CS4375 Assignment 2

https://github.com/ThePublicCheese/CS4375-Assignment-2

Zane Royal Taylor

ZRT220000

**1 Introduction and Data**

Using a dataset of Yelp reviews which are associated with a rating y ∈ Y := {1, 2, 3, 4, 5}, this project applies Feedforward Neural Networks and Recurrent Neural Networks to perform sentiment analysis, aiming to predict review ratings based on the text. Due to using both methodologies, it is easy to compare the runtime and training efficiency of each method in real time. The Yelp dataset given is divided into training, validation (development), and test sets stored in JSON files. The models are trained using “training.json”, tuning and evaluation is done by “validation.json”, and final testing is done with “test.json”.

|  |  |
| --- | --- |
| Dataset | Number of Examples |
| training.json | 8000 Reviews |
| validation.json | 800 Reviews |
| test.json | 800 Reviews |

**2.1 FFNN**

The following changes were made to the feedforward neural networks forward function:

  
First, an intermediate representation of W1 is created by intertwining it with the input vector. This step is used to capture relationships between input features. Then ReLU is used to introduce nonlinearity within the input data so that the model will learn complex patterns and relationships within its own data.

  
Next, the previous output h1 is transformed into a five-dimensional vector. This vector represents raw confidence scores for each of the five possible values.

  
Softmax is applied to convert the confidence scores into a normalized probability distribution. This sums the output values to 1, which represents the likelihood of each rating.

  
Finally, the final distribution is returned.

**2.2 RNN**

The following changes were made to the Recurrent Neural Networks forward function:

  
initializes an embedding layer  
  
  
Converts inputs into corresponding word embeddings

  
Embeddings are processed and unpacked into output and hidden using self.rnn, output holds timestep data, whereas hidden contains the hidden state after each sequence.

  
Output is summed into a single dimension which stores all the information across all timesteps into output\_sum.

  
By averaging output\_sum and our last hidden state, we can balance the contributions from the overall sequential representation of the data and the final hidden state.

  
The average is then transformed using the output weight matrix, mapping the average to the output space alongside raw unnormalized scores.

  
Softmax is then used to convert the confidence scores into a normalized probability distribution. This sums the output values to 1, which represents the likelihood of each rating.

  
Finally, the final distribution is returned.  
  
  
The following changes were made in various functions within the Recurrent Neural Network:

Word Embedding-

A screen shot of a computer code

Description automatically generated  
This block iterates through an individual review, lowercasing, removing punctuation and isolating words which are then stored into vocab. After indexing is finished, the total amount of words is output.

  
These lines of code create a dictionary, mapping every word unknown or within vocab to a random value and 50-dimensional vector.

  
This saves the dictionary as a pickle file for later re-loading.

Embedding layer-

  
This loads the pickle file that was previously created.

A black background with white text

Description automatically generatednn.Embedding creates an embedding layer initialized with the word vectors previously created, the layers weights are set to the embedding values so that we may immediately use them and “embedding\_layer.weight.requires\_grad = True” is used so that the embeddings are tuned during training.

A computer screen shot of text

Description automatically generated  
Saves our word embeddings.

**3 Experiments and Results**

**3.1 Evaluations**

During training, the models are evaluated on the validation set after each epoch to monitor their generalization performance. The test set, which contains 800 unseen reviews, is used to assess the final performance of the models once training is complete. By contrasting the training and validation accuracy at each epoch we can understand how the program is training in real time.

**3.2 Results**

The smaller hidden dimensions for FFNN, the shorter the training time. The larger the hidden dimensions for FNNN, the longer the training time. Validation and accuracy stayed around the same values no matter how small or large the hidden dimensions were.

There was no noticeable accuracy and validation change when using a small hidden dimension for RNN, it took a slightly shorter time to train. There was also no noticeable accuracy and validation change using a large hidden dimension for RNN, there was a slight increase in training time. Overall, changing the hidden dimensions for RNN caused little to no change. Such little change was noticed in fact, I am worried that the hidden dimensions did not change although specified to.  
  
 For FFNN, the training accuracy performs better the more epochs are ran, bottlenecking at around 90% training accuracy and keeping around 50-60% validation accuracy within 50 epochs. This is shown in the table below in 5 epoch intervals:

|  |  |  |
| --- | --- | --- |
| Epoch 1 | Training accuracy for epoch 1: 0.513 | Validation accuracy for epoch 1: 0.55875 |
| Epoch 5 | Training accuracy for epoch 5: 0.64075 | Validation accuracy for epoch 5: 0.57 |
| Epoch 10 | Training accuracy for epoch 10: 0.7595 | Validation accuracy for epoch 10: 0.57 |
| Epoch 15 | Training accuracy for epoch 15: 0.817375 | Validation accuracy for epoch 15: 0.60875 |
| Epoch 20 | Training accuracy for epoch 20: 0.820125 | Validation accuracy for epoch 20: 0.49625 |
| Epoch 25 | Training accuracy for epoch 25: 0.771 | Validation accuracy for epoch 25: 0.48125 |
| Epoch 30 | Training accuracy for epoch 30: 0.851125 | Validation accuracy for epoch 30: 0.54875 |
| Epoch 35 | Training accuracy for epoch 35: 0.8505 | Validation accuracy for epoch 35: 0.58125 |
| Epoch 40 | Training accuracy for epoch 40: 0.888 | Validation accuracy for epoch 40: 0.57875 |
| Epoch 45 | Training accuracy for epoch 45: 0.92575 | Validation accuracy for epoch 45: 0.5925 |
| Epoch 50 | Training accuracy for epoch 50: 0.89125 | Validation accuracy for epoch 50: 0.58875 |

As for RNN, we need to run as many training sessions as possible to get the best trained file, which after many runs, will begin to improve validation and training accuracy

With a new pkl file

|  |  |  |
| --- | --- | --- |
| First run | Training accuracy for epoch 1: 0.524625 | Validation accuracy for epoch 1: 0.605 |
|  | Training accuracy for epoch 2: 0.668875 | Validation accuracy for epoch 2: 0.60625 |
| Overfit! | Training accuracy for epoch 3: 0.754875 | Validation accuracy for epoch 3: 0.5825 |
| Second run | Training accuracy for epoch 1: 0.536 | Validation accuracy for epoch 1: 0.58625 |
| Overfit! | Training accuracy for epoch 2: 0.669625 | Validation accuracy for epoch 2: 0.585 |
| Third run | Training accuracy for epoch 1: 0.537 | Validation accuracy for epoch 1: 0.5675 |
| Overfit! | Training accuracy for epoch 2: 0.673875 | Validation accuracy for epoch 2: 0.56375 |
| Fourth run | Training accuracy for epoch 1: 0.521875 | Validation accuracy for epoch 1: 0.535 |
|  | Training accuracy for epoch 2: 0.670125 | Validation accuracy for epoch 2: 0.5775 |
|  | Training accuracy for epoch 3: 0.7645 | Validation accuracy for epoch 3: 0.615 |
| Overfit! | Training accuracy for epoch 4: 0.812375 | Validation accuracy for epoch 4: 0.5725 |
| Fifth run | Training accuracy for epoch 1: 0.53175 | Validation accuracy for epoch 1: 0.58375 |
|  | Training accuracy for epoch 2: 0.653625 | Validation accuracy for epoch 2: 0.60875 |
| Overfit! | Training accuracy for epoch 3: 0.767625 | Validation accuracy for epoch 3: 0.56875 |
| Sixth run | Training accuracy for epoch 1: 0.529625 | Validation accuracy for epoch 1: 0.5675 |
|  | Training accuracy for epoch 2: 0.65425 | Validation accuracy for epoch 2: 0.5725 |
|  | Training accuracy for epoch 3: 0.731375 | Validation accuracy for epoch 3: 0.585 |
| Overfit! | Training accuracy for epoch 4: 0.801375 | Validation accuracy for epoch 4: 0.56875 |
|  | Training accuracy for epoch 1: 0.51725 | Validation accuracy for epoch 1: 0.58875 |
|  | Training accuracy for epoch 2: 0.669625 | Validation accuracy for epoch 2: 0.60625 |
|  | Training accuracy for epoch 3: 0.768375 | Validation accuracy for epoch 3: 0.60625 |
|  | Training accuracy for epoch 4: 0.847125 | Validation accuracy for epoch 4: 0.61125 |
| Overfit! | Training accuracy for epoch 5: 0.8805 | Validation accuracy for epoch 5: 0.58625 |

**4 Analysis**

If we were to gather large amounts of training data, I am going to assume with this slowly downward spiking trend that training loss would begin to smooth into a downward curve.  
  
 There are many reasons for error within this model, it could be within the reviews themselves, our training data leaving our program with many unknown words, or an imbalance of highly negative, highly positive or highly neutral reviews. Among these possible issues, things like ambiguous language, humorous reviews, sarcasm, typos and vagueness may also lead to errors. Using better models, better data, better text processing and using pre-trained word embeddings instead of my own would lead to much better results. Using a word prediction model to handle typos would be extremely beneficial as I noticed a lot of words ended up being misspelled in the review which wasted computational space and time on things like “atlathough” instead of “although”.

**5 Conclusion and Others**

Individual member contribution: Zane Taylor 100%

I spent an extremely long time on this, mainly due to the pains of running varying epochs with varying test data. I felt like it was a very understandable homework. The only thing I would have liked to have is a sample metric to shoot for, I.E. “With FFNN a training accuracy of 70-80% at 30 epochs is normal” or “When using RNN, it should be possible to train a file so that your accuracy is around 60%”, or “If you are able to get an validation accuracy of 55% with RNN your solution is acceptable” something to know that I am on the right track.