CS6375 Assignment 1

https://github.com/Fetlach/assignment1

Joshua Martin

Net ID: JCM240004

# Introduction and Data (5pt)

The main experiments done were the testing of various hyperparameter permeations on the training of both the FFNN and RNN implementations. These, when varied, correlate strongly with the resulting accuracy of the neural network compared to the validation data and the training speed of the neural network.

My results concluded that while the FFNN was better able to model the data in training, the RNN sometimes showed better results when tested on data outside of its training depending on the number of hidden dimensions specified when invoking the program.

The data provided is a list of yelp reviews following the format of {text, stars} where the text is the contents of a given review and the stars are a floating point value. Of these there is a single training, validation, and test set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Set name | 1 star reviews | 2 star reviews | 3 star reviews | 4 star reviews | 5 star reviews |
| Training | 3200 | 3200 | 3200 | 3200 | 3200 |
| Validation | 320 | 320 | 160 | 0 | 0 |
| Test | 0 | 0 | 160 | 320 | 320 |

The task, sentiment analysis, falls under the category of classification.

# Implementations

## FFNN

A screenshot of a computer program

AI-generated content may be incorrect.

The forward() method in the FFNN class - as described in the pytorch documentation – is overloaded to define how a node transforms an input into an output.

The W1 and W2 functions are used to transform and format the input according to the node’s hidden layer and then into the output’s desired format. Essentially, these functions serve as an IO interface for the data traveling between nodes.

After the W1 function and before the W2, I use the nn.Relu() activation function to determine whether or not to provide output from the node mathematically.

To obtain the probability distribution, I then used the softmax function declared in the class init. This wrapper of the [logSoftMax function](https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html) returns a copy of the input tensor on a logarithmic scale after applying the softmax statistical function to create a vector of probabilities for the next hidden layer. Softmax is added after the W2 function as the sentiment analysis task we’re performing requires multi-classification, and the logarithmic version of the function is used to avoid error accumulation from floating point arithmetic.

The remaining functions in the file, not including main, act as helpers to vectorize the training data so that it is in a format usable by the neural network. This process uses the JSON library to load the data and then follows a process of parsing (make data) -> defining a dict from the vocab (make\_indices) -> vectorizing (convert\_to\_vector\_representation) -> and finally training.

The optimizer in the training periodically modifies a node’s parameters by calling the step() function. The calls to zero\_grad() exist to “zero out” the gradient of nodes between optimization steps, as shown by their position at the start of loops.

A screenshot of a computer

AI-generated content may be incorrect.

The initialization inputs are for the stochastic descent gradient (sgd) which takes the list of model parameters, the learning rate (0.01 as opposed to default of 0.001) and the learning momentum (default 0).

A bulk of the remaining main code initializations are fairly self-explanatory.

* Random is seeded at “42” to ensure that the random number generator produces the same outputs results each run. Presumably, this is to make debugging easier.
* “model” is initialized as a feed-forward neural network with the class defined at the start of the file, with the vocabulary driving the vector length.
* Loss is initially set to “none” as a default, which is later used by backward() to backwards propagate the gradient.

Finally, I modified the end of the main code block to include a test output pass to see how the model performs on an input after being trained.

## RNN

A screen shot of a computer program

AI-generated content may be incorrect.

While this implementation of the forward() method is mostly the same as the FFNN version, the key difference is the retrieval of the output layer. This output layer serves as memory of the last state and thus allows the “recurrence” part of the neural network to take place. The initial hidden layer is now gotten by the rnn function, and the predicted vector is derived from the previous entry.

The helper function for loading the input data is the same as the FNN version, although the functions for vectorizing said data after it has been loaded are notably missing. This seems to instead be done directly within the main code block.

However, some things change in the main code of the RNN version:

* The optimizer used is now “Adam” by default, as opposed to SGD in the FFNN version.  
  Adam is also a stochastic gradient optimizer, so the initialization’s purpose is mostly the same.
* The pickle library is used to retrieve a word embedding file, which is then used to create a list of vectors.
* The iteration of the RNN training is based on a stop condition rather than in FFNN where it is a finite for loop, which reflects the “recurrence” aspect of how this neural network operates.

# Experiments and Results

**Evaluations**

To evaluate the models I looked at how well the given model predicts its own training data, its validation data, and the test data. This is done as a benchmark to determine three factors:

1. How accurately the model can train to its training data in a number of epochs
2. How accurately the model extends to the training validation data after a number of epochs
3. How accurately the model extends to outside test data after being trained

These metrics are put in context by a varied number of hidden layers used in training with a constant epoch count.

In addition to these metrics, I also tracked variables for training speed to provide an insight into the amount of real time used for computation in each model’s training. These include the minimum and minimum times spent training an epoch along with the total time taken to train the model.

**Results**

FFNN Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| H Layers | Epochs | Training Time (s) | Min Epoch Time (s) | Max Epoch Time (s) | Training Accuracy (%) | Validation Accuracy (%) | Test Accuracy (%) |
| 8 | 30 | 664.132 | 19.646 | 30.317 | 0.855 | 0.497 | 0.497 |
| 16 | 30 | 1132.419 | 35.312 | 42.411 | 0.928 | 0.463 | 0.541 |
| 24 | 30 | 1548.238 | 47.827 | 58.041 | 0.922 | 0.483 | 0.558 |

As expected, this shows that the number of H Layers and Epochs generally correlate with the time needed to train and the overall accuracy for validation and training. The FFNN reached 0.8+ training accuracy very fast, but as demonstrated it does not generalize its training very well only being able to achieve 0.56- accuracy in both validation and testing.

RNN Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| H Layers | Epochs (until overfit) | Training Time (s) | Min Epoch Time (s) | Max Epoch Time (s) | Training Accuracy (%) | Validation Accuracy (%) | Test Accuracy (%) |
| 1 | 2 | 384.703 | 189.134 | 195.569 | 0.263 | 0.298 | 0.341 |
| 2 | 2 | 343.476 | 168.794 | 174.681 | 0.288 | 0.36 | 0.316 |
| 3 | 5 | 839.969 | 161.708 | 172.465 | 0.374 | 0.398 | 0.386 |

While the RNN on average took much longer to train per epoch compared to the FFNN it did begin fitting the data much more quickly.

The most notable difference in the RNN is that it takes substantially less hidden layers to operate due to how the neural network is structured. While the number of epochs is also less predictable due to the tendency to begin overfitting, there is a much stronger correlation between the dataset accuracies as compared to the FFNN.

Additionally, I saved the following data from training sessions wherein I used a bad training data set. The bad training data set was missing half the three star reviews and all of the four and five star reviews. This provides a good insight into how that data negatively influences the model’s training.

FFNN Results (bad training data)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| H Layers | Epochs | Training Time (s) | Min Epoch Time (s) | Max Epoch Time (s) | Training Accuracy (%) | Validation Accuracy (%) | Test Accuracy (%) |
| 8 | 30 | 208.328 | 4.684 | 7.775 | 0.896 | 0.568 | 0.08 |
| 16 | 30 | 409.462 | 10.256 | 34.620 | 0.943 | 0.587 | 0.08 |
| 24 | 30 | 548.501 | 16.230 | 20.620 | 0.956 | 0.568 | 0.076 |
| 8 | 15 | 104.337 | 5.556 | 7.548 | 0.735 | 0.595 | 0.051 |
| 16 | 15 | 187.725 | 11.372 | 13.804 | 0.807 | 0.602 | 0.076 |
| 24 | 15 | 318.114 | 17.227 | 36.339 | 0.78 | 0.586 | 0.101 |

RNN Results (bad training data)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| H Layers | Epochs (until overfit) | Training Time (s) | Min Epoch Time (s) | Max Epoch Time (s) | Training Accuracy (%) | Validation Accuracy (%) | Test Accuracy (%) |
| 1 | 9 | 945.782 | 103.046 | 107.637 | 0.496 | 0.472 | 0.141 |
| 2 | 4 | 373.404 | 90.146 | 100.922 | 0.487 | 0.525 | 0.00125 |
| 3 | 2 | 195.508 | 96.782 | 98.726 | 0.476 | 0.426 | 0.04875 |

This data demonstrates the problems that can arise when training for and testing against data that is too strongly or too weakly correlated. While we can expect different accuracies from the validation and test data against the training data due to how neural networks operate, this was far outside expectations. In fact, I had initially written here in my writeup before correcting the issue that there must have been a mismatch in data contexts due to the lack of correlation between the test results.

Notably, in both models the validation accuracy is somewhat inflated while the test accuracy is significantly diminished.

Interestingly, the RNN initially started out with and maintained a much higher training accuracy between the training and validation data sets during the training process and was generally more influenced by the bad data than the FFNN. This was due to the classification task being limited to only three tiers of reviews (1, 2, and 3 stars) instead of the full 5 classes as the 4 and 5 star reviews happened to be absent from both the bad training dataset and the validation set.

# Conclusion and Others

I had no teammates during this assignment, so I was responsible for all aspects during its completion.

One issue I had with the project was the different versions of training data provided, one in the zipped folder and the other not which can be easily mistaken if the file sizes are overlooked. The one not contained in the zipped folder lacked half as many 3 star reviews, and no 4 or 5 star reviews at all compared to the zipped folder which had 3.2k reviews for each category. I had to redo most of my trials due to this mistake in not addressing the data first which led to much confusion initially.

Aside from that issue, the rest of the assignment was alright in terms of workload. The time loss from testing was substantial, though there is not much that could be done to remedy that outside of use of 3rd party equipment. The best suggestion for improvement I can provide is that a hint to the range of expected hyperparameter and training values would make the assignment much more approachable as it is difficult to begin determining if the forward function is correct or not when the output cannot provide any clues.

It should also be mentioned that in the case of operating on a machine with windows OS, substantial effort is needed to create a working project environment due to grappling with WSL.