Noun Compound Classification

Name: Ansh Khurana, Drumil Trivedi, Riya K Baviskar
Roll Number 170050035, 170020016, 170050011

Assumptions

1. train-val split on train.csv
2. The code was written on a Google Colab notebook, hence to run it, please run all the cells one by one.

Implementation Details

For the word embeddings, we are using GloVe embeddings downloaded from https://nlp.stanford.edu/projects/glove/ (Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip)

The model to be used, embedding dimensions, K (no. of context sentences), model choice ('simple_ff', 'lstm', 'bidir_lstm', 'self_attn')

```
embed_size = 50
SEED = 42
k=15 # no. of context sentences to be picked.
input_data_context = True
model_name = 'lstm'
lstm_rep_size = 32
```

This implementation can be found in the submitted noun_compound_classification.ipynb notebook. Along with this main notebook, task1 training has been done in a separate copy called task1.ipynb.

All our models are implemented using the Keras API in TensorFlow 2.

Google Colab link:

https://colab.research.google.com/drive/18orHfmw-rTa-LInlz8PDjX9sEwKibrHw?usp=sharing

Approach

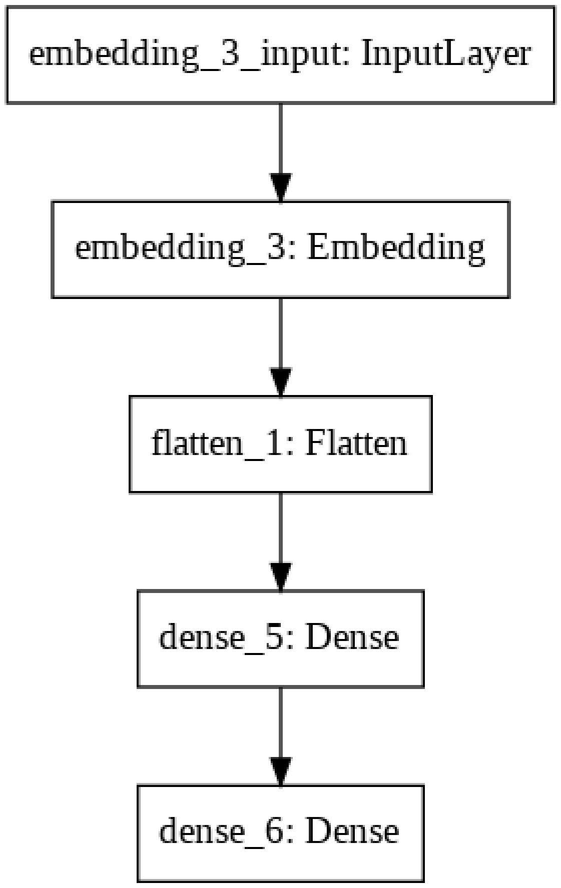
We are using the Glove Word Embeddings for converting the words into their dense vectorial representations. We have set the GLoVe embedding dimension to 50.

Simple Feed Forward Network for NNC

The words forming the noun compounds are used as input. In the test JSON file, we found a noun compound that was four words long. Thus the max length of the input is set to 4, and appropriate padding is done. Therefore input size is fixed to batch_size x 4 x 50.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 4, 50)             2500000   
_________________________________________________________________
flatten_1 (Flatten)          (None, 200)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 32)                6432      
_________________________________________________________________
dense_6 (Dense)              (None, 37)                1221      
=================================================================
Total params: 2,507,653
Trainable params: 7,653
Non-trainable params: 2,500,000
_________________________________________________________________
```

The architecture of the Feed Forward Network used for this task is shown below:

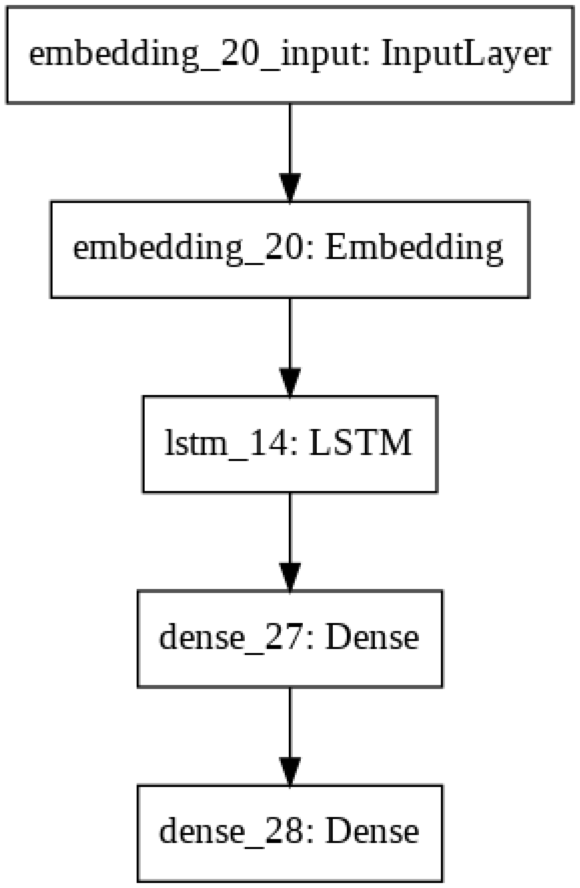


The output layer (dense_6) corresponds to the softmax activation and probability scores onto the 37 Noun Compound Classes.

LSTM Model

For task 2, we are required to take in variable-length inputs, which are the context sentences for a given noun compound that'll help us make better predictions. We have picked the hyperparameter 'k' to be 15, denoting the number of context sentences to be used in the input. Our LSTM has 32 units. We also experimented with a bidirectional LSTM, but the training for that network is quite slow.

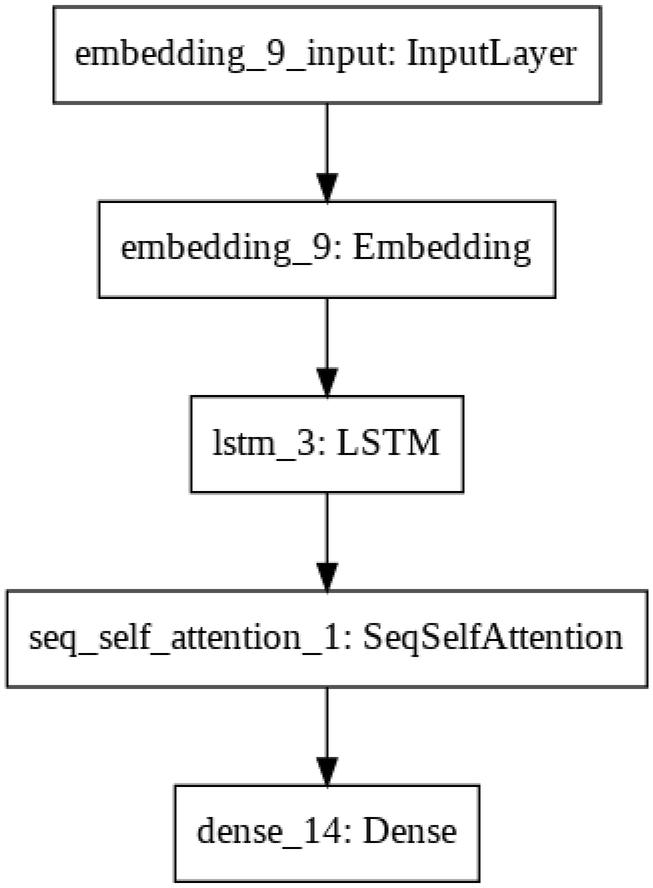
The architecture of the LSTM Network used for this task is shown below:



LSTM w/ Self Attn

For our LSTM with Self Attention implementation, we replace the ‘dense_27’ layer in the previous LSTM architecture and add a self-attention layer. For the Self Attention layer, we have imported SeqSelfAttention from keras_self_attention. Thus, the hidden layer outputs are fed into the self-attention mechanism to obtain the final weighted representation. This representation is the input to our Dense softmax layer which makes the final predictions.

The architecture for this approach is shown below:


embedding_9_input: InputLayer
embedding_9: Embedding
lstm_3: LSTM
seq_self_attention_1: SeqSelfAttention
dense_14: Dense

Results

The outputs for the test set are stored in the provided test.json format. For each Noun Compound, the model returns the prediction out of the 37 classes. The predicted label is added to the test json format.

We had split the training set into a 75-25 train/validation set for testing out our implementation.

From our three approaches, we were able to obtain the best validation accuracy of ~ 60%. We have attached results from this model in our final submission in **test.json**.