Assignment 3: Neural Networks

Fall 2018

Due Date: October 12, 2018

Instructions

- There are two parts to this assignment. The first part requires you to solve some theoretical/numerical questions, and the second part requires you to code a 2 hidden layer neural network.
- For the programming part, please use parameters and not hard coded paths or values. All instructions for compiling and running your code must be placed in the README file.
- All work submitted must be your own. Do not copy from online sources. If you use any references, please list them.
- You should use a cover sheet, which can be downloaded from: `http://www.utdallas.edu/~axn112530/cs6375/CS6375_CoverPage.docx`
- You are allowed to work in pairs i.e. a group of two students is allowed. Please write the names of the group members on the cover page.
- **You have a total of 4 free late days for the entire semester. You can use at most 2 days for any one assignment. After four days have been used up, there will be a penalty of 10% for each late day. The submission for this assignment will be closed 2 days after the due date.**
- Please ask all questions on Piazza, not via email.

1 Theoretical Part (40 points)

For the following, please show all steps of your derivation and list any assumptions that you make. You can submit typed or **legible** hand-written solutions. If the TA cannot read your handwriting, no credit will be given.

1.1 Revisiting Backpropagation Algorithm

In class we had derived the backpropagation algorithm for the case where each of the hidden and output layer neurons used the sigmoid activation function:

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

Revise the backpropagation algorithm for the case where each hidden and output layer neuron uses the
a. tanh activation function

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

b. ReLu activation function:

$$ReLu(x) = \max(0, x)$$

Show all steps of your derivation and the final equation for output layer and hidden layers.

1.2 Gradient Descent

Derive a gradient descent training rule for a single unit neuron with output o, defined as:

$$o = w_0 + w_1(x_1 + x_1^2) + \cdots + w_n(x_n + x_n^2)$$

where $x_1, x_2, \ldots, x_n$ are the inputs, $w_1, w_2, \ldots, w_n$ are the corresponding weights, and w_0 is the bias weight. You can assume an identity activation function i.e. $f(x) = x$. Show all steps of your derivation and the final result for weight update. You can assume a learning rate of η.

1.3 Comparing Activation Function

Consider a neural net with 2 input layer neurons, one hidden layer with 2 neurons, and 1 output layer neuron as shown in Figure 1. Assume that the input layer uses the identity activation function i.e. $f(x) = x$, and each of the hidden layers uses an activation function $h(x)$. The weights of each of the connections

are marked in the figure.

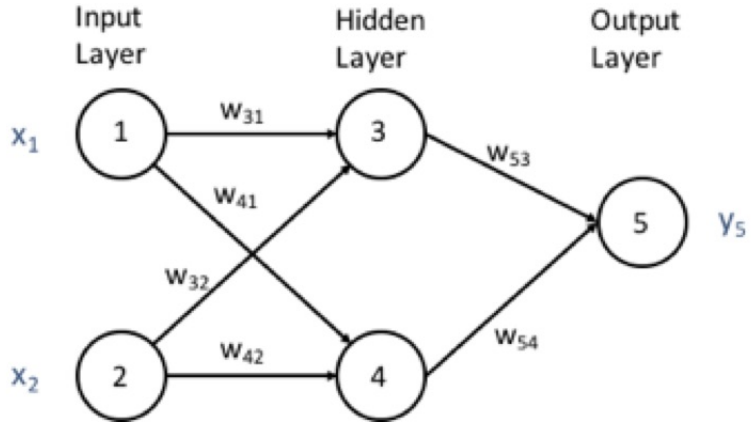

Figure 1: A neural net with 1 hidden layer having 2 neurons

a. Write down the output of the neural net y_5 in terms of weights, inputs, and a general activation function $h(x)$.

b. Now suppose we use vector notation, with symbols defined as below:

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$W^{(1)} = \begin{pmatrix} w_{3,1} & w_{3,2} \\ w_{4,1} & w_{4,2} \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} w_{5,3} & w_{5,4} \end{pmatrix}$$

Write down the output of the neural net in vector format using above vectors.

c. Now suppose that you have two choices for activation function $h(x)$, as shown below:
Sigmoid:

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

Tanh:

$$h_t(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Show that neural nets created using the above two activation functions can generate the same function.

Hint: First compute the relationship between $h_s(x)$ and $h_t(x)$ and then show that the output functions are same, with the parameters differing only by linear transformations and constants.

1.4 Gradient Descent with a Weight Penalty

Solve question 4.10 from Tom Mitchell's textbook. Show all steps of derivation and clearly state the final update rule for output as well as hidden layers.

2 Programming Part (60 points)

In this part, you will code a neural network (NN) having two hidden layers, besides the input and output layers. You will be required to pre-process the data and then run the processed data through your neural net. A sample code file in Python is provided for you to use as a starter. It uses sigmoid as the activation function, and you have to add two more activation functions - **tanh** and **ReLu**.
Below are the requirements of the program

- A starter code written in Python 3.6 that includes the **sigmoid** functions is provided, you need to add the two other activation functions. You can use modify the code as you need.
- The code has sections marked "TODO" that you need to complete. These include code for pre-processing the dataset, adding 2 other activation functions, and adding a method for using the model for prediction on a test dataset.
- For the pre-processing method, you need to convert non-numerical attributes e.g. categorical attributes to numerical values, standardizing, and scaling the attributes. You also need to handle null or missing values. A skeleton method is provided.
- When you create the two new activation functions, remember to modify the helper methods, such as **compute_output_delta**, and **compute_hidden_layer1_delta**, etc
- For the prediction part, you have to apply the learned model to a test dataset whose path is specified by the parameter. A skeleton method is provided.

You can make the following assumptions and simplifications for this program:

- You can ignore the bias neurons
- You can assume that the last column will be the class label column
- You can assume that the training and test datasets will have the same format i.e. same number and placement of columns
- You don't need to implement regularization, adaptive learning rate, or momentum factors.

You should test your code on **any one** of the three datasets using all three activation functions:

1. Car Evaluation Dataset: `https://archive.ics.uci.edu/ml/datasets/Car+Evaluation`
2. Iris Dataset: `https://archive.ics.uci.edu/ml/datasets/Iris`

3. Adult Census Income Dataset: `https://archive.ics.uci.edu/ml/datasets/Census+Income`

You can partition these datasets into any suitable training and testing ratio and output your results. You should try to tune the parameters (e.g. learning rate) as much as possible, however you are not responsible for finding the best parameters. Submit your results in a tabular format.

What to submit:
You need to submit the following for the programming part:

- Your source code
- Output for your dataset summarized in a tabular format for different combination of parameters
- A brief report summarizing your results. For example, which activation function performed the best and why do you think so.
- Any assumptions that you made or any bugs in the supplied code that you found.