CS6320 Assignment 2

https://github.com/nishd8/rnn-fnnn

|  |  |  |
| --- | --- | --- |
| Group21 | Nishad Raisinghani  NXR200053 | Ishwari Joshi IGJ180000 |

# Introduction and Data (5pt)

**TODO:** Briefly describe the project and your main experiments and results, including mentioning the data you use.

**TODO:** Briefly describe task and data (e.g. how many examples are in the training, development, and test sets.), it is best to report all the statistics, including counts, in a table. including how you treat casing, tokenization, and anything else that you did to the raw data before computing features from it.

In this project a Feedforward Neural Network (FFNN) and a Recurrent Neural Network (RNN) are implemented for performing a 5-class Sentiment Analysis task. The data used is in 3 json files: training.json, validation.json, and test.json. The json files have two key-value pairs for each review. Key-value pairs are “text”: “text from review” and “stars”: rating from {1.0, 2.0, 3.0, 4.0, 5.0}. The main experiment is to train the models (FFNN and RNN) with different hyperparameters (hidden layer dimension and number of epochs) and evaluate the models based on accuracy. There are 8000 examples in the training set, 800 examples in the validation set, and 800 examples in the test set. The data given is used as is. No further preprocessing is done. The data is not lowercased, and punctuation is not removed. The data loading is done in the load\_data function where the data is loaded and put into lists in which each element is (the text split by whitespace, rating minus 1). The rating minus 1 is used to align with Python’s zero-based indexing.

# Implementations (45pt)

## FFNN (20pt)

**TODO:** Explain briefly how you implemented filled in the incomplete code for FFNN.py in the form of screenshot (and explanations) in the report. Provide any other libraries/tools that are used; tutorials/materials that you referred to; or how you were doing debugging. Try to understand what other part of the code is doing, and write your understandings here (e.g. optimizers, initializations, stopping, etc.)

The forward computation part in forward() is filled in to implement a 1 hidden layer FFNN with ReLU activation. The \_\_init\_\_(self) function has two parameters: the input dimension and the hidden layer dimension. Layers and parameters are defined in the \_\_init\_\_(self) function. Linear(n,m) is a PyTorch function that creates a single layer feedforward network with n inputs and m outputs. There is 1 Linear hidden layer W1 in which Linear has two parameters: the input dimension (size of vocabulary) and the hidden unit dimension. There is a ReLU activation function that uses the ReLU() function. The Linear output layer W2 has a dimension of 5 (for 5 possible ratings). For W2, Linear has two parameters: the hidden unit dimension and the output layer dimension.

A screenshot of a computer code

Description automatically generated

Figure 1: Implementation of forward() function for FFNN

Figure 1 shows how the forward() function is implemented. (1) The hidden layer representation is obtained by calling W1 with the input vector. The hidden layer is passed through the ReLU activation function (let this be hidden\_layer\_rep). (2) The output layer representation is obtained by calling W2 with hidden\_layer\_rep. (3) The probability distribution is obtained by passing the output layer through softmax.

The optimizer used is Stochastic Gradient Descent with the FFNN model parameters, a learning rate of 0.01, and a momentum of 0.9. For each epoch, after the model is trained, the optimizer’s gradients are cleared for every parameter x in the optimizer (optimizer.zero\_grad()). This is again done for each minibatch of size 16 that the data is split into. Each example in the minibatch is iterated through. Before the next minibatch starts, dloss/dx is computed for every parameter x and these are accumulated into their respective gradient (loss.backward()) and the value of x is updated using its gradient (optimizer.step()).

## RNN (25pt)

**TODO:** Explain briefly how you implemented filled in the incomplete code for RNN.py in the form of code-snippet screenshot (and explanations) in the report. Provide any other libraries/tools that are used; and tutorials/materials that you referred to. Try to understand what other part of the code is doing, and write your understandings here (especially parts that is functioning differently as compared to FFNN).

The forward computation part in forward() is filled in to implement a 1 hidden layer RNN with tanh activation. The \_\_init\_\_(self) function has two parameters: the input dimension and the hidden layer dimension. Layers and parameters are defined in the \_\_init\_\_(self) function. RNN(n,m) is a PyTorch function that creates a recurrent network with n inputs and m hidden units using numOfLayer hidden layers (in this case 1) and a nonlinearity function (in this case tanh). There is 1 Linear output layer W in which Linear has two parameters: the hidden unit dimension and the output dimension of 5.

A screenshot of a computer

Description automatically generated

Figure 2: Implementation of forward() function for RNN

Figure 2 is how the forward() function is implemented. (1) The hidden layer representation is obtained by calling the RNN function with the inputs (let this be hidden\_layer\_rep). (2) The output layer representation is obtained by calling W with hidden\_layer\_rep. (3) The sum over the output is obtained (the vectors for each token of the output layer is summed as a representation for the entire sequence). (4) The probability distribution is obtained by passing the summed output layer through softmax.

The optimizer used is Adam with the RNN model parameters and a learning rate of 0.01 which is different than the one used in FFNN. Word embeddings are used for initialization of word representations rather than the bag-of-words vectorization in FFNN. For each review in the data, the input vector is joined by space to get the words, punctuation is removed, and each word’s embedding is looked up and used. Instead of running over all epochs like in FFNN, RNN has a stopping condition. By keeping track of the last validation accuracy and last training accuracy (the previous epoch), the model checks if the validation accuracy is less than the last validation accuracy and if the training accuracy is greater than the last training accuracy. If this is true, the model stops. In this way, training is done to avoid overfitting and the last validation accuracy is the best one obtained.

# Experiments and Results (25pt)

**Evaluations (5pt) TODO:** Explain how you evaluate the models. What metric is used – you can refer to the current implementation.

The models are evaluated by accuracy. To use the model, we pass it the input vector (FFNN) or word embedding (RNN). This executes the model’s forward() implementation. The gold label is obtained directly from the data and the predicted label is obtained from the maximum value from the model’s forward pass. From here, we can check if the predicted label is equal to the gold label. Accuracy is then the number of correct divided by the total.

**Results (20pt) TODO:** Apart from the default hyperparameters, try multiple variations (between 1-2 for FFNN and RNN each) of models by changing hidden unit sizes.

**TODO:** Summarize the performance of your system and Put the results into tables or diagrams and include your observations and analysis.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hidden Layer Dimension | Total Epochs | Current Epoch | Training Accuracy | Validation Accuracy |
| 5 | 10 | 1 | 0.66475 | 0.6225 |
| 5 | 10 | 2 | 0.68925 | 0.6 |
| 5 | 10 | 3 | 0.6895 | 0.5775 |
| 5 | 10 | 4 | 0.714625 | 0.53125 |
| 5 | 10 | 5 | 0.7325 | 0.53875 |
| 5 | 10 | 6 | 0.751375 | 0.58875 |
| 5 | 10 | 7 | 0.769125 | 0.615 |
| 5 | 10 | 8 | 0.763 | 0.62125 |
| 5 | 10 | 9 | 0.798875 | 0.6025 |
| 5 | 10 | 10 | 0.7825 | 0.56875 |

Table 1: Results of FFNN model with hidden layer dimension 5 and 10 epochs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hidden Layer Dimension | Total Epochs | Current Epoch | Training Accuracy | Validation Accuracy |
| 10 | 10 | 1 | 0.703375 | 0.59375 |
| 10 | 10 | 2 | 0.7225 | 0.595 |
| 10 | 10 | 3 | 0.729375 | 0.62 |
| 10 | 10 | 4 | 0.754625 | 0.60625 |
| 10 | 10 | 5 | 0.7875 | 0.58625 |
| 10 | 10 | 6 | 0.791125 | 0.6225 |
| 10 | 10 | 7 | 0.783375 | 0.585 |
| 10 | 10 | 8 | 0.783 | 0.535 |
| 10 | 10 | 9 | 0.77075 | 0.56875 |
| 10 | 10 | 10 | 0.824375 | 0.55125 |

Table 2: Results of FFNN model with hidden layer dimension 10 and 10 epochs

By fixing the values of learning rate = 0.01, momentum = 0.9, and minibatch size = 16, we are able to observe only the effect of hidden layer dimension on training and validation accuracy. The FFNN model is run with hidden layer dimensions 1, 5, 10, and 25 and 1, 5, 10, and 25 epochs. Here, two runs are shown to observe the effect of changing hidden layer dimension on accuracy over the same number of epochs. As hidden layer dimension increases, validation accuracy is highest at a higher number of epochs. Validation accuracy for the hidden layer dimension of 5 is maximum (0.6225) at epoch 1 while validation accuracy for the hidden layer dimension of 10 is maximum (0.6225) at epoch 6. Increasing the hidden layer dimension further to 25 (over 10 epochs) results in the highest validation accuracy at epoch 1 (0.6325), so comparing different models shows different observations. As the current epoch increases, the difference between training and validation accuracy increases since training accuracy continues to increase over the epochs.

Of the two models above, the one with a hidden layer dimension of 10 performs better even though its validation accuracy peaks at a later epoch. This is because the training accuracy at epoch 1 for the model with a hidden dimension layer of 5 is 0.665 compared to training accuracy of 0.791 for the model with a hidden dimension layer of 10. The model with a hidden dimension layer of 5 is undertrained while the model with a hidden dimension layer of 10 has an acceptable training accuracy that shows the model is sufficiently trained. The test accuracy for the Table 1 model is 0.329 and for the Table 2 model is 0.282. The low test accuracy is because ?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch | Training Accuracy (10 hidden dim) | Training Accuracy (20 hidden dim) | Validation Accuracy (10 hidden dim) | Validation Accuracy (20 hidden dim) |
| 1 | 0.468 | 0.46 | 0.545 | 0.483 |
| 2 | 0.487 | 0.494 | 0.489 | 0.501 |
| 3 |  | 0.504 |  | 0.515 |
| 4 |  | 0.487 |  | 0.538 |
| 5 |  | 0.501 |  | 0.359 |

Table 2: RNN Results (Run for 10 Epochs)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch | Training Accuracy (10 hidden dim) | Training Accuracy (20 hidden dim) | Validation Accuracy (10 hidden dim) | Validation Accuracy (20 hidden dim) |
| 1 | 0.455 | 0.466 | 0.436 | 0.319 |
| 2 | 0.492 | 0.504 | 0.414 | 0.516 |
| 3 |  | 0.491 |  | 0.516 |
| 4 |  | 0.508 |  | 0.519 |
| 5 |  | 0.406 |  | 0.47 |
| 6 |  | 0.45 |  | 0.471 |
| 7 |  | 0.461 |  | 0.466 |

Table 3: RNN Results (Run for 20 Epochs)

The validation accuracies for the RNN runs above are low. When run for 10 epochs, the model stops after 2 epochs for hidden unit size of 10 and stops after 5 epochs for hidden unit size of 20. When run for 20 epochs, the model stops after 2 epochs for hidden unit size of 10 (same as run for 10 epochs) and stops after 7 epochs for hidden unit size of 20. The validation accuracy is higher over the 10 epoch run than the 20 epoch run. The validation accuracy is higher for hidden unit size of 10 for the 10 epoch run while it is higher for hidden unit size of 20 for the 20 epoch run. From observing the results, we can infer that a greater hidden unit size needs to run for more epochs.

**TODO:** (Extra Bonus), try other variations of the model by changing the default code and report resutls, e.g. you can try different number of layers, initializations, et

# Analysis (20pt)

Figure 3: Learning Curve for Best System

The best system for FFNN is the model with a hidden layer dimension of 10 and 10 epochs. Other considerable models have hyperparameters hidden layer dimension of 10 and 5 epochs as well as hidden layer dimension of 25 and 5 epochs; however, these two have training accuracy around 0.6-0.7 which could be improved with more training. A “good” system has increasing training accuracy over the epochs and validation accuracy that peaks at some epoch before decreasing as shown in Figure 3.

**TODO:**

* (10pt) Plot the learning curve of your best system. The curve should include the training loss and development set accuracy by epoch.
* (10pt) Error analysis. List some (one or more than one) error examples and provide some analysis. How might you improve the system?
* (Extra Bonus) other analysis and discussions.

# Conclusion and Others (5pt)

**TODO:**

* Individual member contribution.
* Feedback for the assignment. e.g., time spent, difficulty, and how we can improve