Assignment 2: Neural Networks

Spring 2020

Due Date: Mar 5, 2020

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 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 and output layer use an activation function $h(x)$. The weights of each of the connections are marked in the figure.

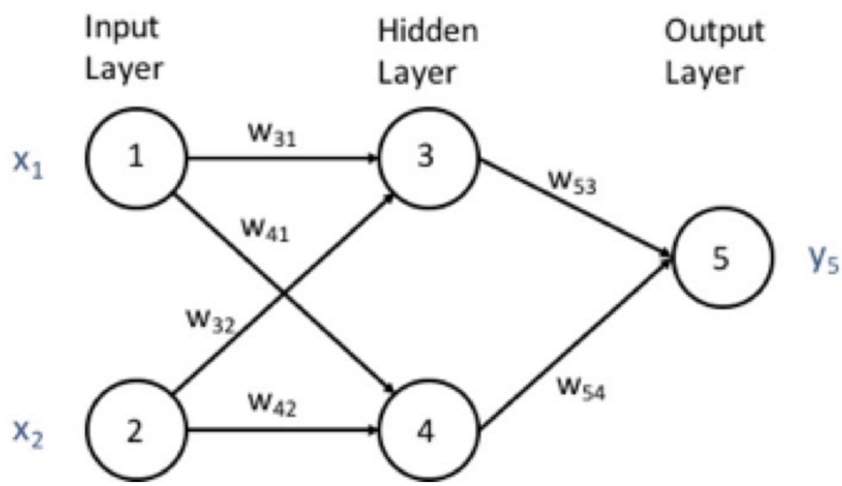

Figure 1: A neural net with 1 hidden layer having 2 neurons

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

Go through Chapter 4 of Tom Mitchell's Machine Learning textbook, which is available at the following link:
`https://users.cs.northwestern.edu/~pardo/courses/eecs349/readings/chapter4-ml.pdf`
Solve question 4.10 from the book. 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 at least two hidden layers, besides the input and output layers. You are required to pre-process the data and then run the processed data through your neural net. Below are the requirements and suggested steps of the program

- You can use any programming language from the following list:
  - Python 3.x
  - Java
  - R
  - C#
- **You cannot use any libraries for neural net creation or pre-processing.** The only exception is that you can use libraries for data loading and manipulation, such as pandas or numpy.
- As the first step, pre-process and clean your dataset. There should be a method that does this. Use the techniques discussed in class as well as any other techniques that you would like.
- Split the pre-processed dataset into training and testing parts. You are free to choose any reasonable value for the train/test ratio, but be sure to mention it in the README file.
- Code a neural net having **at least two hidden layers**. You are free to select the number of neurons in each layer. Each neuron in the hidden and output layers should have a bias connection.
- You are required to code three different activation functions:
  1. Sigmoid
  2. Tanh
  3. ReLu

  The earlier part of this assignment may prove useful for this stage. The activation function should be a parameter in your code.
- Code a method for creating a neural net model from the training part of the dataset. Report the training accuracy.
- Apply the trained model on the test part of the dataset. Report the test accuracy.
- You should try to tune parameters like learning rate, activation functions, etc. Report your results in a tabular format, with a column indicating the parameters used, a column for training accuracy, and one for test accuracy.
- You don't need to implement regularization, adaptive learning rate, or momentum factors.

Dataset

You can use **any one** dataset from the UCI ML repository:
`https://archive.ics.uci.edu/ml/datasets.php`
Note: If the above direct link does not work, you can just Google the UCI ML repository.

What to submit:

You need to submit the following for the programming part:

- Link to the dataset used. *Please do not include the data as part of you submission.*
- Your source code and a README file indicating how to run your 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.