Assignment 1 - AISE 3010

Arnav Goyal - 251244778

February 6, 2024

The Submission

My submission includes three python files.

- `classes.py`
- `classification.py`
- `regression.py`

The `classes.py` script includes the class object used to instantiate the neural network, and a helper function to one hot encode labels. The `classification.py` script tackles the MNIST classification problem using a fully-connected mlp network. The `regression.py` script tackles the california housing regression problem from scikit-learn (but doesnt work very well). Due to this, the rest of the report will be discussing the classification network (unless stated otherwise).

Note: The classification script first trains, outputs a trainining curve (which must be closed to proceed) and then proceeds to test while printing the test accuracy for each test batch. It then performs a average of all batches and prints the training accuracy

The Network

The network architecture is as follows.
Note: In this context, the word 'size' refers to the number of neurons within each layer.

- size 784 input layer (in)
- size 10 hidden layer (h1)
- size 10 hidden layer (h2)
- size 10 hidden layer (h3)
- size 10 output layer (out)

The input size and output size have been chosen according to specification of the dataset and classification problem. Namely, the input size is determined by the number of features (in this case pixels) of the 28 x 28 image. In this case we treat every pixel value as a feature, thus resulting in a need for a 784 size input layer. The output size has been chosen to conform to the task of classification, n-class classification requires an n dimensional output, where the index of the maximum value in the output is taken as the predicted class label. MNIST has 10 classes (for digits 0 to 9) thus we require a 10 dimensional output.

The network contains one hyperparameter: the learning rate, α which is used to determine the magnitude of the weight and bias updates during gradient descent. A generalized example of this update is shown below. It is currently set to $\alpha = 0.02$

$$W_{t+1} := \alpha \cdot \nabla W_t$$

$$B_{t+1} := \alpha \cdot \nabla B_t$$

Analysis & Improvement

As of submission, the `classification.py` script stops training around a minibatch train accuracy of approximately 60-70%. This training amount results in a good generalization of approximately 62-68% accuracy on the test set. Considering that guessing randomly would result in a test accuracy of around 10% this is a major upgrade. These numbers can be improved by doing a couple things.

- Increasing training time
- Optimization of the learning rate, α
- Increasing training set size

The most obvious way to improve a neural network at this stage is to increase the training time, this is a viable option as the training curve seems to still want to decrease before we stop providing it with mini-batches. We must be careful as to not overfit this though, thus we must pay careful attention to the gap between training accuracy and test accuracy while increasing training time.

The next method is to optimize the value of α. I have done this experimentally, which means that this value is (almost certainly) not the optimal value for the learning rate. The learning rate has a major impact on training metrics and optimization of this hyperparameter will improve the results. Lastly, we could allocate more data to the training set, which would allow the model to potentially learn a bit more than what is currently available. Once again, special care must be taken to avoid overfitting.

```
Batch 39 -  Train Accuracy: 0.726
40
40
Testing
Batch 40 -  Test Accuracy: 0.702
Batch 41 -  Test Accuracy: 0.688
Batch 42 -  Test Accuracy: 0.6906666666666667
Batch 43 -  Test Accuracy: 0.7393333333333333
Batch 44 -  Test Accuracy: 0.7653333333333333
Batch 45 -  Test Accuracy: 0.7813333333333333
Batch 46 -  Test Accuracy: 0.48733333333333334
Total Average Test Accuracy: 0.6934285714285713
PS C:\Users\arnav\Documents\GitHub\university\Winter 2023-2024\AISE 3010\Assignment 1>
```

Figure 1: Test Accuracy Output of `classification.py`

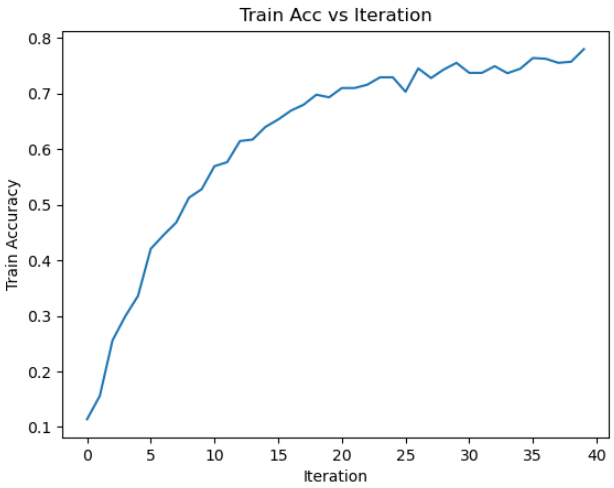

Figure 2: Training curve from `classification.py` (super high this time!)

Regression

For regression I used the *california housing dataset from scikit-learn*. It involves 8 features with 1 continuous response variable:

- MedInc - median Income in block group
- HouseAge - median house age in block group
- AveRooms - avg number of rooms in block group
- AveBedrms - avg num of bedrooms per block group
- Population - block group population
- AveOccup - avg num of household members
- Latitude - block group latitude
- Longitude - block group longitude

The response variable is the mediuan house value, expressed in units of $100,000. In other words a target value of 1.7 means $170,000.

The purpose of this section is to bring up the non-functionality of the regression task contained in `regression.py`. This script is basically functional but something within the neural network class (probably backpropagation) is preventing the weight and bias updates from being meaningful. In other words, the network isnt really training.

This can be seen upon running the script and observing the (extremely ugly!) training curve. The training curve suggests that the mean squared error (MSE) is not decreasing, hinting to a problem with the learning method (backpropagation) of the neural network. The MSE achieved on the training curve is quite high considering the small scale of the target variable.

This being said, i'm not entirely sure on how to fix it. It also prints a LOT of overflow errors, im not sure if these two problems are entirely related. Regardless, I've attached my work so far within the script (I tried my best).

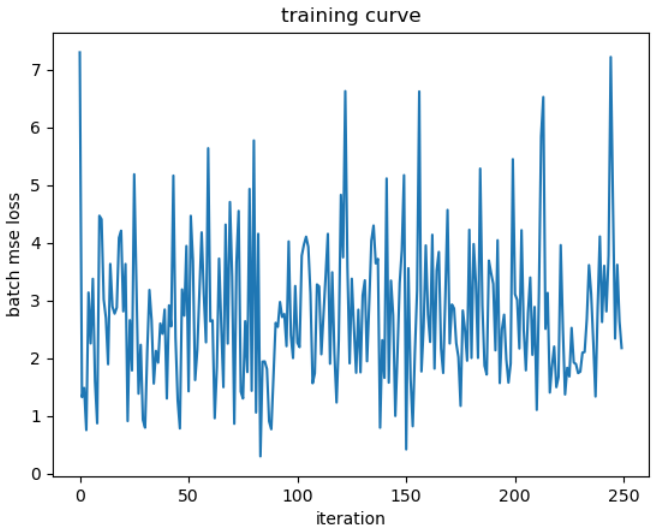

Figure 3: Training 'curve' of the `regression.py` script suggesting backprop issues