AST 5900: Problem Set 5

Gilberto Garcia

April 22, 2024

Problem 1a.

Answer 1a. We use the given neural network to code the following functions, which represent the connection between our nodes.

$$N_1 = \sigma(w_{x1,N1} * x_1 + w_{x2,N1} * x_2 + b_{N1})$$
$$N_2 = \sigma(w_{x1,N2} * x_1 + w_{x2,N2} * x_2 + b_{N2})$$
$$N_3 = \sigma(w_{x1,N3} * x_1 + w_{x2,N3} * x_2 + b_{N3})$$
$$y = \sigma(w_{y,N1} * N_1 + w_{y,N2} * N_2 + w_{y,N3} * N_3 + b_y)$$

where σ is the activation function tanh().

Problem 1b.

Answer 1b. We pass three points to the neural network above and report the results in the table below:

point	result
(0,0)	-0.005
(7.5,2.5)	0.606
(-5,-2)	-0.795

Problem 1c.

Answer 1c. We now make a 20 by 20 grid for the values -10 through 10 on both axis. We pass this grid to our neural network and plot the result as a heat map, shown in figure 1.

Problem 1d.

Answer 1d. We make similar plot to that of figure 1, but in this instance, we change some of the weights. The plot is shown below in figure 2. The new set of weights can be found within the code.

Problem 2a.

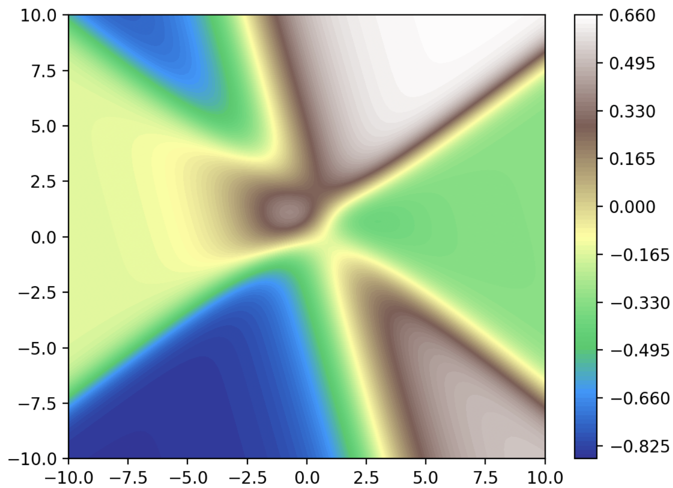


Figure 1: Heat map of solutions to our neural network in the x and y slice of -10 to 10.

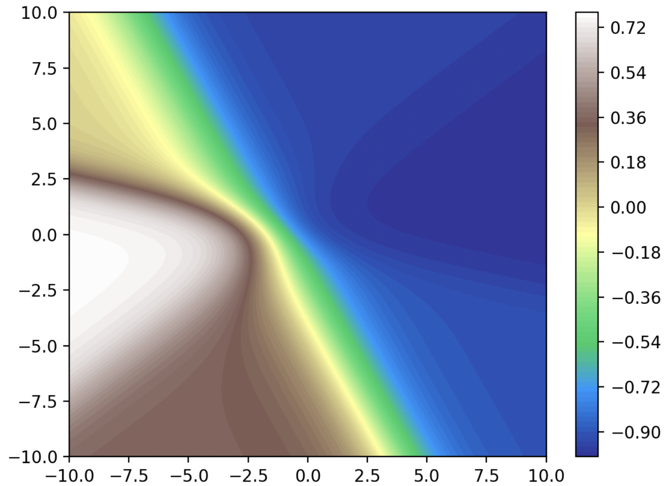


Figure 2: Heat map of solutions to our neural network in the x and y slice of -10 to 10 with a different set of weights.

Answer 2a. We are given that $L(\hat{y}, y) = (\hat{y} - y)^2$. We differentiate with respect to $w_{N1,y}$:

$$\frac{\partial L}{\partial w_{N1,y}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{N1,y}}$$

where

$$\boxed{\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)}.$$

Problem 2b.

Answer 2b. Using the given equations for $\hat{y}$ and σ', we find that

$$\boxed{\frac{\hat{y}}{w_{N1,y}} = \sigma' N_1}.$$

Problem 2c.

Answer 2c. We combine our answers for (2a) and (2b) to get that

$$\boxed{\frac{\partial L}{\partial w_{N1,y}} = 2(\hat{y} - y)\sigma' N_1}$$

Problem 2d.

Answer 2d. We want to use the following equation to update our weight:

$$w_{i+1} = w_i - l\frac{\partial L}{\partial w_i} \tag{1}$$

where $l = 0.1$, $w_i = w_{N1,y} = -0.56$, $\hat{y} = 0.5$, $y = 1$, $N_1 = 0.8$, and $\sigma' = 3$.

Plugging everything in,

$$w_{i+1} = w_i - l\big(2(\hat{y} - y)\sigma' N_1\big)$$

$$w_{i+1} = -0.56 - (0.1)\big(2(0.5 - 1)(3)(0.8)\big)$$

$$\boxed{w_{i+1} = -0.32}$$

Problem 3a.

Answer 3a. We use `TensorFlow` to make a neural network that follows the specifications given in the question and train it using the magic04 data set. We achieve a test accuracy of $\boxed{0.801}$. We plot the accuracy as a function of epoch in fig 3.

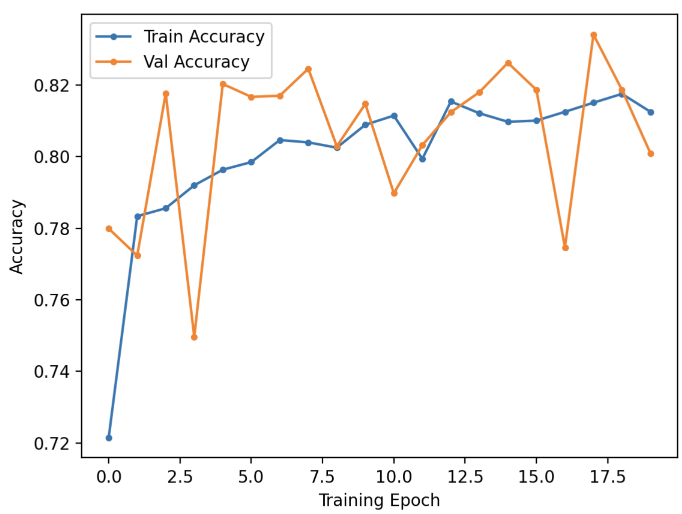
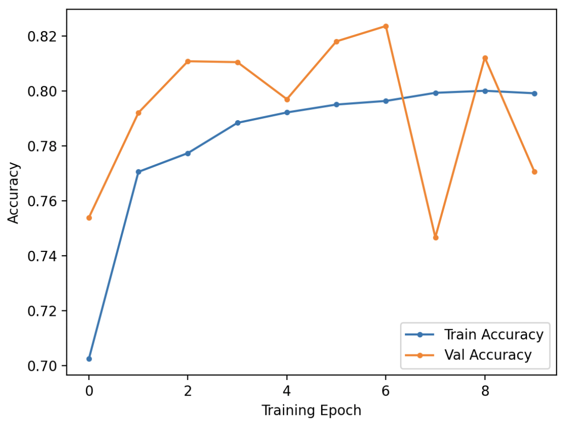


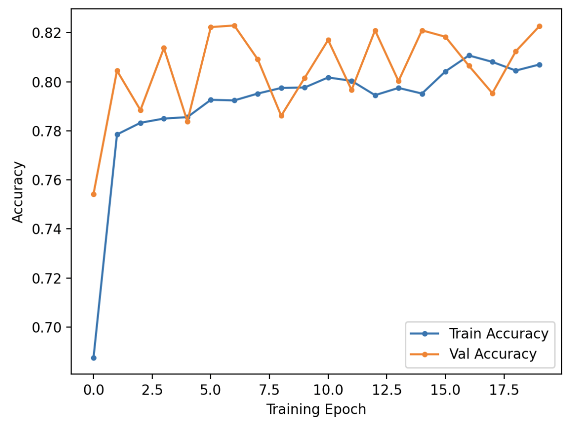
Figure 3: Accuracy as a function of epochs for training and validation sets.

Problem 3b.

Answer 3b. To see the impact of epochs and number of nodes, we train two more neural networks where we change number of epochs to 10 in one and change number of nodes to 50 in the other. We obtain a train accuracy of $\boxed{\text{0.775 and 0.820}}$, respectively. We plot their accuracy as a function of epochs in figure 4. Clearly, the number of epochs has a stronger effect on accuracy than the number of nodes. But both remained relatively close to the original accuracy.



(a) Retrained neural network with 10 epochs.



(b) Retrained neural network with 50 nodes.

Figure 4: Retraining the same neural network with some parameters adjusted.

Problem 3c.

Answer 3c. We use `scikit-learn-normalize` to normalize the data before training it. After we train it, we obtain a train accuracy of 0.812. We plot the accuracy as a function of epochs in figure 5. The accuracy stayed about the same but the accuracy curve shows a much smoother convergence to the final accuracy. This tells us that it could be preferable to use a normalized data set.

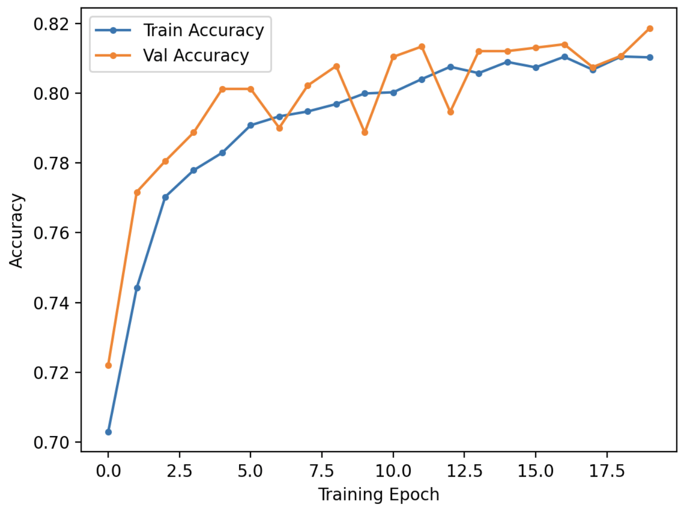


Figure 5: Accuracy as a function of epochs for the normalized data set.

Code

```python
'''
Gilberto Garcia
Computational Physics
April 15 2024
HW5 - Neural Networks
'''

#import required libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import ticker, cm

####
#Q1#
####


    #a#

#hard coding a neural network

#we first hard code the pre-defined weights
w_x1n1, w_x1n2, w_x1n3 = -0.91, 0.09, 0.72
w_x2n1, w_x2n2, w_x2n3 = 0.72, 0.34, 0.62
w_n1y, w_n2y, w_n3y = -0.56,0.94,-0.47
b_n1,b_n2,b_n3,b_y = 0.29,0.05,-0.99,-0.25
#define our activation function
```

```python

#now we define the fxns that connect our nodes
def neural_network(activation_fxn,x1,x2):
    n1 = activation_fxn((w_x1n1 * x1) + (w_x2n1 * x2) + b_n1)
    n2 = activation_fxn((w_x1n2 * x1) + (w_x2n2 * x2) + b_n2)
    n3 = activation_fxn((w_x1n3 * x1) + (w_x2n3 * x2) + b_n3)
    y = activation_fxn((w_n1y*n1) + (w_n2y*n2) + (w_n3y*n3) + b_y)
    return y


    #b#

#use your neural network on the following points:
# point1(0,0),point2(7.5,2.5),point3(-5,-2)


point1 = neural_network(np.tanh,0,0)
point2 = neural_network(np.tanh,7.5,2.5)
point3 = neural_network(np.tanh,-5,-2)
print(point1,point2,point3)


    #c#
N = 100
value_matrix = np.zeros((N,N))
x,y = np.linspace(-10,10,N),np.linspace(-10,10,N)

for i in range(value_matrix.shape[0]):
    for j in range(value_matrix.shape[1]):
        value_matrix[i,j] = neural_network(np.tanh,x[i],y[j])

#X,Y = np.meshgrid(x,y)


cs = plt.contourf(x,y,value_matrix, levels=N,cmap='terrain')
cbar = plt.colorbar(cs)
plt.show()



    #d#

#we change the weights now for fun
w_x1n1, w_x1n2, w_x1n3 = -0.5, 0.21, 0.9
w_x2n1, w_x2n2, w_x2n3 = 0.4, 0.4, 0.2
w_n1y, w_n2y, w_n3y = -0.33,-0.94,-0.5
b_n1,b_n2,b_n3,b_y = 0.5,0.3,-0.1,-0.75
```

```
N = 100
value_matrix = np.zeros((N,N))
x,y = np.linspace(-10,10,N),np.linspace(-10,10,N)

for i in range(value_matrix.shape[0]):
    for j in range(value_matrix.shape[1]):
        value_matrix[i,j] = neural_network(np.tanh,x[i],y[j])

cs = plt.contourf(x,y,value_matrix, levels=N,cmap='terrain')
cbar = plt.colorbar(cs)
plt.show()





####
#Q3#
####

import tensorflow as tf
from sklearn.preprocessing import normalize
import pandas as pd
#read in our data first
magic04 = pd.read_csv('magic04.data',sep=',',index_col=None,header=None)
#we randomize our rows
magic04 = magic04.sample(frac=1)

#we want to create training and testing data sets
#to do this, we split our magic04 dataset into these two components
#we will use 80% of our data for the training set and 20% for the testing set
sample_percent = 0.8
training_index = int(sample_percent*magic04.shape[0])
testing_index = magic04.shape[0] - training_index
training = magic04[:training_index]
testing = magic04[testing_index:]
#we split both of them into sample and label data sets
#the sample is the data (x values) and the label is the output (y values,
    detection or non-detection)
training_sample,training_label = training.iloc[:,:-1],training.iloc[:,-1]
testing_sample,testing_label = testing.iloc[:,:-1],testing.iloc[:,-1]


#we have our data ready, we now want to create and ready our model
```

```python
#we will create a model with 1 hidden layer of 100 nodes and 1 outer layer with 2
    nodes
#we use the code from class to make the model, compile, and test it
counter = 0
for nodes in [100,50,100,100]:
    if counter == 3:
        training_sample = normalize(training_sample)
        testing_sample = normalize(testing_sample)
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(10,)),
        tf.keras.layers.Dense(nodes, activation='relu'),
        tf.keras.layers.Dense(2, activation='softmax')
    ])
    #we ready our model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    #we can now pass our data to the model and train it
    if counter == 0:
        epochs = 10
    else:
        epochs = 20
    counter +=1
    batch_size = 32
    validation_split = 0.2
    history = model.fit(training_sample,
                        training_label,
                        batch_size=batch_size,
                        epochs=epochs,
                        validation_split=validation_split)


    #we can evaluate the model and print our accuracy
    test_loss, test_acc = model.evaluate(testing_sample, testing_label)
    print(f"Test Accuracy: {test_acc:.3f}")

    #let's plot the accuracy across epochs
    plt.xlabel("Training Epoch")
    plt.ylabel("Accuracy")
    plt.plot(history.epoch, history.history['accuracy'], marker = '.', label = '
        Train Accuracy')
    plt.plot(history.epoch, history.history['val_accuracy'], marker = '.', label =
         'Val Accuracy')
    plt.legend()
    plt.show()
```