CS291A Deep Learning NLP Assignment 2: Fake News Detection

Report by Vivek Pradhan

The major components of my solution architecture are outlined below:

Feature identification – There were two broad categories of features – the word statement and the meta data. A task was to have good representations for both of these features. Two options for word representations were considered – static glove embeddings taken from the Wikipedia (6 Billion token dataset) and non static embeddings initialized by the model and learned during the training time. Word embeddings (both static and non static) were 100 dimensional.

Meta data was difficult to categorize because of the diverse information present. I tried to quantify different meta tags into discrete classes and a symbolic 'rest' class that allowed me to condense the values into a fixed (limited) number.

Details of meta data classes:

- Speakers – Top 20 speakers based on value counts
- Jobs – Top 10 based on value counts
- Party – Top 5 based on value counts
- States – 50 states
- Subject list – Top 12 subjects based on value counts
- Venues – Top 10 venues based on value counts

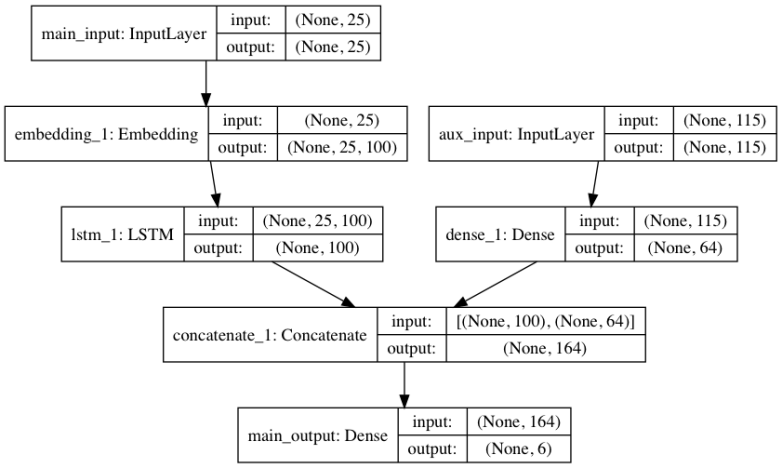
The mapping of values to these classes was not always an exact match based approach. To bucket similar identifiers (for instance to bucket 'TV interviews' and 'Television interviews' in same broad category), I used substring matching and therefore the classes were not exact values picked from the data-set. For a lot of cases the classes were hand-picked so that it represented a majority of the data very well.

Models

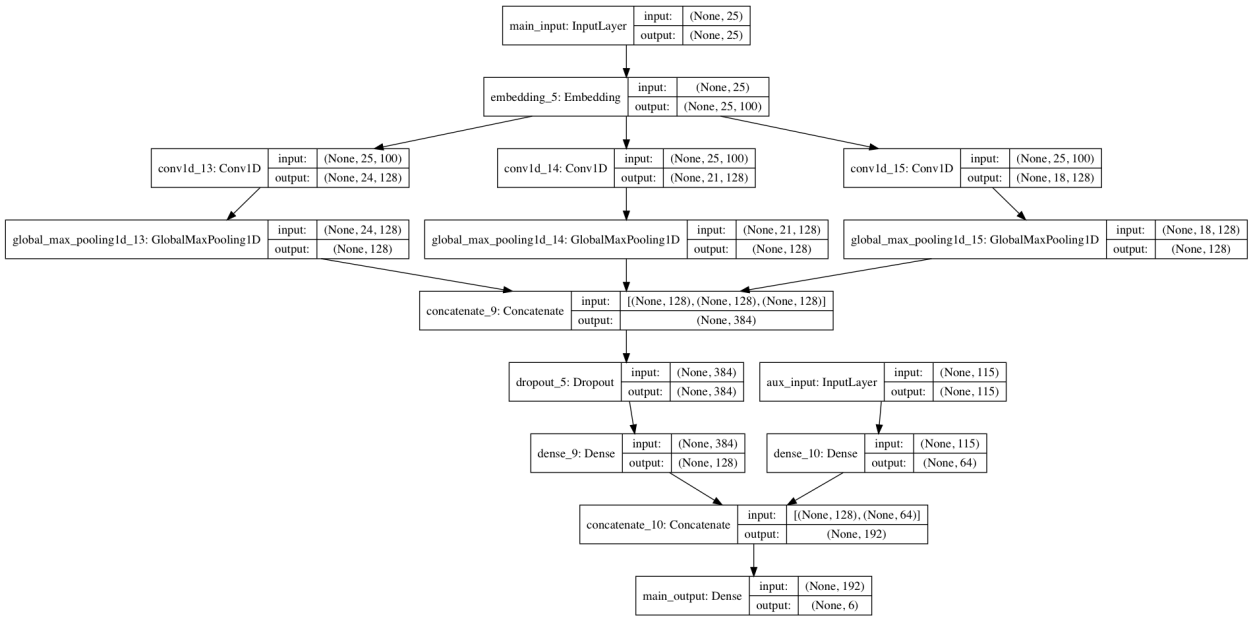
I implemented two models to test the efficacy on the Fake news task.
Inputs –
main_input – Word embeddings input from statement
aux_input – Input vector formed from Meta data.

Concatenating the 6 meta data classes as one hot vectors and then getting dense representations out of it and adding that to LSTM/CNN output worked best for me. It also seemed to follow the intuition that meta data was considered to be discrete classes by me. So, learning representations for them in continuous space and applying a CNN or LSTM to that representation did not seem to be the best way forward because I wanted to capture the correlation of meta data to the final classification of the statement rather than understand the relation between different meta data classes. The CNN or LSTM approach would be more suitable for meta data classes where we are trying to understand the co-dependence or similarity between them. I assumed they are independent.

LSTM based model:
Hyper params for best result (28% test accuracy)
Word embeddings – Static glove embeddings , Batch size – 40, Starting learning rate for SGD – 0.025 with Nesterov momentum, Dropout in LSTM layer – 0.25, epochs – 30



CNN based Model:
Hyper params for best result (30% test accuracy)
Word embeddings – Non static 100d, Batch size – 48, Starting learning rate for SGD – 0.025 with Nesterov momentum, Dropout in CNN layer – 0.7, epochs – 30, Filters – 2,5,8 (128 each)



Interesting finds

- The LSTM model overfit very easily with standard optimizers like Adam or RMSProp. Tuning them is quite difficult too so SGD worked best for me.
- Simple things like batch size and starting learning rate made a huge difference in the test performance of model and it's convergence properties
- LSTM tended to do extremely well with static glove embeddings whereas CNN did very well with Non static word embeddings
- Keras has two dropout settings for LSTM's. One is the simple dropout and the other one is recurrent dropout. Adding these dropouts led to some unexpected results and training performance flattened quite quickly and the model's improvement slowed down drastically after 20 epochs. This adversely affected validation and therefore test accuracy too for LSTMs. (shown in the figure below)