 - g_{\text{logistic}}(z))
\end{aligned}$$

```
#!/home/erud1t3/anaconda3/bin/python

import torch #for torch tensors
```

```
import torch.nn as nn #for torch neural networks


N_EPOCHS = 100000
DATASET_SIZE = 100000

def readDataset(filepath):
   Tensor_Input = []
   Tensor_Output = []
   with open(filepath, 'r') as file:
       counter = 0
       for line in file:

           counter += 1
           if counter > DATASET_SIZE: break # determines how much data to
train upon

           # lines below process the data
           replaced = line.replace('[', '').replace(',', '').replace(' ',
'').replace(']','').replace('\n', '')
           split = replaced.split(';')
           inputline = split[0]
           outputline = split[1]

           temp_input_array = []
           for char in inputline: temp_input_array.append(int(char))
           Tensor_Input.append(temp_input_array)

           temp_output_array = []
           for char in outputline: temp_output_array.append(int(char))
           Tensor_Output.append(temp_output_array)

   return Tensor_Input, Tensor_Output



class Neural_Network(nn.Module):
```

```
    def __init__(self, ):
        super(Neural_Network, self).__init__()

        # parameters
        self.inputSize = 10 # 10 for list of ten unsorted digits
        self.outputSize = 10 # 10 for a list of ten sorted digits
        self.hiddenSize = 10 # 10 for hidden layer of ten nodes

        # weights
        self.W1 = torch.randn(self.inputSize, self.hiddenSize) # 10 X 10
tensor
        self.W2 = torch.randn(self.hiddenSize, self.outputSize) # 10 X 10
tensor
        # print('W1 : ' + str(self.W1.size()))
        # print('W2 : ' + str(self.W2.size()))

    def forward(self, X):
        self.z = torch.matmul(X, self.W1) # 10 X 10 matrix product
        self.z2 = self.sigmoid(self.z) # sigmoid activation function
        self.z3 = torch.matmul(self.z2, self.W2)
        o = self.sigmoid(self.z3) # final activation function
        return o

    def sigmoid(self, s):
        return 1 / (1 + torch.exp(-s))

    def sigmoidPrime(self, s):
        return s * (1 - s)  # derivative of sigmoid

    def backward(self, X, y, o):
        '''
            Back propagation
            src:
https://medium.com/dair-ai/a-simple-neural-network-from-scratch-with-pytor
ch-and-google-colab-c7f3830618e0
        '''
        self.o_error = y - o # error in output
```

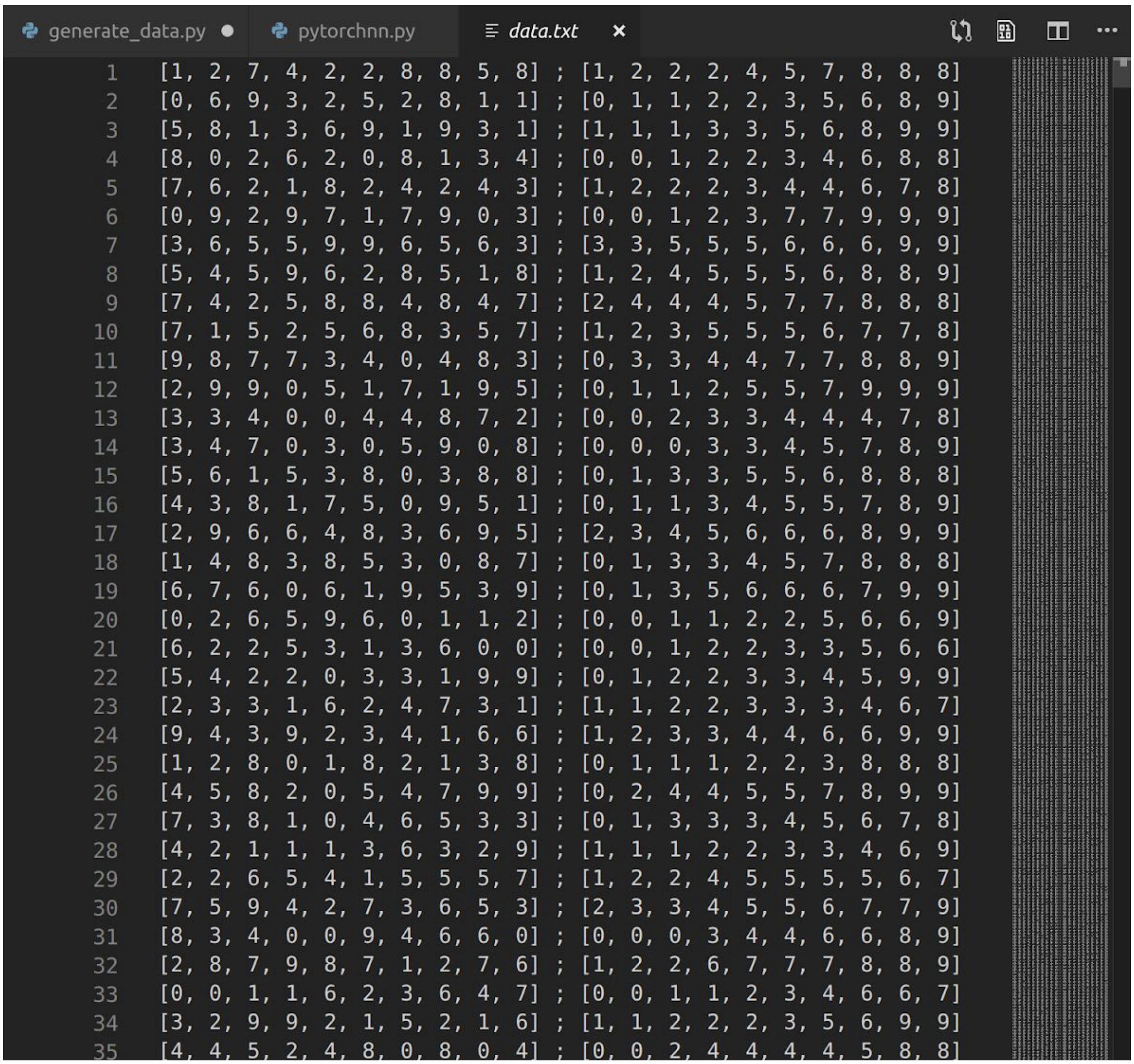
```
        self.o_delta = self.o_error * self.sigmoidPrime(o) # derivative of
sig to error
        self.z2_error = torch.matmul(self.o_delta, torch.t(self.W2))
        self.z2_delta = self.z2_error * self.sigmoidPrime(self.z2)
        self.W1 += torch.matmul(torch.t(X), self.z2_delta)
        self.W2 += torch.matmul(torch.t(self.z2), self.o_delta)

    def train(self, X, y):
        '''
            forward + backward pass for training
        '''
        o = self.forward(X)
        self.backward(X, y, o)

    def saveWeights(self, model):
        torch.save(model, "NN") # PyTorch internal storage functions
        # torch.load("NN") # you can reload model with all the weights and
so forth with:

    def predict(self):
        '''
            Predict data based on trained weights
        '''
        # xPredicted = torch.FloatTensor(xPredicted)
        print ("\nPredicted data based on trained weights: \n")
        print ("Input : \n" + str(xPredicted_unscaled))
        print ("Target output: \n" + str(target))
        # print ("Output: \n" + str(torch.round(9 *
self.forward(xPredicted))))
        print ("Output: \n" + str(torch.round(9 *
self.forward(xPredicted))))
        print ("Output (unrounded): \n" + str(9 *
self.forward(xPredicted)))

Tensor_Input, Tensor_Output = readDataset('./dataset/data.txt')
X = torch.FloatTensor(Tensor_Input) # 100000 x 10
y = torch.FloatTensor(Tensor_Output) # 100000 x 10
```

```
print(X.size())
print ('x : ' + str(X))
print(y.size())
print('y : ' + str(y))

# xPredicted_unscaled = xPredicted = X[2]
testlist = [9, 3, 5, 1, 0, 5, 2, 7, 8, 1] # test list to predict
# testlist = [0, 0, 1, 3, 6, 5, 1, 9, 2, 2]
targetlist = testlist.copy()
targetlist.sort()


xPredicted_unscaled = xPredicted = torch.FloatTensor(testlist)
# print(targettestlist)
target = torch.FloatTensor(targetlist)

X_max, _ = torch.max(X, 0)
xPredicted_max, _ = torch.max(xPredicted_unscaled, 0)

X = torch.div(X, X_max)
xPredicted = torch.div(xPredicted_unscaled, xPredicted_max)
y = y / 9  # max test score is 100

# print(y[0])

NN = Neural_Network()
for i in range(0, N_EPOCHS):  # trains the NN 1,000 times
   print ("Epoch " + str(i) + " | Loss: " + \
   str(torch.mean((y[i:10+i] - NN(X[i:10+i]))**2).detach().item()))  #
mean sum squared loss
   NN.train(X[i:10+i], y[i:10+i])
NN.saveWeights(NN)


NN.predict()
```

Snippet 1. Source code for the Neural Network

Dataset

The dataset was generated algorithmically using the code below (snippet 2) and consists of a list of unsorted and sorted digits delimited by a semicolon. Overall, the dataset is about a million entries large. 10^5 entries are used for training and 1 is used for verification.




```
[1, 2, 7, 4, 2, 2, 8, 8, 5, 8] ; [1, 2, 2, 2, 4, 5, 7, 8, 8, 8]
[0, 6, 9, 3, 2, 5, 2, 8, 1, 1] ; [0, 1, 1, 2, 2, 3, 5, 6, 8, 9]
[5, 8, 1, 3, 6, 9, 1, 9, 3, 1] ; [1, 1, 1, 3, 3, 5, 6, 8, 9, 9]
[8, 0, 2, 6, 2, 0, 8, 1, 3, 4] ; [0, 0, 1, 2, 2, 3, 4, 6, 8, 8]
[7, 6, 2, 1, 8, 2, 4, 2, 4, 3] ; [1, 2, 2, 2, 3, 4, 4, 6, 7, 8]
[0, 9, 2, 9, 7, 1, 7, 9, 0, 3] ; [0, 0, 1, 2, 3, 7, 7, 9, 9, 9]
[3, 6, 5, 5, 9, 9, 6, 5, 6, 3] ; [3, 3, 5, 5, 5, 6, 6, 6, 9, 9]
[5, 4, 5, 9, 6, 2, 8, 5, 1, 8] ; [1, 2, 4, 5, 5, 5, 6, 8, 8, 9]
[7, 4, 2, 5, 8, 8, 4, 8, 4, 7] ; [2, 4, 4, 4, 5, 7, 7, 8, 8, 8]
[7, 1, 5, 2, 5, 6, 8, 3, 5, 7] ; [1, 2, 3, 5, 5, 5, 6, 7, 7, 8]
[9, 8, 7, 7, 3, 4, 0, 4, 8, 3] ; [0, 3, 3, 4, 4, 7, 7, 8, 8, 9]
[2, 9, 9, 0, 5, 1, 7, 1, 9, 5] ; [0, 1, 1, 2, 5, 5, 7, 9, 9, 9]
[3, 3, 4, 0, 0, 4, 4, 8, 7, 2] ; [0, 0, 2, 3, 3, 4, 4, 4, 7, 8]
[3, 4, 7, 0, 3, 0, 5, 9, 0, 8] ; [0, 0, 0, 3, 3, 4, 5, 7, 8, 9]
[5, 6, 1, 5, 3, 8, 0, 3, 8, 8] ; [0, 1, 3, 3, 5, 5, 6, 8, 8, 8]
[4, 3, 8, 1, 7, 5, 0, 9, 5, 1] ; [0, 1, 1, 3, 4, 5, 5, 7, 8, 9]
[2, 9, 6, 6, 4, 8, 3, 6, 9, 5] ; [2, 3, 4, 5, 6, 6, 6, 8, 9, 9]
[1, 4, 8, 3, 8, 5, 3, 0, 8, 7] ; [0, 1, 3, 3, 4, 5, 7, 8, 8, 8]
[6, 7, 6, 0, 6, 1, 9, 5, 3, 9] ; [0, 1, 3, 5, 6, 6, 6, 7, 9, 9]
[0, 2, 6, 5, 9, 6, 0, 1, 1, 2] ; [0, 0, 1, 1, 2, 2, 5, 6, 6, 9]
[6, 2, 2, 5, 3, 1, 3, 6, 0, 0] ; [0, 0, 1, 2, 2, 3, 3, 5, 6, 6]
[5, 4, 2, 2, 0, 3, 3, 1, 9, 9] ; [0, 1, 2, 2, 3, 3, 4, 5, 9, 9]
[2, 3, 3, 1, 6, 2, 4, 7, 3, 1] ; [1, 1, 2, 2, 3, 3, 3, 4, 6, 7]
[9, 4, 3, 9, 2, 3, 4, 1, 6, 6] ; [1, 2, 3, 3, 4, 4, 6, 6, 9, 9]
[1, 2, 8, 0, 1, 8, 2, 1, 3, 8] ; [0, 1, 1, 1, 2, 2, 3, 8, 8, 8]
[4, 5, 8, 2, 0, 5, 4, 7, 9, 9] ; [0, 2, 4, 4, 5, 5, 7, 8, 9, 9]
[7, 3, 8, 1, 0, 4, 6, 5, 3, 3] ; [0, 1, 3, 3, 3, 4, 5, 6, 7, 8]
[4, 2, 1, 1, 1, 3, 6, 3, 2, 9] ; [1, 1, 1, 2, 2, 3, 3, 4, 6, 9]
[2, 2, 6, 5, 4, 1, 5, 5, 5, 7] ; [1, 2, 2, 4, 5, 5, 5, 5, 6, 7]
[7, 5, 9, 4, 2, 7, 3, 6, 5, 3] ; [2, 3, 3, 4, 5, 5, 6, 7, 7, 9]
[8, 3, 4, 0, 0, 9, 4, 6, 6, 0] ; [0, 0, 0, 3, 4, 4, 6, 6, 8, 9]
[2, 8, 7, 9, 8, 7, 1, 2, 7, 6] ; [1, 2, 2, 6, 7, 7, 7, 8, 8, 9]
[0, 0, 1, 1, 6, 2, 3, 6, 4, 7] ; [0, 0, 1, 1, 2, 3, 4, 6, 6, 7]
[3, 2, 9, 9, 2, 1, 5, 2, 1, 6] ; [1, 1, 2, 2, 2, 3, 5, 6, 9, 9]
[4, 4, 5, 2, 4, 8, 0, 8, 0, 4] ; [0, 0, 2, 4, 4, 4, 4, 5, 8, 8]
```

Fig 2. Generated Dataset using a random number generator and sorting algorithm

```python
#!/home/erud1t3/anaconda3/bin/python
import random
#import torch



def generateTenDigitList():
    '''
        generate a list of ten number
    '''
    random.seed()
    List = []
    #stochastically generate a 10 numbers between 0 and 9,
    #adds them to List
    for i in range(10): List.append(random.randint(0, 9))

    # print(List)
    return List

def generateDataset(size):
    '''
        Generate data to output file
    '''
    with open("data.txt", "a") as file:
        for i in range(size + 1):

            Unsorted = generateTenDigitList() #unsorted generated list
            Sorted = Unsorted.copy() #copy the unsorted
            Sorted.sort() #sort the copy

            file.write(str(Unsorted)) #write the list in file
            # file.write("\n")
            file.write(" ; ") #delimited between sorted and unsorted
            file.write(str(Sorted)) # write sorted to the file
            file.write("\n")
            # print(Unsorted)
```

```
        # print(Sorted)
        # file.closed


generateDataset2(1000) # 1000 entries in dataset
```

Snippet 2. Code for generating the dataset for the neural network

Results

After training for 100000 epochs (iterations), the result below in fig 3 is obtained. It shows that the neural network is mostly working with rounding errors for some of the values. On average, the network comes to an accuracy of 71% when sorting those list. Increasing the dataset doesn't seem to increase the accuracy of the network, leading us to believe that the network might need more hidden layers, or neurons other than the linear neurons used in this project.

```
PROBLEMS 2    OUTPUT    DEBUG CONSOLE    TERMINAL

Epoch 99956 | Loss: 0.004783336538821459
Epoch 99957 | Loss: 0.005329110659658909
Epoch 99958 | Loss: 0.005754021927714348
Epoch 99959 | Loss: 0.005773779936134815
Epoch 99960 | Loss: 0.0060280608013272285
Epoch 99961 | Loss: 0.005765067413449287
Epoch 99962 | Loss: 0.005994296632707119
Epoch 99963 | Loss: 0.005647574085742235
Epoch 99964 | Loss: 0.004716663155704737
Epoch 99965 | Loss: 0.004848442506045103
Epoch 99966 | Loss: 0.00559783773496747
Epoch 99967 | Loss: 0.006598487962037325
Epoch 99968 | Loss: 0.006251503247767687
Epoch 99969 | Loss: 0.008868611417710781
Epoch 99970 | Loss: 0.006505672354251146
Epoch 99971 | Loss: 0.007395375519990921
Epoch 99972 | Loss: 0.006338523235172033
Epoch 99973 | Loss: 0.006395278498530388
Epoch 99974 | Loss: 0.007438764441758394
Epoch 99975 | Loss: 0.007527890149503946
Epoch 99976 | Loss: 0.006785225123167038
Epoch 99977 | Loss: 0.006884321570396423
Epoch 99978 | Loss: 0.007644311059266329
Epoch 99979 | Loss: 0.0071035148575901985
Epoch 99980 | Loss: 0.006676497869193554
Epoch 99981 | Loss: 0.006214580498635769
Epoch 99982 | Loss: 0.006752565037459135
Epoch 99983 | Loss: 0.006675730459392071
Epoch 99984 | Loss: 0.008002104237675667
Epoch 99985 | Loss: 0.00798657163977623
Epoch 99986 | Loss: 0.008537156507372856
Epoch 99987 | Loss: 0.007391761057078838
Epoch 99988 | Loss: 0.007956686429679394
Epoch 99989 | Loss: 0.007191718090325594
Epoch 99990 | Loss: 0.010293816216289997
Epoch 99991 | Loss: 0.006008201744407415
Epoch 99992 | Loss: 0.005676586646586657
Epoch 99993 | Loss: 0.005628600250929594
Epoch 99994 | Loss: 0.0046866824850440025
Epoch 99995 | Loss: 0.004136854782700539
Epoch 99996 | Loss: 0.0036275130696594715
Epoch 99997 | Loss: 0.0033568714279681444
Epoch 99998 | Loss: 0.0033174429554492235
Epoch 99999 | Loss: 0.0026417416520416737

Predicted data based on trained weights:

Input :
tensor([9., 3., 5., 1., 0., 5., 2., 7., 8., 1.])
Target:
tensor([0., 1., 1., 2., 3., 5., 5., 7., 8., 9.])
Output:
tensor([0., 1., 1., 2., 3., 4., 5., 6., 8., 9.])
```

Fig 3. Result output by the Neural Network after 100000 epochs training

References:

"PyTorch Documentation." PyTorch,pytorch.org/docs/stable/index.html.

Parmar, Ravindra. "Common Loss Functions in Machine Learning – Towards Data Science." Towards Data Science, Towards Data Science, 2 Sept. 2018, towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

MOAWAD, Assaad. "Neural Networks and Backpropagation Explained in a Simple Way." Medium.com, Medium, 1 Feb. 2018, medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e.

Elvis. "A Simple Neural Network from Scratch with PyTorch and Google Colab." Medium.com, Medium, 13 Aug. 2018, medium.com/dair-ai/a-simple-neural-network-from-scratch-with-pytorch-and-google-colab-c7f3830618e0.

V, Avinash Sharma. "Understanding Activation Functions in Neural Networks." Medium.com, Medium, 30 Mar. 2017, medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

Jain, Yashvardhan. "Create a Neural Network in PyTorch - And Make Your Life Simpler." Medium.com, Medium, 27 June 2018, medium.com/coinmonks/create-a-neural-network-in-pytorch-and-make-your-life-simpler-ec5367895199.

"Derivation: Derivatives for Common Neural Network Activation Functions." The Clever Machine, 9 Sept. 2014, theclevermachine.wordpress.com/2014/09/08/derivation-derivatives-for-common-neural-network-activation-functions/.

DOCUMENTATION (NOT PART OF REPORT)

Prompt

Multilayer Perceptron to sort ten digits. I prefer implementation with PyTorch.

Input: list of ten digits between 0 through 9, with repetitions allowed.
Note that some digits may be missing from the list.
Output: a sorted list of input digits.

Example, for an input (5, 6, 6, 2, 7, 9, 0, 1, 2, 7) the output is (0, 1, 2, 2, 5, 6, 6, 7, 7, 9). Efforts will be graded rather than the success/failure of the project. [20 pts]
More details may be posted later on training, expected materials in the report, etc.

Issues to think on

1. The number of layers in the neural network (NN).
    1. 10 neurons input layer for each digit (each for a digit)
        1. 1 Hidden layer with 10 neurons
        2. 10 neuron output
    2. 10 neurons output layer (each for a digit)
2. Loss function, activation function, and other parameters.
    1. Loss function: Mean Square Error
    2. Activation function: Sigmoid
3. Size of the data set to generate for the training/verification purpose.
4. Does there exist any relevant literature on the web?

Report Guidelines

1. Map of your trained NN along with their weights will be part of your report.
2. How successful is your NN in sorting is to be answered quantitatively for validation?

Report Requirements

1) Type of neural network (NN) and its graphics drawing including exact layers in the network, nodes per layer.
2) Loss function, activation function, and other parameters used: you may copy-paste your source code (Tensorflow/Caffe/…).
3) Sample data set and its size used to train/verify your net.
4) The description of how you generated training data (even if manual).
5) Map of your finally trained NN along with their final weights (after training). If possible combine this with 1).
6) Verification results (accuracy, etc.).
7) Any reference you found and/or used?