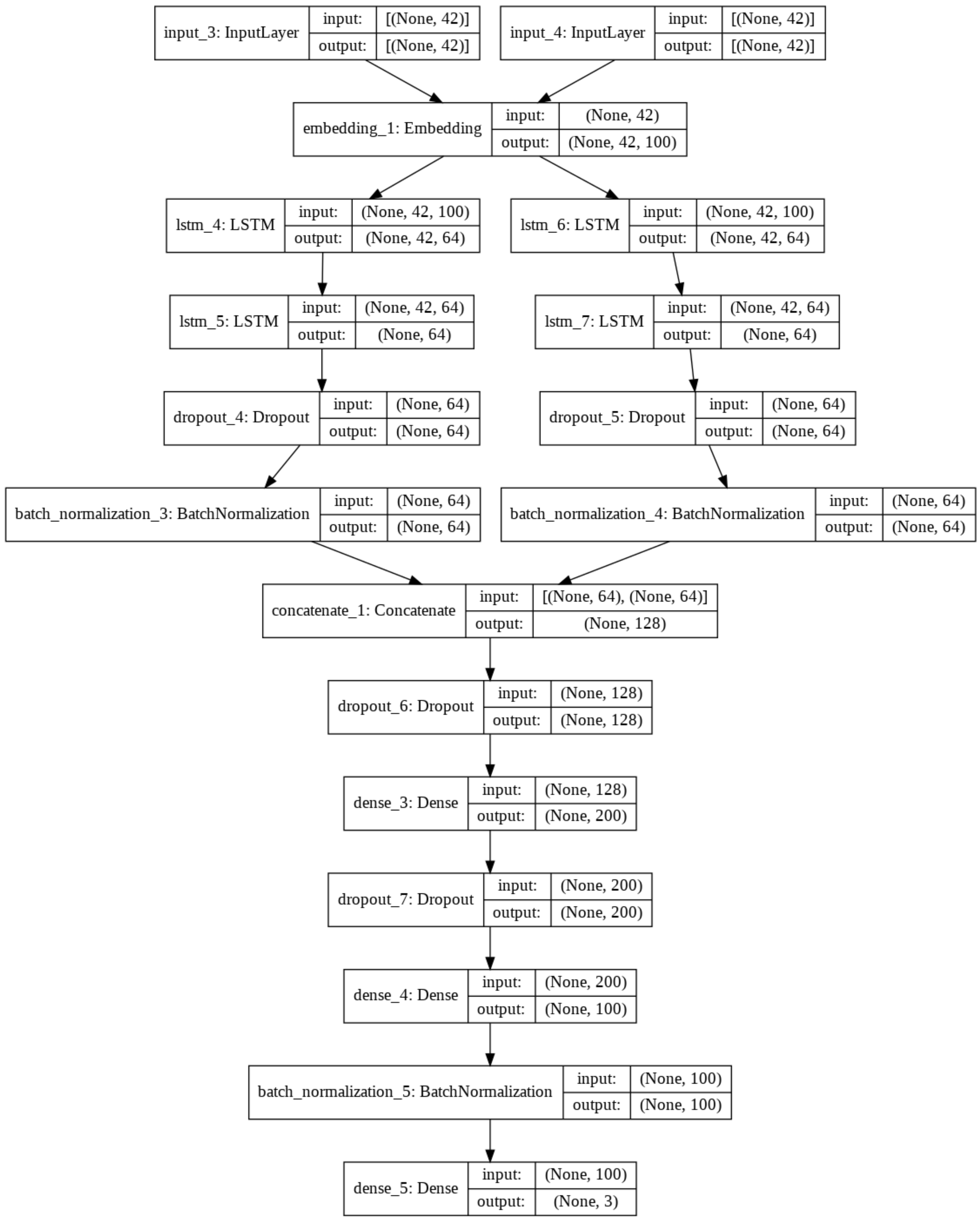
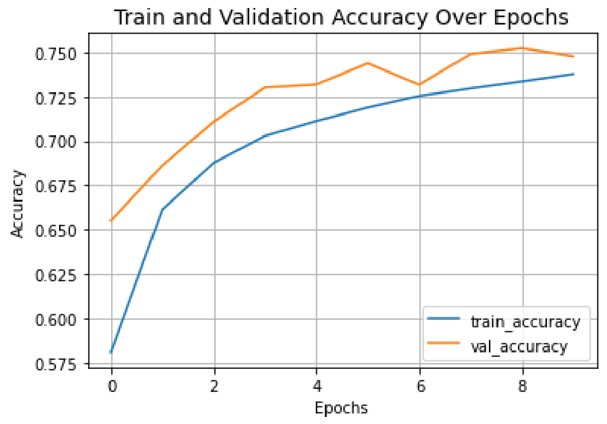
NLP TASK

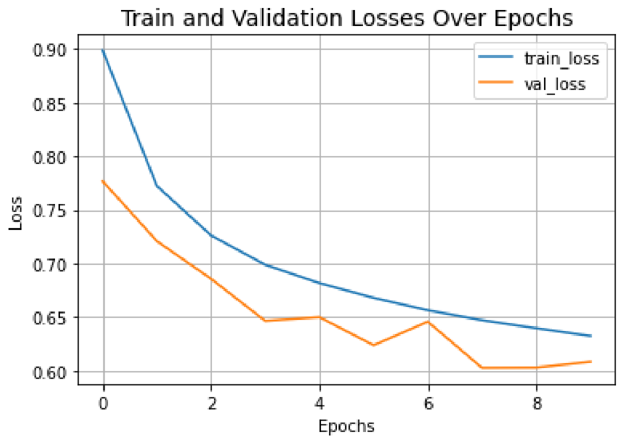
Task 1

1. Extracted training, validation, testing data from the SNLI dataset .json files by removing the brackets and special characters from sentence parse
2. Splitting the data further into texts and their corresponding labels and encoding the labels {‘contradiction’: 0, ‘neutral’: 1, ‘entailment’: 2}
3. From this point training data = training, validation data = validation, testing data = test
4. Where training[0] is the premise and training[1] is the hypothesis
5. Both are concatenated and then tokenized using the Tokenizer() method and then the text corpus is converted to sequences of integers
6. The sequence of integers is post padded with 0s post to a maximum length of 42
7. Created an embedding matrix with embedding vectors from the pre-trained GloVe embedding, this will be used to embed ‘premise’ and ‘hypothesis’ text for model training
8. I used two layers of Bidirectional LSTM as the backbone as it had better validation accuracy and embedded the premise and hypothesis using the embedding matrix created in the last step.
9. We then pass the embeddings to BiLSTM layers with 64 units for each embedding and then apply batch normalization to them
10. We concatenate the normalized premise and hypothesis and apply a dropout layer and then pass it through dense layers and dropout layers
11. We finally pass it through a dense layer with units as the number of categories(3) and a softmax activation function. We use an Adam optimizer with a learning rate of 0.001 and a categorical cross-entropy loss function.
12. I got a training accuracy of 76.3 and validation accuracy = 75.57 after using a batch size of 512 and training it for 50 epochs



input_3: InputLayer | input: [(None, 42)] | output: [(None, 42)]
input_4: InputLayer | input: [(None, 42)] | output: [(None, 42)]
embedding_1: Embedding | input: (None, 42) | output: (None, 42, 100)
lstm_4: LSTM | input: (None, 42, 100) | output: (None, 42, 64)
lstm_6: LSTM | input: (None, 42, 100) | output: (None, 42, 64)
lstm_5: LSTM | input: (None, 42, 64) | output: (None, 64)
lstm_7: LSTM | input: (None, 42, 64) | output: (None, 64)
dropout_4: Dropout | input: (None, 64) | output: (None, 64)
dropout_5: Dropout | input: (None, 64) | output: (None, 64)
batch_normalization_3: BatchNormalization | input: (None, 64) | output: (None, 64)
batch_normalization_4: BatchNormalization | input: (None, 64) | output: (None, 64)
concatenate_1: Concatenate | input: [(None, 64), (None, 64)] | output: (None, 128)
dropout_6: Dropout | input: (None, 128) | output: (None, 128)
dense_3: Dense | input: (None, 128) | output: (None, 200)
dropout_7: Dropout | input: (None, 200) | output: (None, 200)
dense_4: Dense | input: (None, 200) | output: (None, 100)
batch_normalization_5: BatchNormalization | input: (None, 100) | output: (None, 100)
dense_5: Dense | input: (None, 100) | output: (None, 3)

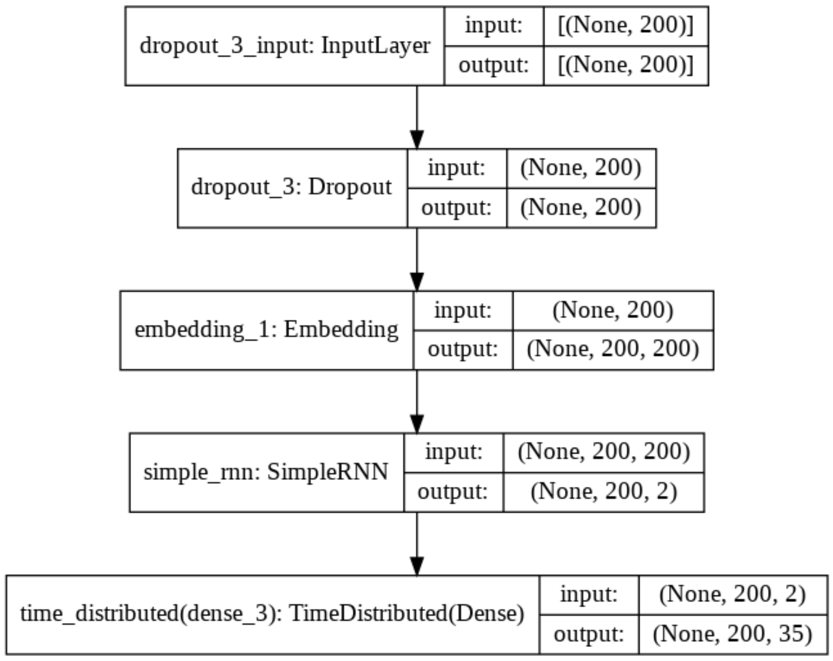




Task2

1. For this task, I will be attempting POS tag probing.
2. We find tokenize the sentences from the training dataset by using nltk method to tokenize sentences and thereafter store them in a list
3. We then iterate through this list which contains the tokenized sentences and evaluate their POS tags and then append word and its POS tag pair into a list(Y)
4. We then iterate through the list Y and take the (word, POS) pair to append word and its POS tag to two separate lists I and O respectively.
5. Now, we are at a stage where we have separated training data and their respective POS tags
6. We tokenize each word and its POS tags and encode them into a sequence of numbers. Which is then further post padded with 0 to a maximum length of 200.
7. We further one-hot encode the POS tags using keras to_categorical function.
8. Next, we construct an architecture to implement this task
   - We probe on a model representation from the previous tasks i.e a specific frozen layer of the model and we start building the model from this layer. We take the 4th layer from last and add it to a keras.Sequential function.
   - We create an embedding layer over the probed layer, which had parameters - input_dim as vocab size(20,000) which tells the number of unique words expected in data, output_dim as the embedding size(200) which is the length of the vector with which each word is represented. And finally the length of the input sentence(200)

- We then add an RNN layer with 2 cells with returns sequences to the next layer
- We then add a time distributed layer to get output from each sequence layer with units set as the number of classes and using the softmax activation function
- We use an Adam optimizer for training with a learning rate of 0.01 and a categorical cross-entropy loss function and we have 511 trainable parameters which are sufficient for the probe to perform well and not get the results by overfitting.
- We train the model for 5 epochs and get a validation accuracy of 92, which means the model has a center of gravity in the middle, i.e the POS tag features are captured in middle layers of the model
- We see a significant increase in the validation accuracy due to the fact that the Bi-LSTM model is able to interpret the POS tags. To address the probe confounder problem, it may seem that due to the complexity of RNN as compared to MLP, the model may be memorizing the outputs from the probed layer with some supervision. But, in fact, the learned parameters of the probe is low of ~800 parameters only. In literature shallow MLPs are preferred which will be a part of future work. Literature also uses 'selectivity' as a metric for the performance of the probe, selectivity = linguistic accuracy - control accuracy



*I had tried to probe on different layers of the trained model but ran into dimensionality issues. This will be taken up as a future work