TP1 Homework Submission

Vincent, Wilmet
vincent.wilmet@student-cs.fr

January 22, 2022

1 Problem 1: Multiclass

This question asks us to find hyperparamters for a custom neural network model that can outperform a base Linear model on multi-classification of digits. We use the data provided by the United States Postal Service (USPS) dataset, which contains 7291 samples of envelopes automatically scanned by the USPS.

1.1 Model Architecture and Performance

The model architecture I used for this problem was a Convolutional Neural Network (CNN) with 4 Conv2d layers and 2 linear layers. I found that this model performed much better than the alternatives (Linear, [2,3,5,10] conv2d layers).
However, the model greatly depends on the number of epochs, criterion, learning rate, batch size, optimizer and drop out rate. I go into more detail in section @3.2, but the base model had a performance of 89.3% on the test set, and my best model with 0.01 learning rate, Adam optimizer, 10 images per batch, CrossEntropyLoss and 0.1 drop out rate had a performance of 93.42%.

2 Problem 2: Regression

This question asked us to look at housing prices and create a model that best predicts the house selling price. We are given a .csv file with 1460 samples.

2.1 Training Process

Initially, without any modifications to the given code, the model performs very poorly. It gives NaN values as an output for both training loss and validation accuracy. With this in mind, I decided preprocess the data for better model performance. I did this by trying sklearn methods like MinMaxScaler, StandardScaler and normalize with l1, l2, and max norms. Since we don't have any categorical data, there was no need to do OneHotEncoding.

2.2 Model Architecture and Performance

Following the preprocessing of the data, and the suggestion of the professor in lectures, I have implemented a 2 layer network which is sufficiently deep to show noticeable improvements.
To be concrete, the model with MSE and Linear Regression yields a training error of 0.0147 and validation error of 0.0163. My custom model with 16 neurons in both layers and GaussianNLLLoss yields a training error of 0.0039 and validation error of 0.0106. The loss on the unseen test set is 0.0071.

3 Global Discussion

3.1 Hyperparameters and GridSearch

The following are the hyperparameters applied to both problems:

```
num_epochs = [2, 5, 10, 50]
batch_sizes = [10, 100, 500, 1000]
criterions = [nn.MSELoss(), nn.CrossEntropyLoss(), nn.GaussianNLLLoss]
drop_outs = [0.1, 0.25, 0.5]
lr = [0.1, 0.05, 0.01, 0.001, 10]
optimizers = ["Adam", "RMSProp", "SGD"]
```

Because GridSearch is not implemented in PyTorch, I used a very oldschool technique: nested for-loops for each of the hyper-parameters listed above.
Ultimately, this took about 27 hours to run since at some point I had exceeded the monthly allocation for Google Colab's GPUs. In all there are 4*4*3*3*5*3=2160 possible configurations. Thus the trade-off between training one model and time is about 27*60/2160*60=45 seconds per model. I would normally avoid spending this much time to train every combination of the network hyperparameters, but I can understand the value for the more rigorous researchers. For future models, I would attempt pruning techniques or model checkpoints to save at [2,5,10] epochs rather than spending the time retraining the same first 2 epochs in each of larger sized variations.

3.2 Discussion

Overall, I found that if the number of epochs was too small, e.g. 2-5 the model does not reach a point of convergence and achieves only 15.65% accuracy. If the batch sizes are large then the model takes longer to converge but arrives to 17.89%. The best criterion for solving this problem ranked was 1. GaussianNLL, 2. MSE 3. Cross Entropy. The best optimizer for solving this problem ranked was 1. Adam, 2. RMSProp, 3. SGD. Initially, I tried to solve these rankings by averaging their scores but I found that they were heavily influenced by the low scoring outliers. Instead, I aggregated their relative rankings per hyperparameter variation and averaged those instead. Strangely, 0.1% and 0.5% drop out rate performed about the same, but 0.25% did much more poorly. I could

not find a reason for why this would be the case. Finally, the most impactful parameter was the learning rate, which I found that the smaller, the better. Ranging from 8.82% at 10 to 89.7% at 0.001. Big values like 1 or 10 would never work well. On any model for that matter.
When looking at the relation between variance ($\sigma(x)$) and Overall Quality, there is a clear trend. The volatility of Sales Price will increase and develop a a larger range of Sales Prices if the (variance of) Overall Quality increases. Any input with larger Overall Quality produces a $\sigma(x)$ that is also large.