CS 722/822 Machine Learning
Homework #4
Due: Nov. 10th, by the end of the day

Programming Assignment

1. Introduction

In this assignment, you will be implementing feed forward neural networks using stochastic gradient descent in **python**. You will implement one neural network template. Number of layers and number of neurons in each layer will be input to this neural network template to instantiate specific neural networks. You will compare the performance of both models on the UCI Wine Dataset. The task is to predict the quality of a wine (scored out of 10) given various attributes of the wine (for example, acidity, alcohol content).

2. Neural Networks

We will use a three layer neural network as an example to illustrate how the training and prediction can be done for neural networks. Follow figure 1 shows an architecture of such a three layer feedforward neural network.

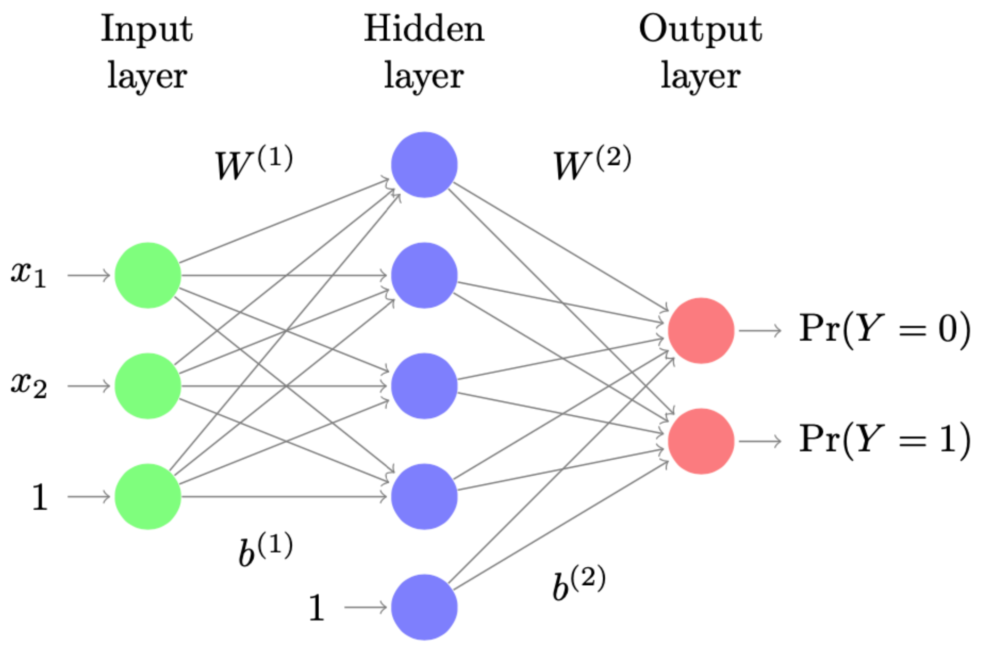

Figure 1

2.1 Training

We will first consider the loss and gradient for a single training data point pair $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^k$. For example in Figure 1, we assume input dimension n=2, and output dimension k=2.

We will use the summed square error loss. That is we want to minimize $\frac{1}{2}(a^{(3)} - y)^2$ for each training example, where $a^{(3)}$ (Vector) is the activation from the neural network on a single example, which is described in detail below.

2.1.1 Forward propagation

First we have to compute the response of the network to individual data examples. We let $W^{(i)}$ represents the weights on the edges between the i and i+1 layer. $b^{(i)}$ are the corresponding biases. S is an activation function. In this assignment, we will use logistic function $S(z) = \frac{1}{1+e^{-z}}$ as activation function. Then we get the activations as follows

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = S(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(3)} = S(z^{(3)})$$

Note, even in output layer we still have activation function. Also, the same activation function is being used across different layers.

We can then compute the cost for a single training example as

$$J(W, b, x, y) = \frac{1}{2}(a^{(3)} - y)^2$$

Hint: when you implement the neural networks, you can use separate variables for $W^{(1)}$ and $b^{(1)}$, but you can also use the same variable $\hat{W^{(1)}}$, where $\hat{W^{(1)}} = [W^{(1)}, b^{(1)}]$, which is concatenation of original variables $W^{(1)}$ and $b^{(1)}$. In this new setting, we can consider input dimension as $n+1$, the new input vector will be [x,1], the value for the last dimension of this new vector will always be 1. So that $W^{(1)}x + b^{(1)} = [W^{(1)}, b^{(1)}]^T[x, 1]$, where T is matrix transpose.

2.1.2 Backpropagation

Now that we have computed the output activation for an input we need to update the weights of the network for minimize error in loss. This is where we use backpropagation algorithm discussed in class. We will summarize the algorithm as follows.

1. Letting * denotes element-wise multiplication, we define the $error$ and δ (weighted error averages) as

$$error^{(3)} = (a^{(3)} - y)$$
$$\delta^{(3)} = error^{(3)} * S(z^{(3)} * (1 - S(Z^{(3)})))$$
$$error^{(2)} = (W^{(2)})^T\delta^{(3)}$$
$$\delta^{(2)} = error^{(2)} * S(z^{(2)}) * (1 - S(z^{(2)}))$$

2. The partial derivatives are then. Note that partial derivative for $W^{(1)}$ and $b^{(1)}$ are different.

$$\nabla J_{W^{(n)}}(W,b,x,y) = \delta^{(n+1)}(a^{(n)})^T$$
$$\nabla J_{b^{(n)}}(W,b,x,y) = \delta^{(n+1)}$$

The weights W and biases b needs to be initialized randomly.

Implement the function `applyneuralNet` , `applyneuralNet` compute output values in each layer in forward pass. For example in Figure 1, the function should return both $a^{(2)}$ and $a^{(3)}$.

Implement the function `trainNeuralNet` . `trainNeuralNet` function will be responsible for running backpropogation, which will update all The weights W and biases b. Updated weights W and biases b will needs to be returned.

You can imagine there will be another function which can call the above two functions to orchestrate the network training and testing process.

2.1.3 stochastic gradient descent

For the training the neural network, we will be using stochastic gradient descent algorithm to keep updating weights in the neural network to minimize the loss function. Stochastic gradient descent algorithm can be summarized as follows

> - Choose an initial vector of parameters W and b and learning rate λ
> - Repeat until an approximate minimum is obtained or predefined number of iterations are done:
>   - Randomly shuffle examples in the training set.
>   - For training samples $i = 1, 2, \ldots, n$, do:
>   - $w := w - \lambda \nabla J_W$
>   - $b := b - \lambda \nabla J_b$

2.1.4 Testing

Now that we have trained the parameters of network, we want to use those parameters to make predictions. We can do this by simply running the forward pass and returning $a^{(3)}$, using function `applyneuralNet`.

2.1.5 Neural Network Template

Please do not hard code the number of layers and the number of neurals in each layer in your implementation. It needs to be flexible enough that given an input [11,10,10], it will specify the network with 3 layers, one input layer, one hidden layer, and one output layer, each layers has 11, 10, 10 neurons respectively. While given an input [11,100,20,10], it will specify the network with 4 layers.

3. Applying neural network to Wine dataset

Now we will use the Neural Network template you just wrote to classify wine. Specifically, we will let each attribute of wine dataset be an input unit in our network and we will let the ten classes be the output neurons in the last layer.

Implement script `main` which will train and evaluate your Neural Network. You can use `train_test_split function` available in `sklearn.model_selection` to randomly split dataset into 80% for training and 20% for testing.

The reasonable setting for learning rate can be 0.01, you can also experiment with other values.

3.1 Evaluation

After the neural network has been trained using training data, we want to use those learned parameters to make predictions. The output of the neural network will be a vector having 10 dimensions, the dimension with largest value will be predicted label. If the predicted label and true class label matches, then it is a correct prediction. Please report **Train Accuracy** and **Test Accuracy** which is percentage of correctly predicted samples in training and testing perspectively. Besides report Accuracy, please also report **how the loss changes along the training**. For example, if you run 100 epochs (each epoch means finish training of all training samples once), you can report sum of square error ($\frac{1}{2}(a^{(3)} - y)^2$) in every 10 epochs. The same interval should be used across different experiments for better comparison. For example, if you want to report total loss in every 10 epochs, please do so for different learning rates experiments.

3.2 Experiment

Dataset used in this experiment is file **wine.txt**.

Experiment 1. Comparing performance of different neural network architectures.

Select 2 different settings of neural architectures with different number of layers. Sample settings can be NN1: [11,10,10], NN3: [11,50,30,10]

The minimum number of layers should be 3. In another words, NN should has at least one hidden layer.

Please report both Accuracy and training loss progress.

Experiment 2. Comparing different learning rates.

Select 3 different values of learning rate and we want to see how the different values of learning rate can affect the learning of neural networks. Please report both Accuracy and how training loss changes along the training progress.

Note, for each single experiment, please train and test neural network 3 times, and report mean and deviation from 3 runs.

4. Submission

1. submit all the code
2. submit a report
    1. report the best performance neural network architecture and setting (learning rate, number of epochs)
    2. includes all the required information mentioned in Sec 3.2 Experiment

5. Grading policy

After the deadline, all submissions will be re-evaluated and best scores will be posted on the scoreboard. Scoreboard will be published in Blackboard.

This will be the last homework we have this semester. This homework worth 7.5% of total grade. We will use 100 points to report grade in Blackboard.

- (**50 points**) Successfull implementation of one neural network. All functions are correctly implemented. The training loss decrease steadily.
- (**25 points)** Finish 3.2 Experiment 1
- (**25 points)** Finish 3.2 Experiment 2
- Students ranking from 1 to 5 will get 5 extra points on this assignment.
- Students ranking from 6 to 10 will get 3 extra points on this assignment.

If you used or referenced any online resources, please add a reference in your report.

Helpful Resources:

1. Stanford CS231n: Convolutional Neural Networks for Visual Recognition https://cs231n.github.io

You can use your own computer or deparment resources or google colab to write and test your implementations.