ML Recipe Classification Deep Learning Report

1. Introduction

Text classification is a popular problem in the deep learning field with various possible applications and abundant implementations already in place. Classification in text is a common task of classifying a sequence into a single category. This could be sentiment of a sentence, marking an email as spam or inappropriate comments getting flagged among other examples. Classification problems could be binary or multiclass. Deep neural networks, particularly recurrent networks, are being used for the purpose of text classification also known as sequence learning.

In this project, the main objective was to learn to train a recurrent neural network from scratch for the purpose of classifying recipes into one of 12 categories. The input to the model would be text sequences and the output would be the class of the given recipe between 1 and 12. The rest of the report is organized as follows: Section 1 introduces the project, section 2 explains the architecture used. The details of the dataset and training as well as graphical results and analysis are showcased in section 3 and section 4 concludes the paper.

2. Architecture

I conducted a total of 30 experiments with various learning rates, batch sizes, model structures, layers, types of layers etc. Figure 1 showcases the progressive improvement as I conducted more experiments and tuned the parameters and altered the structure based on the performance of the previous experiments.

Figure 1: F1 Score Improvement throughout the Project

I started the experiments with a simple model with three LSTM layers. The initial model was overfitting which could be seen through the validation and training loss. I added a dropout layer to reduce overfitting and improve the performance. To further improve the performance, I added a bidirectional layer to the LSTM layers which increased the final F1 score on 50% of the test set. The forward and backward outputs are concatenated together. Since our dataset wasn't that big, I decided to try GRU layers instead of LSTM since they're known to work better and faster for

smaller datasets. Replacing the LSTM layers with GRU layer worked well as the final F1 score increased further. I then added a second bidirectional GRU layer which still further improved performance; however, adding anymore layers did not help. Finally, I tried using branch method wherein I had two identical branches: a left and right branch. Each branch consists of an embedding layer followed by a dropout layer followed by two bidirectional GRU layers and finally a dense layer as a last layer to each branch. The first bidirectional GRU layer was set to return sequences. The final dense layers of both branches had ReLU activation function. The outputs of the branches are concatenated and sent to the final layer of the model which used a SoftMax activation function and outputs the final output for the model. Figure 2 showcases the final architecture while figure 3 showcases the shapes for each layer. The final model architecture seemed to work well as could be seen from both validation accuracy, loss as well as final F1 score for 50% of the test set.

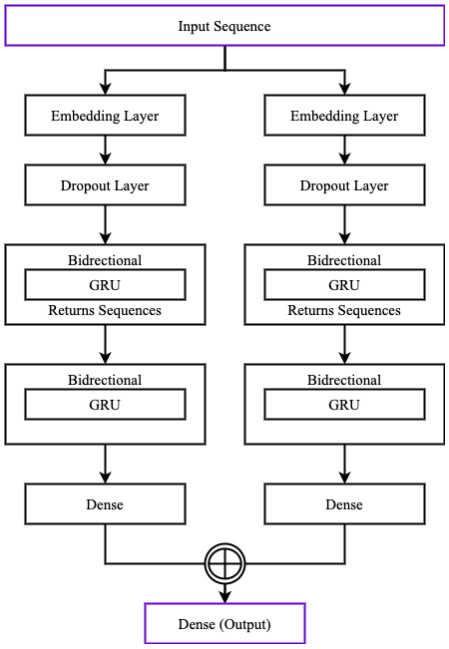

Figure 2: Architecture

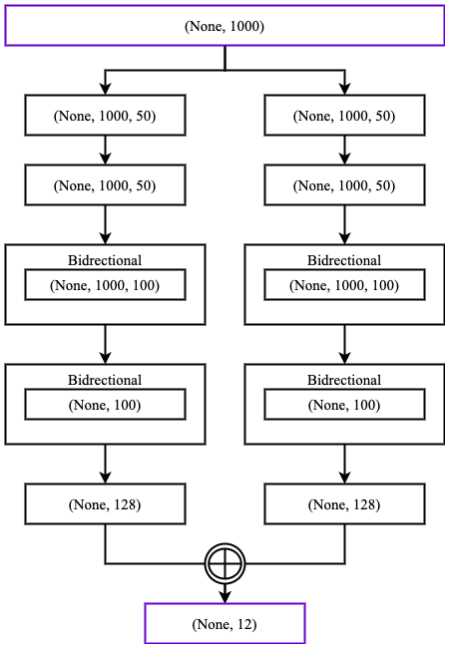

Figure 3: Layer Shapes

3. Experimental Results & Analysis

In this section, I explain the training and implementation details, dataset used, explain the hyperparameter tuning done before and during training and showcase the results of the experiments.

3.1 Dataset and Implementation Details

The dataset consisted of training, validation and test text files of recipes which could be belonging to one of 12 categories which was specified in their respective label files. Training was done as showcased in the notebook using a GPU which took about an hour on average for one experiment where the model trained for 15 to 20 epochs and stopped based on early stopping which studied the validation loss with a delta of 0.001 and patience of 7.

3.2 Hyperparameter Tuning

For the hyperparameters, I firstly tried various learning rates and finally settled on 0.001 as a starting learning rate and included the function to reduce the learning rate in case of a plateau. This seemed to work well. I added dropout layer to reduce overfitting as showcased in figures 4 and 5. I also tried different batch sizes starting from 128. The performance improved until batch size 50 as can be seen in figure 6, after which the final F1 score (on 50% of the test set) decreased. Therefore, I chose batch size 50 for the rest of the experiments.

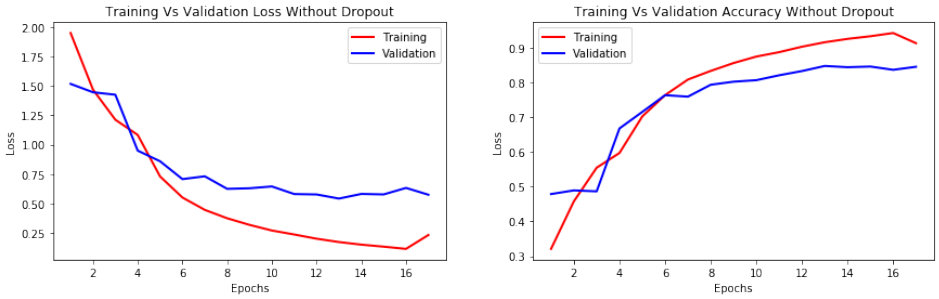

Figure 4: Training and Validation Loss and Accuracy Before Dropout

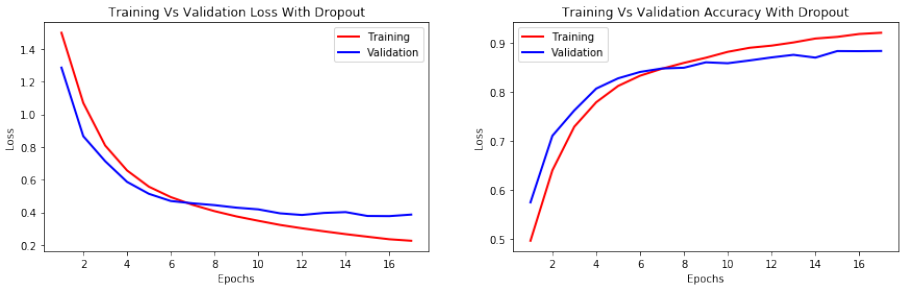

Figure 5: Training and Validation Loss and Accuracy After Dropout

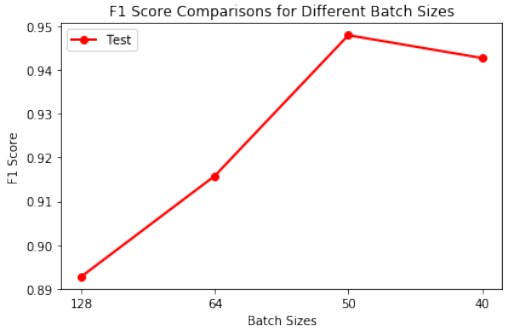

Figure 6: F1 Score with Different Batch Sizes

3.3 Results

The final F1 score I got was 0.9479. The training and validation loss and accuracy plots can be seen in figures 7 and 8. The plots showcase sufficient training and it can be seen where the training would've stopped due to early stopping when considering validation loss. Figure 8 showcases a

sample prediction for the first recipe in the test set. The predicted category is printed below the recipe.

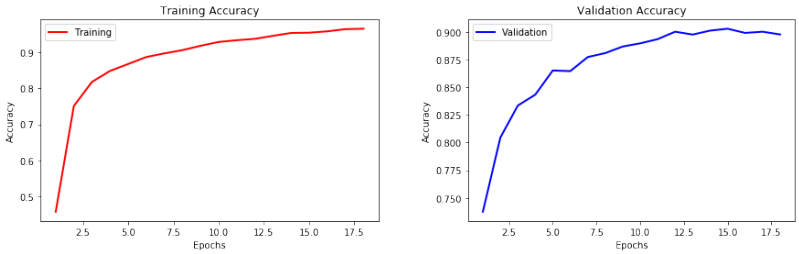

Figure 7: Training and Validation Accuracy for Final Submitted Experiment

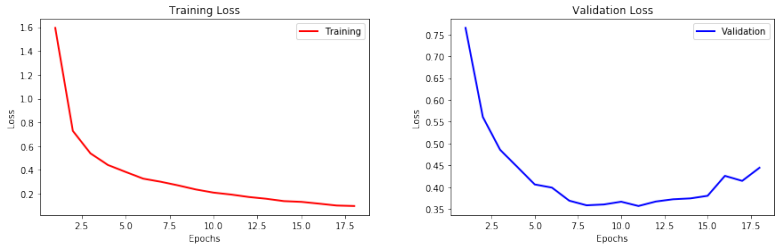

Figure 8: Training and Validation Loss for Final Submitted Experiment

```
Recipe 1
Bring a large pot of water to a boil. Add potatoes, and cook until tender but still firm, 12 to 15 minutes; drain. Pl
ace bacon in a large, deep skillet. Cook over medium high heat until evenly brown. Cut into small chunks; set aside.
Place potatoes into skillet, and cook on medium heat until browned. Flip potatoes occasionally to prevent sticking. S
tir in green pepper, red pepper, onion, and mushrooms. Cook until vegetables are tender. Stir in cooked bacon, and se
ason with salt and pepper. Cover with shredded cheese, and turn mixture until cheese is melted. Keep on low heat whil
e cooking eggs. Cook eggs to your preferred style. Place potatoes in a large serving dish, and top with eggs (2 per s
erving). Hearty breakfast skillets. Serve with toast or muffins.

Predicted category: 3
```

Figure 8: Sample Prediction

3.4 Running the Code

Training and testing both can be done through the notebook. I've included all code I used for loading the data, training the model, evaluating the model as well as importing GLOVE word embeddings to be used for the embedding layer. To run the code, it would be beneficial to set the root path to wherever you would like to save the model and results or load the model from.

4. Conclusion

There are many possible networks that can be used for the purpose of text classification including both convolutional and recurrent neural networks. For this project, I chose a two-branch recurrent neural network structure which seemed to achieve a competitive F1 score of 0.9479 on 50% of the test set. However, I hope to use what I've learned to train a better model with higher accuracy in the near future. Through this project, I learned about text classification, sequence learning, tokenizing and structuring text, and training a recurrent neural network from scratch. I learned how to improve performance and reduce overfitting when required.

References

[1] Anastasiu, David. COEN 342 Class Slides. Santa Clara University, 2021
[2] Matplotlib tutorial: https://matplotlib.org/
[3] Tensorflow Bidirectional Layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional
[4] Tips for training Recurrent Neural Networks https://danijar.com/tips-for-training-recurrent-neural-networks/