Fake News Detection
CSE 472 - Project II Report

Kevin Strouzas
ASUID 1212913869

Table of Contents

1 Introduction

1.1 Project Overview

For the project, data was provided including Chinese and English-translated titles of articles that had to be classified among three labels: unrelated, agreed, and disagreed for the purpose of fake news detection. The data was preprocessed and a Bidirectional LSTM to LSTM-based neural network was trained on it producing a 86.46% validation accuracy and a 0.8618 weighted average f1-score. Jupyter Notebooks and alternative .py files have been provided for the purpose of running the code along with usage requirements and instructions.

2 Preprocessing and Feature Engineering

2.1 Process Followed

The preprocessing methods followed for the datasets is as follows:

- Data is loaded into a DataFrame with pandas and the **filter_dataset()** function strips all but relevant English/Chinese characters and removes bad lines from the dataset.
- Chinese word segmentation is done via the **jieba** Python package. See the **segment_zh_data()** function.
- EN and ZH tokenizers (using the keras **tokenizer** module) are created with the clean training set.
- Labels are encoded to a range [0,2] of values indicating a label. See the **encode_labels()** function.
- Titles are tokenized into word vectors using their respective language's tokenizer and padded/truncated to their language's mean title size plus one standard deviation. See the **tokenize()** function.
- The data is then split and stored via **save_data()** into feature/label sets with the pickle module.

As mentioned, the preprocessing throws out bad lines and titles with null values, so the output csv may be slightly smaller than the input provided due to bad lines. But the amount of lines removed is pretty insignificant.

3 Model Description

3.1 Model Parameters

After going through many structures, a bidirectional LSTM model was decidedly the best approach, with a bidirectional GRU model of similar architecture following close behind.

To the right can be seen some of the primary parameters used and tuned for the bidirectional LSTM model which produced the best results from testing. Further tuning would improve these results.

Originally the models used only the EN set, but adding in the ZH set improved results drastically (>=2% in validation accuracy improvement).

Parameter	Value
EN SENTENCE SIZE	44
ZH SENTENCE SIZE	33
EN VOCAB SIZE	35000
ZH VOCAB SIZE	70000
EMBEDDING LAYERS	1x2
EMBEDDING SIZE	50
LSTM/GRU LAYERS	1x3
LSTM/GRU SIZE	100x2 (Bidirectional)
SPATIAL DROPOUT	0.75
RECURRENT DROPOUT	0.25
OPTIMIZER	Adam
BATCH SIZE	1024
DENSE ACTIVATION	Softmax

3.2 Model Architecture

The architecture consists of two inputs: the Chinese titles fed together as a 2xZH_SENTENCE_SIZE ndarray and the English titles fed together into a separate model as a 2xEN_SENTENCE_SIZE ndarray. They are both ran through an embedding layer with trainable weights, then a Bidirectional LSTM or GRU layer, and then the two outputs are concatenated and fed through a final LSTM or GRU layer before producing an output with a Dense layer with a softmax activation.

A graphical visualization of the model architecture can be viewed in the **Appendix**.

3.3 Training the Models

Training was done via the keras library. Many different model architectures were attempted using the **Functional API** primarily including LSTM structures with embedding layers.

The typical process for training each of the architectures went as such:

- The features/labels stored during preprocessing are loaded using pickle along with any tokenizers.
- Class weights are calculated using the **sklearn class_weight utility** so that the model isn't too heavily skewed by the abundance of unrelated labels (otherwise the model *may* just have a bias towards picking unrelated).
- Hyperparameter values are set. In some cases, lists of hyperparameters are used and models are created using loops to automate the process of testing for ideal values.
- The model is fit on the training data and validated against validation data.
- The adam optimizer was used to reduce training time, other optimizers such as SGD may be able to produce better results.

There are three callbacks used during the process to improve the training/validating process.

1. EarlyStopping for halting the training when validation loss stops doesn't reduce after several epochs.
2. Tensorboard to visually view historical training data and model architectures.
3. Checkpointer to save after each epoch if validation accuracy improved. Models are saved with a name indicating basic architecture along with a timestamp to prevent overwriting.

3.4 Validating the Models

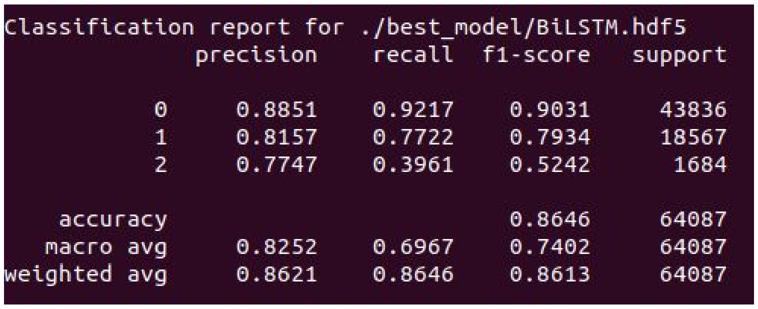
As mentioned, tensorboard was used to validate models during the tuning stages. The visualized results of these results are available in the **Appendix** section.

For validating the best model the sklearn package was used to quickly obtain various metrics including precision, recall, accuracy, and f1-score. The results can be seen in the **Results** section.

4 Results

4.1 Validation Results

The model has an observed 86.46% accuracy with a weighted average f1-score of 0.8613 on validation data. Graphs of historical training data for 32 models are available in the Appendix section. Additionally, sklearn was used to generate various metrics including f1-scores. Metrics can be gathered by calling **python get_metrics.py** where it gets metrics from running on the validation set.



```
Classification report for ./best_model/BiLSTM.hdf5
              precision    recall  f1-score   support

           0     0.8851    0.9217    0.9031     43836
           1     0.8157    0.7722    0.7934     18567
           2     0.7747    0.3961    0.5242      1684

    accuracy                         0.8646     64087
   macro avg     0.8252    0.6967    0.7402     64087
weighted avg     0.8621    0.8646    0.8613     64087
```

5 Use Instructions

5.1 Software Requirements and Installation

To use and interact with the .ipynb files, Jupyter Notebook is required. However, .py files have been included as an alternative and confirmed to work with Ubuntu.

All code is written in Python 3, so additionally an up-to-date installation of Python is required. In addition, all packages used are required to run the code. The packages used are listed both in the Packages and Libraries section of this report, and the requirements.txt file included.

For installing all of the necessary packages, the following should be satisfactory:

1. Ensure Python 3 and pip are working: **python --version** and **pip** in the terminal.
2. Install your choice of tensorflow (recommended) via **pip install tensorflow** or tensorflow-gpu. If installing tensorflow-gpu, you may have to follow certain procedures for your OS. For Windows 10 and Linux, this guide may help. Ensure only one of the two is installed.
3. Install the rest of the packages with **pip install -r requirements.txt**.
    a. **NOTE:** Tensorflow has been omitted from the requirements.txt file since the two (gpu and non-gpu) versions can cause conflicts with one another if both are installed.

5.2 Running the Code

Assuming all packages and software are installed, the code can be run through either the ipynb notebooks or the python files. The .ipynb notebooks are numbered in the order they should be run.

For all steps, ensure the train, test, and validation data is stored in /src/data/train.csv, /src/data/test.csv and /src/data/validataion.csv respectively.

If you want to skip the modelling process and just recreate the **submission.csv** or produce the same metric results, skip to '3. Predicting' or run the preprocessing step and skip to '4. Getting Metrics' respectively.

1. Preprocessing Step:
   a. In your terminal, call **python preprocess.py** to preprocess the data. Wait for it to finish. This may take several minutes as the Chinese word segmentation and tokenization steps can be somewhat intensive. This will save a feature/label split in the split_data directory by default.
2. Model Creation
   a. Call **python create_model.py** to generate a model using the default parameters used to recreate the model used in this project. The results may be slightly different due to random initialization of weights.
3. Predicting
   a. Call **python predict.py -m "your_model_dir"** where **your_model_dir** is the directory of your model. By default, this file runs with the provided BiLSTM in the /src/best_models/ directory.
   b. After a few minutes, a submission.csv file should be generated in the /src/data/ directory.
4. Getting Metrics
   a. (Optional) Ensure BiLSTM.hdf5 is stored in the best_models directory. Calling the metrics file via **python get_metrics.py** will return metrics including f1-scores. This file, like create_model.py, takes a parameter for calling a specific model as well.

6 References

6.1 Software Used

Jupyter Notebook was used throughout all of the process to create Python 3 notebooks for prototyping code and coming up with preprocessing methods as well as model structures.

Tensorboard was also utilized to visualize validation / train results and model structure.

6.2 Code Environment(s)

All of the code was written and operated on a Windows 10 OS in an Anaconda distribution of Python 3.7.3. Training the model was done on a GTX940m GPU, and better systems may be able to train better models.

The code has been tested in a Ubuntu environment (18.04.1) in the latest version of Python 3: 3.7.4, and all of the .py files needed to recreate the conditions in this project successfully ran.

6.3 Packages and Libraries

All modules used are listed in both the imports and the requirements.txt file (aside from tensorflow, see the note on installation), but for reference here are the contents of that file with information on what they were used for.

Modelling Packages

- tensorflow OR tensorflow-gpu. Tensorflow-gpu requires some extra setup and the steps can be different for each system.
- tensorboard==2.0.1: Used primarily for visualization of training history. Can be removed from the code if this isn't desired or tensorboard for some reason doesn't function to little effect.
- keras==2.3.1: Used for the bulk of the work, including creating the word tokenizer and tokenization as well as creating and training models.

General Purpose Packages

- pandas==0.24.2: primarily used for manipulating the data using DataFrames and loading/saving to CSVs.
- numpy==1.17.3; Primarily used for reshaping data, should come bundled from installing pandas.
- scikit-learn==0.21.2: Used for balancing class-weights to reduce the chances of model overfitting on imbalanced training data as well as for obtaining model metrics.
- argparse==1.4.0: Used for command-line argument parsing.

Default Packages (Come installed by default in current iterations of Python 3)

- pickleshare==1.4.0
- os: Along with pickle, used for loading/saving files.
- time: For time-stamping models.

Chinese Word Segmentation

- jieba==0.39: For segmenting Chinese text into words.

6.4 Tutorials Followed

[1] sentdex. 2018. Deep Learning with Python, TensorFlow, and Keras tutorial. (August 2018). Retrieved November 10, 2019 from youtu.be/wQ8BIBpya2k

[2] Ceshine Lee. 2018. [NLP] Four Ways to Tokenize Chinese Documents. (November 2018). Retrieved November 15, 2019 from medium.com/the-artificial-impostor/nlp-four-ways-to-tokenize-chinese-documents-f349eb6ba3c3

[3] Jason Brownlee. 2019. How to Prepare Text Data for Deep Learning with Keras. (August 2019). Retrieved November 15, 2019 from machinelearningmastery.com/prepare-text-data-deep-learning-keras/

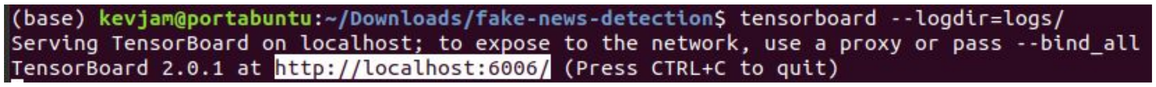
[4] Jason Brownlee. 2017. How to Use Word Embedding Layers for Deep Learning with Keras (October 2017). Retrieved November 30, 2019 from https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

7 Appendix

7.1 Loss/Accuracy Graphs For All Training Data

This section's graphs contain plots for 32 trained models of differing dropout sizes. Unfortunately, Tensorboard does not provide an intuitive way to generate a key for which model is which.

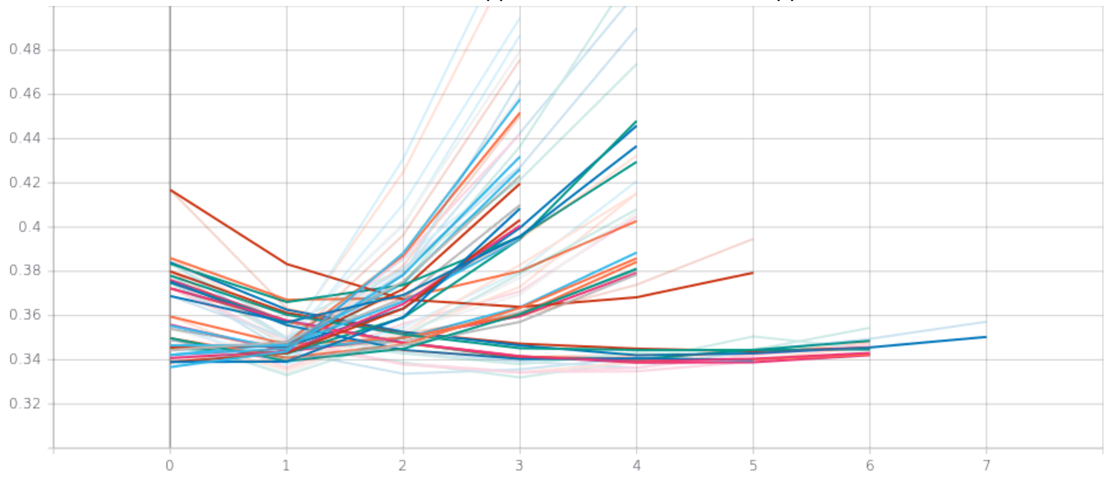
However, graphs can be accessed by navigating to the directory where the logs folder is stored and typing in the terminal: **tensorboard --logdir=logs/** and navigating to the link shown:



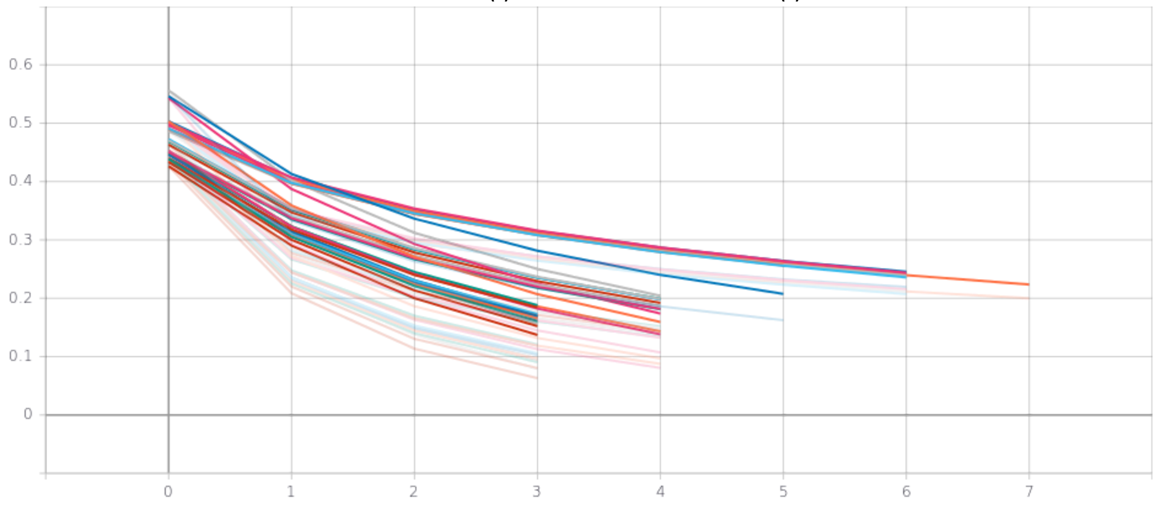
```
(base) kevjam@portabuntu:~/Downloads/fake-news-detection$ tensorboard --logdir=logs/
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.0.1 at http://localhost:6006/ (Press CTRL+C to quit)
```

Historical training logs for these models haven't been included, but will be generated upon training new models. They were removed due to the size of all of them together taking up more space than the rest of the project.

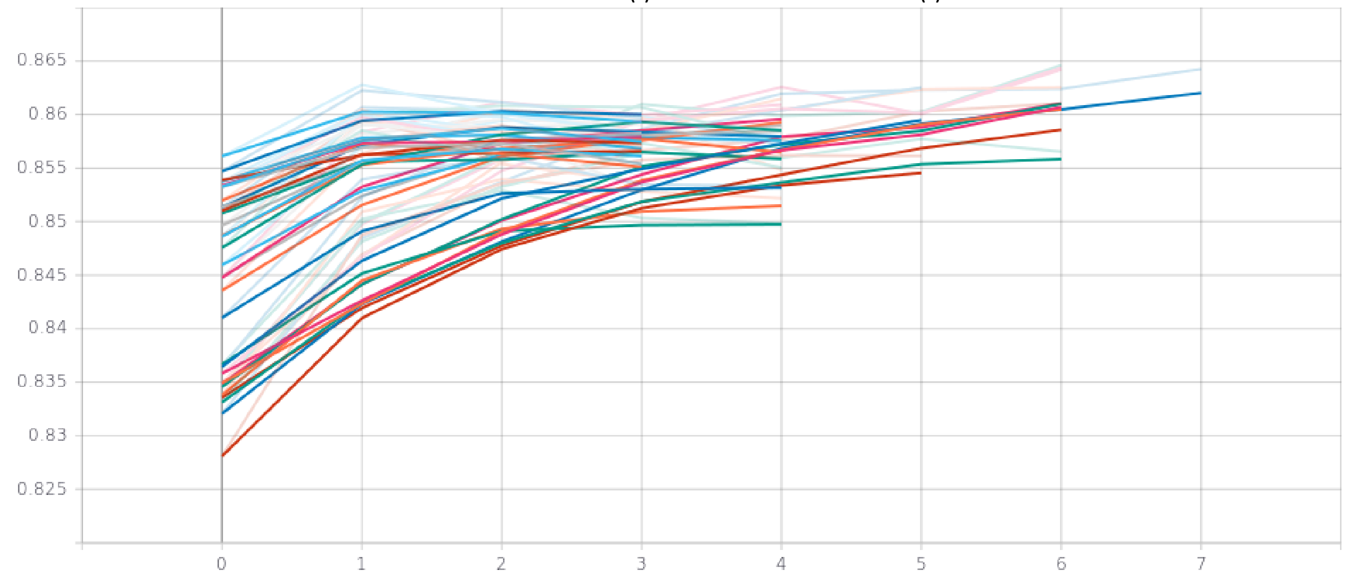
Validation Loss$_{(Y)}$ vs. Number of Epochs$_{(X)}$



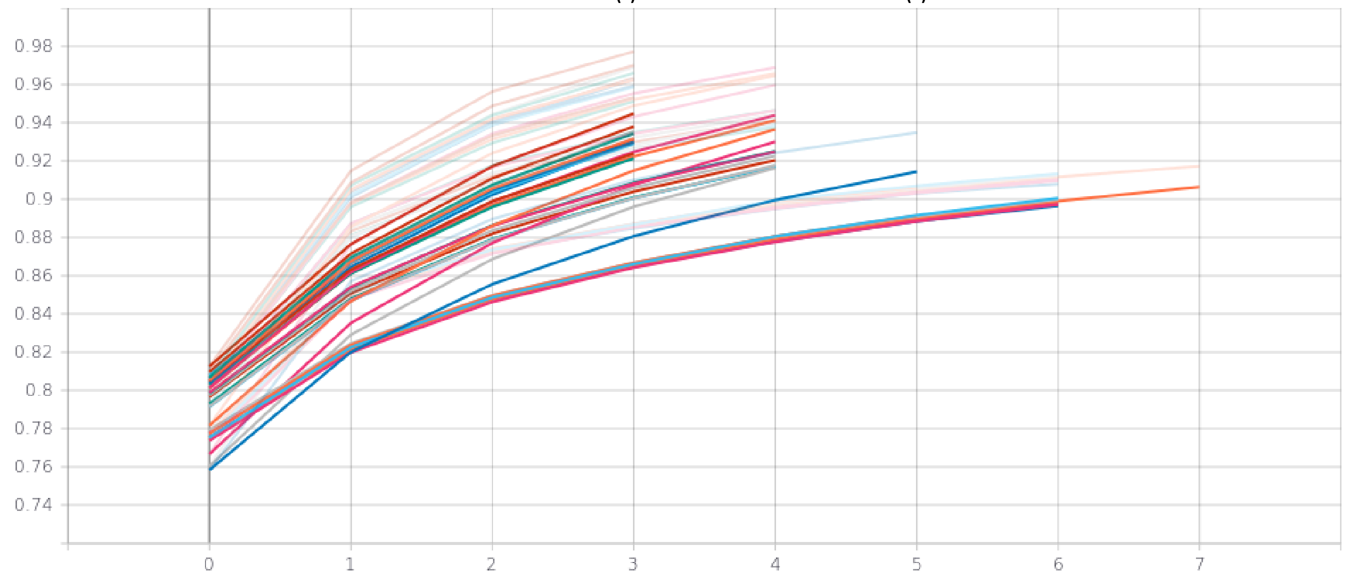
Training Loss$_{(Y)}$ vs. Number of Epochs$_{(X)}$



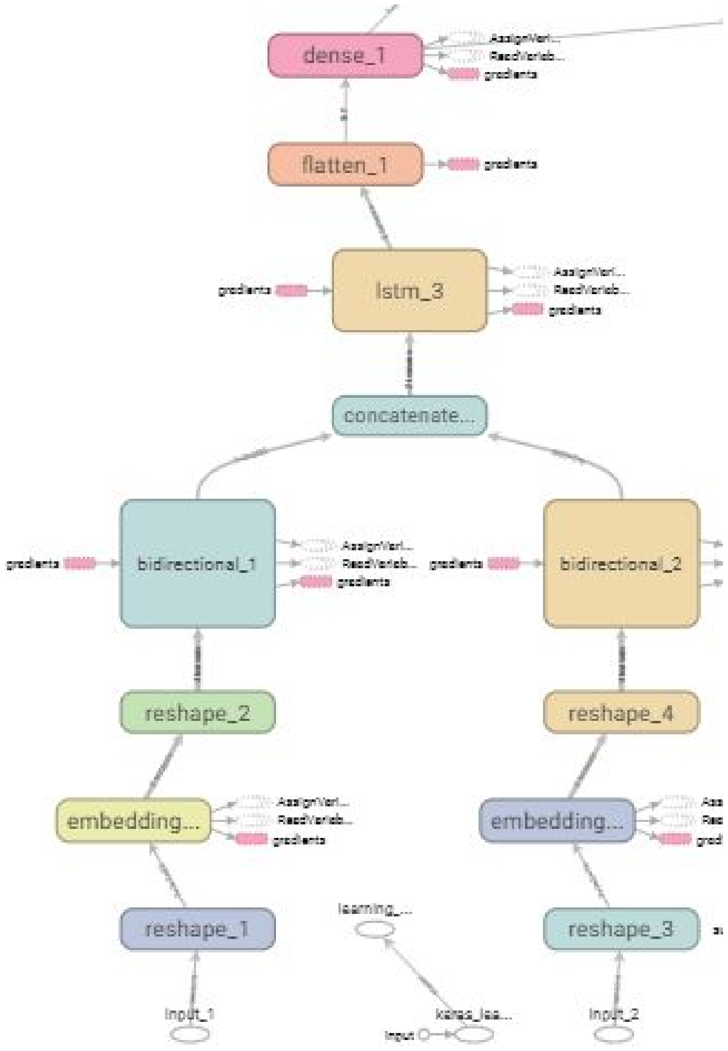
Validation Accuracy$_{(Y)}$ vs. Number of Epochs$_{(X)}$



Training Accuracy$_{(Y)}$ vs. Number of Epochs$_{(X)}$



7.2 Main Model Architecture - Bidirectional LSTM


dense_1
AssignVari...
ReadVariab...
gradients
flatten_1
gradients
lstm_3
gradients
AssignVari...
ReadVariab...
gradients
concatenate...
gradients
bidirectional_1
AssignVari...
ReadVariab...
gradients
gradients
bidirectional_2
reshape_2
reshape_4
embedding...
AssignVari...
ReadVariab...
gradients
embedding...
reshape_1
learning_...
reshape_3
input_1
keras_lea...
input
input_2