Homework Programming Assignment 2: Neural Network Classifier

Due Time: October 7, 2019, 3:00PM

Contents:

In this homework assignment, you need to implement a simple 2-hidden-layer Multi-Layer Neural Network using Python and Numpy. You are given data generated from three blackboxes: blackbox21, blackbox22, and blackbox23. The description and tasks for each blackbox are the same. The following instruction is for blackbox21 as an example, and you can apply the same to the blackbox22 and blackbox23 as well.

1. Homework Description:

Please use ***Python 3*** to implement your homework assignment.

1.1 Data Description

In this assignment, you are given a set of training data and a set of testing data generated from a specific blackbox, say blackbox21, which you may not know the secret function inside. You can use the training set to develop your network and use the testing set to measure the accuracy of your network during the development. For grading, we will generate our hidden testing data from the same blackbox to test the performance of your submitted NN.

For blackbox21, you are given:

- `blackbox21_train.csv`: labeled training data generated from a blackbox21
- `blackbox21_test.csv` : unlabeled testing data
- `blackbox21_example_predictions.csv`: example output, which is also the true class labels for `blackbox21_test.csv` so you can get to know the format of output and measure your model's performance while you are developing your program.

This data format is similar to the last homework assignment. However, The label (target value) of data in this homework assignment is ***not binary***. So this means in this homework assignment, you need to implement a ***Multi-Class Classifier.*** Meanwhile, there are ***three features*** in the blackboxes, not two.

1.2 Task Description

Your task is to implement a multi-hidden-layer neural network learner (see 1.3 model description part for details of neural network you need to implement), named as `NeuralNetwork.py`, that will

(1) Construct a neural network classifier from the given training data,
(2) Use the learned classifier to classify the unlabeled test data, and
(3) Output the predictions of your classifier on the test data into a file named `blackbox2*_predictions.csv` in the ***same*** directory as the `.py`,
(4) ***Finish in 5 minutes (to train one model for one blackbox)***.

Your program will take two input files and produce one output file as follows:

```
python3 NeuralNetowrk.py training_data_path testing_data_path
⇒ prediction_file
```

For example,

```
python3 NeuralNetwork.py blackbox21_train.csv blackbox21_test.csv
⇒ blackbox21_predictions.csv
```

Note: input files may not be in the same directory as your python script [1].

In other words, your algorithm file `NeuralNetwork.py` will take ***labeled training data***, ***unlabeled testing data*** as input, and output your classification predictions on testing data as output. In your implementation, ***please do not use any existing machine learning library call***. You must implement the algorithm yourself. Please develop your code yourself and do not copy from other students or from the Internet.

The format of blackbox2`*_train.csv` looks like:

x1, x2, x3, y

Where x1, x2, and x3 are the attribute values and y is the label, and blackbox2`*_test.csv` are unlabeled. Notice that ***the data here has 3 (not 2) attributes***.

Your output `blackbox2*_predictions.csv` will look like

1
0
2
5
… (A single column indicates the predicted class labels for each unlabeled sample in the input test file)

The format of your `blackbox2*_predictions.csv` file is crucial. It has to be in the ***exact same name and format*** so that it can be parsed correctly to compare with true labels by grading scripts.

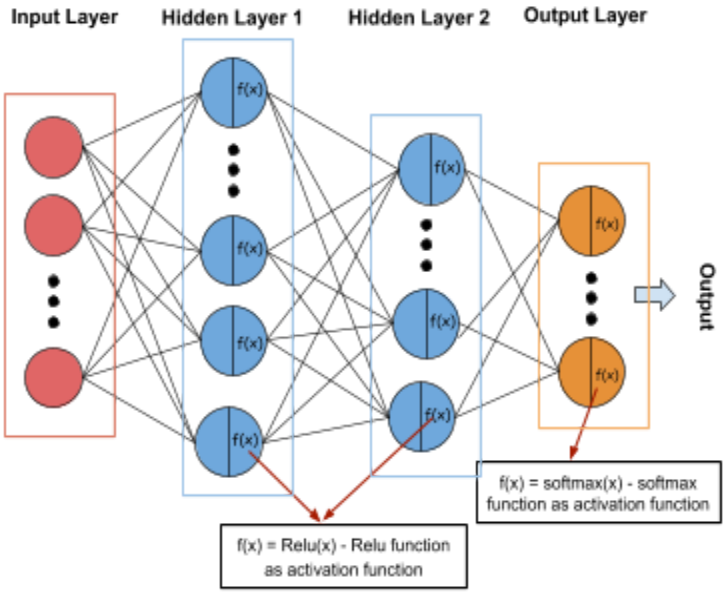
When we grade your algorithm, we will use the same training data but some unlabeled ***hidden*** testing data (generated from the same blackbox21) instead of the testing data that was given to you. Your code will be autograded for technical correctness. Please name your file correctly, or

[1] `os.path.basename(`*`path`*`)` can be used to extract the base name of pathname *path*.

you will wreak havoc on the autograder. **The maximum running time to train a model is 5 minutes *(for a single blackbox)*, so please make sure your program finishes in 5 minutes.**

1.3 Model Description

The basic structure model of neural network in this homework assignment is as below.



The above figure shows a 2-hidden-layer neural network. At each hidden layer, you need to use an activation function (e.g. relu function, see references below). Since it is a multi-class classification problem, you need to use **softmax function** (see references below) as activation at the final output layer to generate probability distribution of each class. ***There is no specific requirement on the number of nodes in each layer***, you need to choose them to make your neural network reach best performance. Also, the number of nodes in the input layer should be number of features, and the number of nodes in output layer should be number of classes.

You are encouraged to implement more than 2-layer neural networks if you want, but in this homework assignment 2-layer neural network is enough to get good performance.

There are some hyper parameters you need to tune to get better performance. You need to find best hyper-parameters so that your neural network can get good performance on test and hidden test data.

- **Learning rate**: step size for update weights (*e.g. weights = weights - learning * grads*), different optimizer have different way to use learning rate. (see reference in 2.1)
- **Batch size**: number of samples processed each time before the model is updated. The size of a batch must be more than or equal to one, and less than or equal to the number

of samples in the training dataset. (*e.g suppose your dataset is of 1000, and your batch size is 100, then you have 10 batches, each time you train one batch (100 samples) and after 10 batches, it trains all samples in your dataset*)

- **Number of epoch**: the number of complete passes through the training dataset (*e.g. you have 1000 samples, 20 epochs means you loop this 1000 samples 20 times, suppose your batch size is 100, so in each epoch you train 1000/100 = 10 batches to loop the entire dataset and then you repeat this process 20 times*)
- **Number of units in each hidden layer**

Remember that the program has to **finish in 5 minutes**, so choose your hyper-parameters wisely.

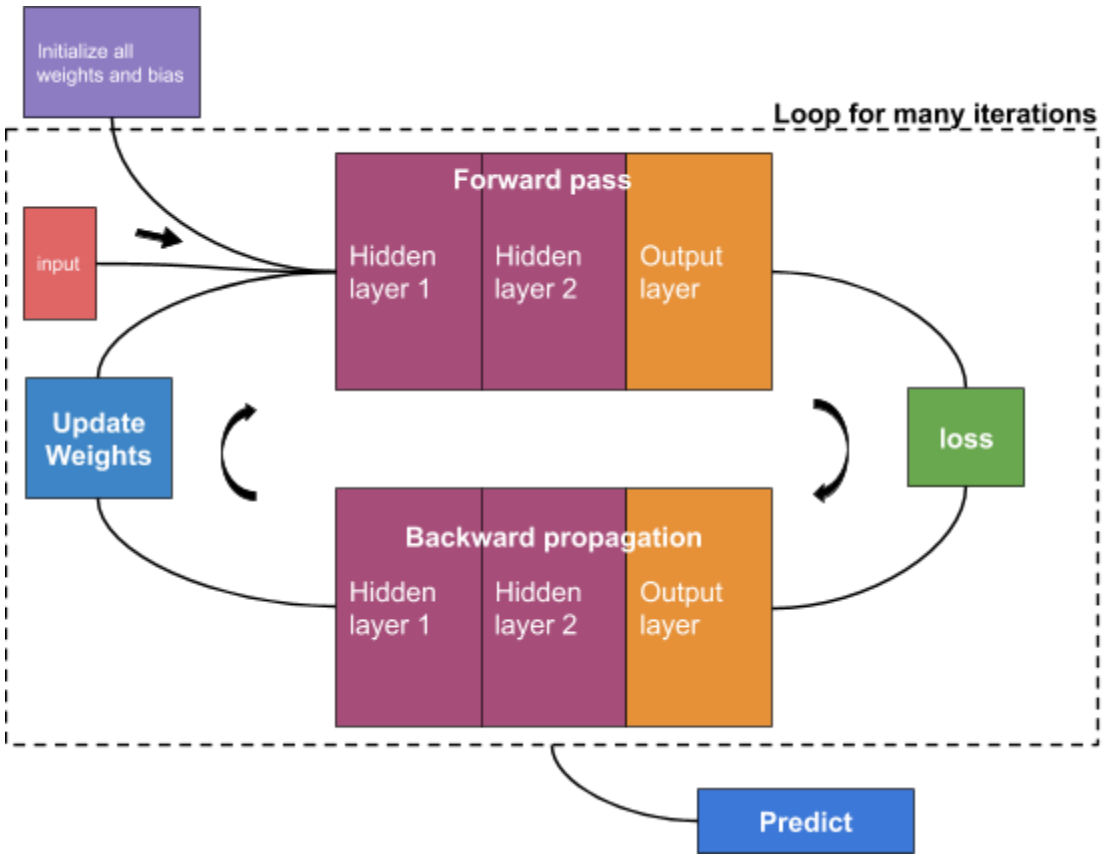
References

[1] Relu function - https://en.wikipedia.org/wiki/Rectifier_(neural_networks)
[2] Softmax function - https://en.wikipedia.org/wiki/Softmax_function#CITEREFGoodfellowBengioCourville2016

2. Implementation Guidance

2.1 Suggested steps

1. **Split dataset to batches** (see 2.3 - there is a helper function to help you generate batches)
2. **Initialize weights and bias** (see reference and 2.3) - suggest to use Xavier initialization
3. **Select one batch of data and calculate forward pass** - follow the basic structure of neural network to compute output for each layer, you might need to cache output of each layer for convenient of backward propagation.
4. **Compute loss function** - you need to use cross-entropy(logistic loss - see references) as loss function ***(see 2.3 - there is a helper function called log_loss to compute this)***
5. **Backward propagation** - use backward propagation (your implementation) to update hidden weights ***(see 2.2)***
6. **Updates weights using optimization algorithms** - there are many ways to update weighs see reference to find different optimizers ***(see 2.3 for SGD and Adam helper functions)***
7. **Repeat 2,3,4,5,6 for all batches** - after finishing this process for all batches (it just iterates all data points of dataset), it is called 'one epoch'.
8. **Repeat 2,3,4,5,6,7 number of epochs times**- You might need to train many epochs to get a good result.



References

[1] cross-entropy loss(logistic loss): https://gombru.github.io/2018/05/23/cross_entropy_loss/

[2] Optimizer for neural network

- http://cs231n.github.io/neural-networks-3/
- https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3

[3] Weights initialization

- https://mnsgrg.com/2017/12/21/xavier-initialization/
- https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78
- Example of Xavier initializations

```
"""Initialize coefficients"""
def _init_coef(self, fan_in, fan_out):
    # Use the initialization method recommended by
    # Glorot et al.
    factor = 6.
    init_bound = np.sqrt(factor / (fan_in + fan_out))

    # Generate weights and bias:
    coef_init = self._random_state.uniform(-init_bound, init_bound,
                                           (fan_in, fan_out))
    intercept_init = self._random_state.uniform(-init_bound, init_bound,
                                                fan_out)
    return coef_init, intercept_init
```

References for Backward propagation

[1] http://cs231n.github.io/optimization-2/
[2] http://cs231n.stanford.edu/2017/handouts/linear-backprop.pdf
[3] http://cs231n.stanford.edu/2017/handouts/derivatives.pdf
[4] https://www.youtube.com/watch?v=q0pm3BrIUFo
[5] https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
[6] https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html
[7]https://medium.com/binaryandmore/beginners-guide-to-deriving-and-implementing-backpropagation-e3c1a5a1e536

Feel free to find more, there are a lot of tutorials on the web.

2.2 Helper functions

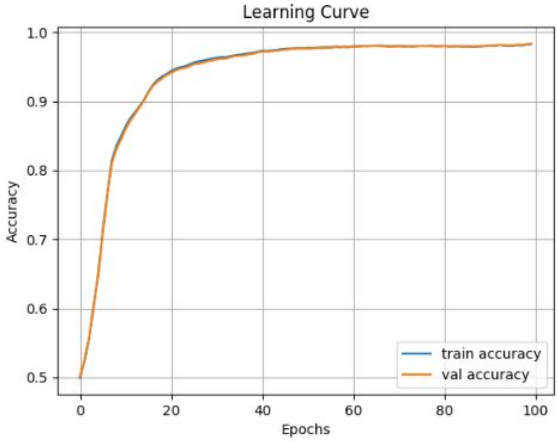
We will provide you with a bunch of helper functions. ***You might need to understand them to use since it needs to be compatible to your program design.***

1. **`relu()`** - it is used to compute output relu function.
1. **`inplace_relu_derivative()`** - it is used to compute derivative of relu function
2. **`softmax()`** - it is used to compute softmax function.
3. **`log_loss()`** - it is used to compute logistic loss (cross entropy)
4. **`SGDOptimizer Class`** - implements stochastic gradient descent optimizer algorithm with momentum.
5. **`AdamOptimizer Class`** - implements adam optimizer algorithm.
6. **`gen_batches()`** - split dataset to generate small batches.
7. **`label_binarize()`** - binarize label for multiclass

There are some more functions that you may use. See `utils.py` for details.

2.3 Learning Curve Graph

In order to make sure your neural network actually learns something, You need to make a plot as below to show the learning process of your neural networks. After every epoch (one epoch means going through all samples of data once), you need to record your accuracy of training set and validation set (it is just the test set we give you) and make a plot of those accuracy.



3. Submission:

Your submission contains 2 parts.

3.1 Submit your code to Vocareum

- Submit your `NeuralNetwork.py` to ***Vocareum***, with other related scripts that you may need (e.g. utils.py).
- After your submission, Vocareum would run two scripts to test your code, a submission script and a grading script. The submission script will test your code with ***only blackbox21*** through the training and test data we give you, while the grading script will test your code with ***all the 3 blackboxes*** through our ***hidden test data***.
- The submission script has a time limit of ***5 minutes***(for 1 blackbox). The grading script has a time limit of ***15 minutes***(for 3 blackboxes). The program would terminate and fail if it exceeds the time limit.
- After the submission script/grading script finishes, you can view your submission report immediately after the program finishes to see if your code works. The grading report will be released after the deadline of the homework. **Only test accuracy for blackbox21 will be displayed in the submission report.**
- You don't have to keep the page open while the scripts are running.

3.2 Submit your report to Blackboard

- Create a single **`.zip`** (`HW2_Firstname_Lastname.zip`) which contains:
  - `NeuralNetwork.py`. You may also submit other related scripts.
  - `Report.pdf`, a ***brief*** report indicates your ***training accuracy, testing accuracy*** and ***learning curve graph*** for blackbox21, blackbox22, and blackbox23***.***
- Submit your zipped file to the blackboard.

4. Rubric:

100 points in total

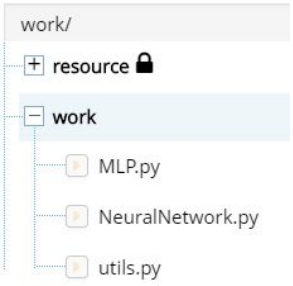
- Program correctness(60 points): your program always works correctly and meets the specifications ***within the time limit***.
- Documentation(20 points): your code is well commented and the submitted learning curve graph looks reasonable.
- Performance (20 points):
  - overall good performance on reported train and test accuracy. (10 points)
  - accuracy on hidden testing data. (10 points)

Appendix

Homework must be submitted through Vocareum. Please only upload your code to the `/work` directory. Don't create any subfolder or upload any other files. Please refer to http://help.vocareum.com/article/30-gettingstarted-students to get started with Vocareum.

How to submit

1. Log in to Vocareum website and you should see ***Assignment 1*** under INF 552 Course
2. Click ***My Work***
3. Select ***work*** folder on the left
4. You can either create new files or upload files from your local machine. Make sure to name your python file correctly based on homework description



5. Click on ***Submit*** and press ***Yes.*** You are able to submit as many times as you want

You can test your code on your local machine in the way described in the homework handout. On vocareum, you only need to ***Submit,*** then the "***Passed/Failed***" message will appear automatically. You can check your submission results in the Terminal or under Details tab. If your code works correctly, you should see the results as below:

```
Submission Report
[Executed at: Sun Sep 8 18:32:08 PDT 2019]
================================================
blackbox21 passed, test accuracy: 90.0
================================================
```

Here, test accuracy is based on the test data given to you, so you can compare this with the results on your local machine to make sure everything works properly on Vocareum.

After the deadline, your ***final submission*** will be graded using hidden test data, and you will be able to see your grades on Vocareum.