Coding Assignment 1

Implement and train a network on MNIST

Overview

In this assignment, you will complete a simple pipeline of training neural network to recognize MNIST Handwritten Digits: http://yann.lecun.com/exdb/mnist/. You will implement two network architectures along with code to lad data, train, and optimize those networks. You are asked to complete a variety of experiments and complete a short report on your findings. ~~There is a report template to fill in your information and results.~~

The `main.py` contains declaration and set up of the neural network architectures. You will modify portions of this code to set up the two different networks you are experimenting with. The script contains hyper-parameters that you will tune to optimize the performance of your learners (e.g., `batch_size`, `learning_rate`, `hidden_layer_size`).

Python and dependencies

In this assignment we will work will Python 3. If you do not have a python distribution installed yet, we recommend installing Anaconda or (miniconda) with Python 3. We provide `environment.yaml` which contains a list of libraries needed to set up the environment for this assignment. You can use it to create a copy of the conda environment. Please refer to the user's manual for assistance.

Code Test

There are several tests provided in the assignment repository which you can run from the command line using

```
$ python -m unittest test.<name_of_tests>
```

Note that passing all local tests does NOT mean that your code is free of bugs, and it does NOT guarantee that you will receive full credit for the assignment. There are additional tests that the grader will use with does not present in your local unit tests

1 Data Preparation

So that your tuning of the hyper-parameters avoids overfitting, you are using a training data set and a testing set. The testing set is merely used to compute the performance of your final model and should not be used in making decisions about tuning your learner. So, it is common to split your TRAINING set into an actual training set and a validation data set and perform hyper-parameter tuning based on the results on validation data. Additionally, in

deep learning, training data is often forwarded to models in batches for faster training and noise reduction.

In our pipeline, we load the training and test set into the system. We also convert the labels from integers to a one-hot encoding. This means that if the label was, say '3', then the new label is a row [0,0,0,1,0,0,0,0,0,0]. The resulting label matrix will be Nx10 where N is the number of examples in the data set.

We then split the training set into an 80% training and 20% validation.

Therefore, your tasks are as follows:

- ☐ one-hot encode the labels in both load_mnist_trainval and load_mnist_testval.
- ☐ follow instructions in the code to complete load_mnist_trainval in utils.py for training/validation split

You can test your data loading functions by running:

```
$ python -m unittest tests.test_loading
```

2 Model Implementation

You will now implement two networks from scratch by coding their layers. Take some time to become familiar with the structure of the code. You will be implementing various layers that are to be inserted into your networks. The two models below demonstrate the interface with the NeuralNetwork class and how the layers are added. It is assumed that we are constructing a classifier using cross-entropy loss so the final layer in your networks will be the SoftmaxLayer.

Model 1

```
net = NeuralNetwork()
net.add(FullyConnectedLayer(input_size,num_classes))
net.add(SigmoidActivationLayer())
net.add(SoftmaxLayer())
```

Model 2

```
net = NeuralNetwork()
net.add(FullyConnectedLayer(input_size,hidden_layer_size))
net.add(ReLUActivationLayer())
net.add(FullyConnectedLayer(hidden_layer_size,num_classes))
net.add(SoftmaxLayer())
```

2.1 Activation Functions

- ☐ There are two activation functions needed for this assignment: ReLU and Sigmoid. Implement both functions as well as their derivatives in NN.py.

You can test your activation functions by running:

```
$ python -m unittest tests.test_activations
```

2.2 Loss Function

- ☐ Implement the Cross-Entropy Loss function NN.py

You can test your loss function by running:

```
$ python -m unittest tests.test_loss
```

2.3 Layer Implementations

Each layer that will be added to your network contains 3 methods or functions: forward, backward, and step.

The forward function computes the output of the layer during the forward pass of gradient descent.

- ☐ Implement the forward pass for each of the layers
    - SigmoidActivationLayer
    - ReLUActivationLayer
    - SoftmaxLayer
    - FullyConnectedLayer
      NOTE: In the Fully Connected Layer, we are computing with a weight matrix of dimension (input_size x output_size) and a bias term that is 1 x output_size. In numpy, when you add an m x n matrix to a 1 x n matrix, the second matrix is broadcast to each row. In other words, each row of the m x n matrix is added to the 1 x n vector.

The backward function computes the gradient during the backward pass

- ☐ Implement the backward pass for each of the layers
    - SigmoidActivationLayer
    - ReLUActivationLayer
    - SoftmaxLayer
    - FullyConnectedLayer

  Remember that you are computing the derivative of the output with respect to the input and multiplying this by the output_gradient (i.e., the value being passed backward). In other words, you are writing the automatic differentiation step for each of these Layers (or nodes) in the network.

You can test your forward and backward functions by running:

```
$ python -m unittest tests.test_layers
```

The step function updates the parameters in that layer using the gradients. These have all been implemented for you. Notice that the Sigmoid, ReLU, and Softmax do not have parameters so there is no update that needs to be performed.

3 The Neural Networks

The Neural Network class is already implemented for you so that you can add layers to your network. The add function adds them in order from front to back (left to right, in our diagrams from class). Again, we will be sure to add the Softmax layer last since we are computing Cross Entropy Loss.

The class also contains a predict function that will accept inputs of the given input-size and predict the output in the form of a single integer for each row of inputs. You should review the rest of the functions as well to gain a feel for how the NeuralNetwork class functions.

There is nothing that needs to be implemented here.

4 Visualization

It is always a good idea to track the training process by visualizing the learning curves. The NeuralNetwork class stores both training and validation results in instance variables such as train_loss_history and val_acc_history. Your task is to plot the learning curves by using these values. A sample plot can be found below in Figure 1.

- ☐ Implement plot_curves in utils.py. You'll get full credit as long as your plot makes sense and includes a legend, x and y axis labels, and a descriptive title.

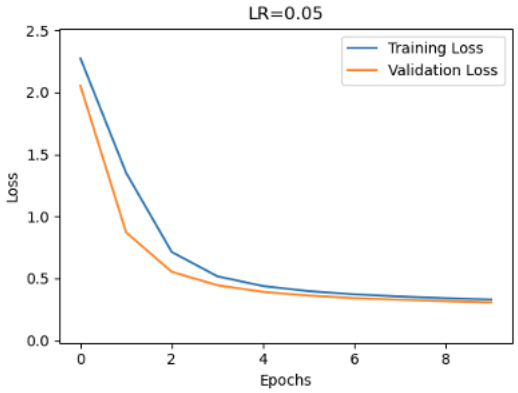

Figure 1

If everything above is successfully implemented, you should now be able to run main using

```
$ python main.py
```

You can try either version in the model implementation from the above Model 1 and Model 2. You also have the freedom to modify the hyper-parameters.

5 Experiments

It's time to play with your model a little. You will use your implementation of Model 1 and Model 2 to experiment with different combinations of your hyper-parameters.

Use as a starting set of parameter values:

```
batch_size = 64
num_epochs = 10
learning_rate = .001
hidden_layer_size = 128
```

- ☐ Try different value of learning rates for each model (after fixing the values of the other parameters) and report your observations
- ☐ Next, using Model 2, tune all 3 hyper-parameters (not including epochs). You may increase or decrease epochs as needed but you are to report on the "best" values of batch_size, learning_rate, and hidden_layer_size

6 Code and Write-Up

You will submit your code (main.py, NN.py, utils.py) as a zip file in Blackboard.

You will also need to submit a report summarizing your experimental results and finding as specified in Section 5. This report will be submitted as a pdf document in Blackboard as well.