CSI5386

Natural Language Processing

Assignment 2

Text Entailment and Semantic Relatedness

Submitted By:

Deepshi Mediratta 300086013

Nikhil Oswal 300074118

PROBLEM STATEMENT

To develop and train a Deep Learning classifier for Recognizing Textual Entailment & Semantic Relatedness and report the best results for all the experiments performed.

SOLUTION

Dataset Analysis

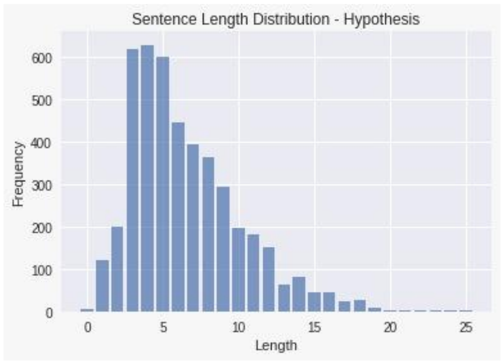
For this task, we have used the SICK (Sentences Involving Compositional Knowledge) dataset which consists of 10,000 pairs of sentences annotated for semantic relatedness and entailment. This dataset was divided with 5000 sentence pair in the training set, 500 sentence pair for the validation set and 4927 pair for testing purpose.
Sample text from the SICK dataset showing different classes of inference relationship.

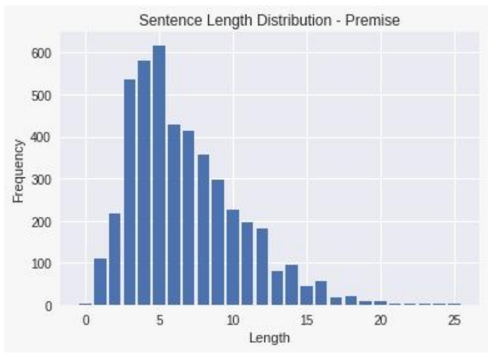
Premise	Hypothesis	Label
A dirty soccer ball is rolling into a goal net	A soccer ball is rolling into a goal net	ENTAILMENT
A person who plays soccer is kicking a ball into the goal	A soccer ball is rolling into a goal net	NEUTRAL
A soccer ball is not rolling into a goal net	A soccer ball is rolling into a goal net	CONTRADICTION

We observed that the dataset was highly imbalanced with more than 50% as Neutral.

	Train	Valid	Test
ENTAILMENT	28.86%	28.8%	28.69%
NEUTRAL	56.35%	56.4%	51.47%
CONTRADICTION	14.77%	14.8%	14.61%

We also observed that most of the sentences have number of words in the range 4-10 with only a few sentences having number of words greater than 25.





Task 1: Text Entailment

Given two sentences, one named sentence_A (Premise) and one named sentence_B (Hypothesis), recognize whether the hypothesis can be inferred from the text. Text entailment task has three classes: Entailment, Neutral, and Contradiction.

Steps:

1. **Preparing the data**
   We loaded the dataset (txt format) and converted them into pandas data frame for easy data manipulation. For task 1, we extracted premise and hypothesis sentences as X train and entailment judgment as Y train. The target values (entailment, neutral and contradiction) are categorical and hence we used dummy variables and converted them into numeric values. We repeated the same steps for validation and test data as well.

2. **Word Embedding**
   We mapped the words in the sentences to vectors as machine learning and almost all deep learning models are not able to process strings or plain text in their raw form. Few of the word embeddings that we worked with for this task are listed as follows.

   - GloVe
     We used GloVe embedding's which is based on factorizing a matrix of word co-occurrence statistics. We experimented on all dimensions of glove vectors and decided to work with 300-dimensional GloVe vectors as they provided the best results for our task.

   - Building our own Embedding from dataset
     Using NLTK and gensim's API we created our own embeddings which would be specific to our dataset. The dataset considered for this is training and validation data. We cleaned the dataset in order to remove unnecessary words (stopwords, punctuation etc.) and used only meaningful words that occurred in the dataset to prepare word embeddings.

   - Keras Embedding Layer
     Keras offers an Embedding layer that can be used for neural networks on text data. The Embedding layer is defined as the first hidden layer of a network with three arguments – input dimension, output dimension and input length.

   Out of all three, the best results were obtained using 300-dimensional GloVe embedding of 2.2m words.

3. **Data Representation**

   Using word embeddings, we transformed our text features in form of vectors. We form tensors from these word vectors which can be fed in the neutral network. Sequences that are short are padded with value 0. We do not update the pre-trained word embeddings during training.

   Example of encoding of a sentence to vectors using 300D GloVe-

   Text: A group of kids is playing in a yard and an old man is standing in the background

   Word sequence: [1, 63, 10, 114, 2, 12, 5, 1, 212, 6, 19, 271, 4, 2, 21, 5, 3, 213]

   ```
   Pad: [ 0   0   1 63  10 114   2  12   5   1 212   6  19 271   4   2  21   5   3 213]
   ```

4. **Model**

   a. Word Representation

      The embedding layer acts as first layer, this takes input as sentence and convert a 300-dimensional vector from the word embedding for each word in the input sentence.

   b. Context Representation

      In this layer, we apply bi-directional LSTM to incorporate contextual information into representation at each time step.

   c. Matching Layer

      This layer compares contextual embeddings of each sentence with contextual embeddings of all other sentences using cosine matching function. This strategy is used at each time step and then concatenate the generated four vectors as matching vector for that time step.

   d. Aggregation Layer

      This layer aggregates the two sequence of matching vectors into a fix-length matching vector and then we apply bi directional LSTM.

   e. Prediction Layer

      At last, we apply softmax function which calculates probability distribution for each of the target variables.

5. Predictions

   The output of softmax function is probability distribution for three classes and hence we have to convert them to their respective class. We determine the maximum probability outcome and map them as either Neutral, Contradiction or Entailment.

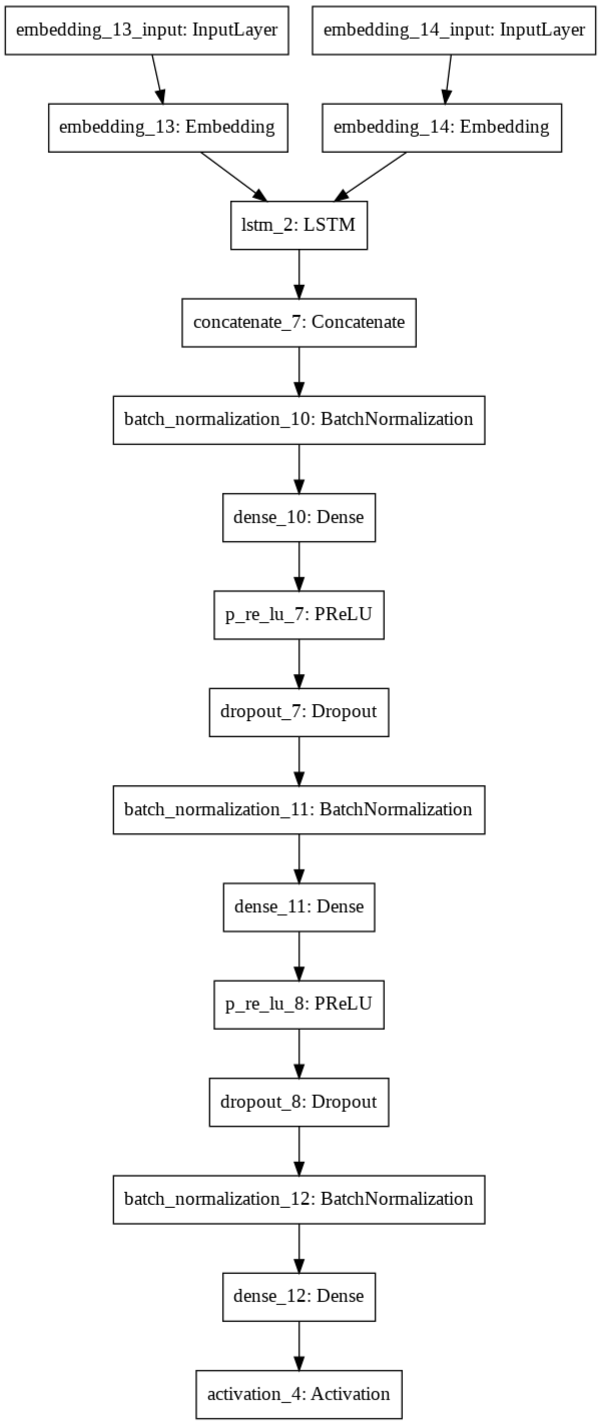
6. Evaluation

   We have used accuracy, precision, recall and F Score as metrics for evaluation of our model.

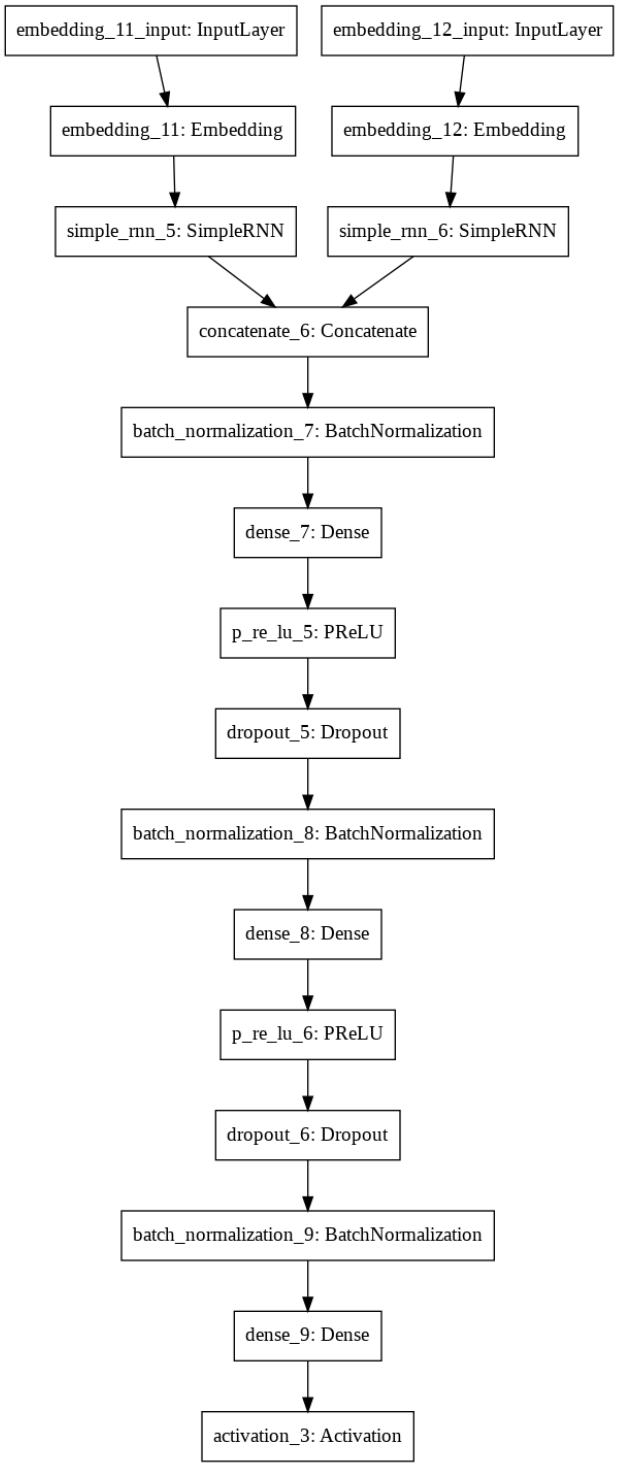
Models tried

We tried four different models, each of which had different outcome and architecture. The best results were obtained using Bi-directional LSTM.
Below are the architecture of the tried models.

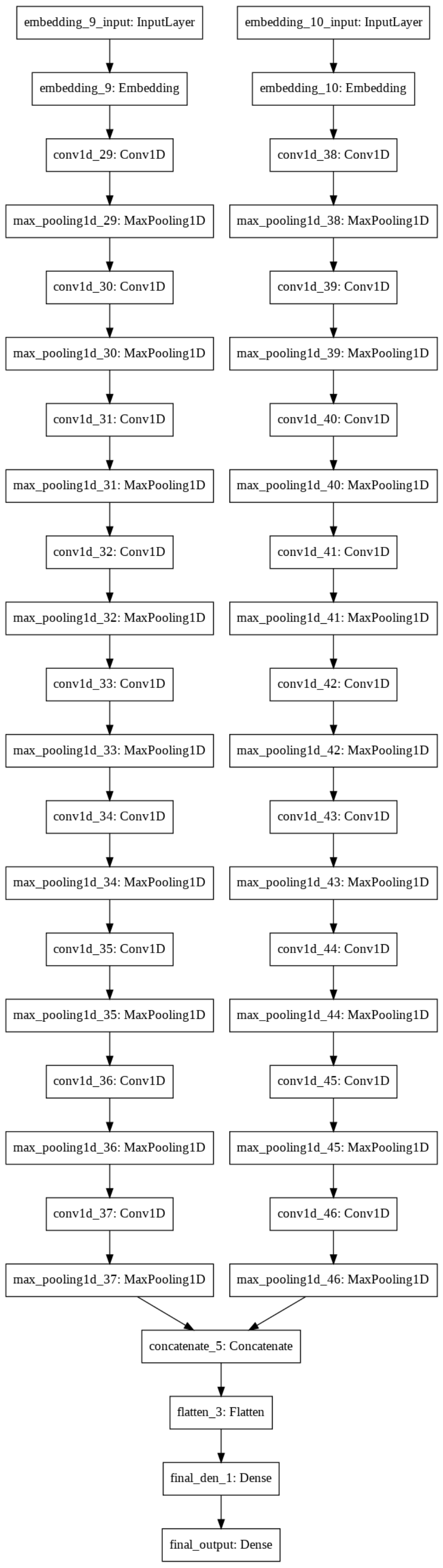
- Long Short Term Memory (LSTM)


embedding_13_input: InputLayer
embedding_14_input: InputLayer
embedding_13: Embedding
embedding_14: Embedding
lstm_2: LSTM
concatenate_7: Concatenate
batch_normalization_10: BatchNormalization
dense_10: Dense
p_re_lu_7: PReLU
dropout_7: Dropout
batch_normalization_11: BatchNormalization
dense_11: Dense
p_re_lu_8: PReLU
dropout_8: Dropout
batch_normalization_12: BatchNormalization
dense_12: Dense
activation_4: Activation

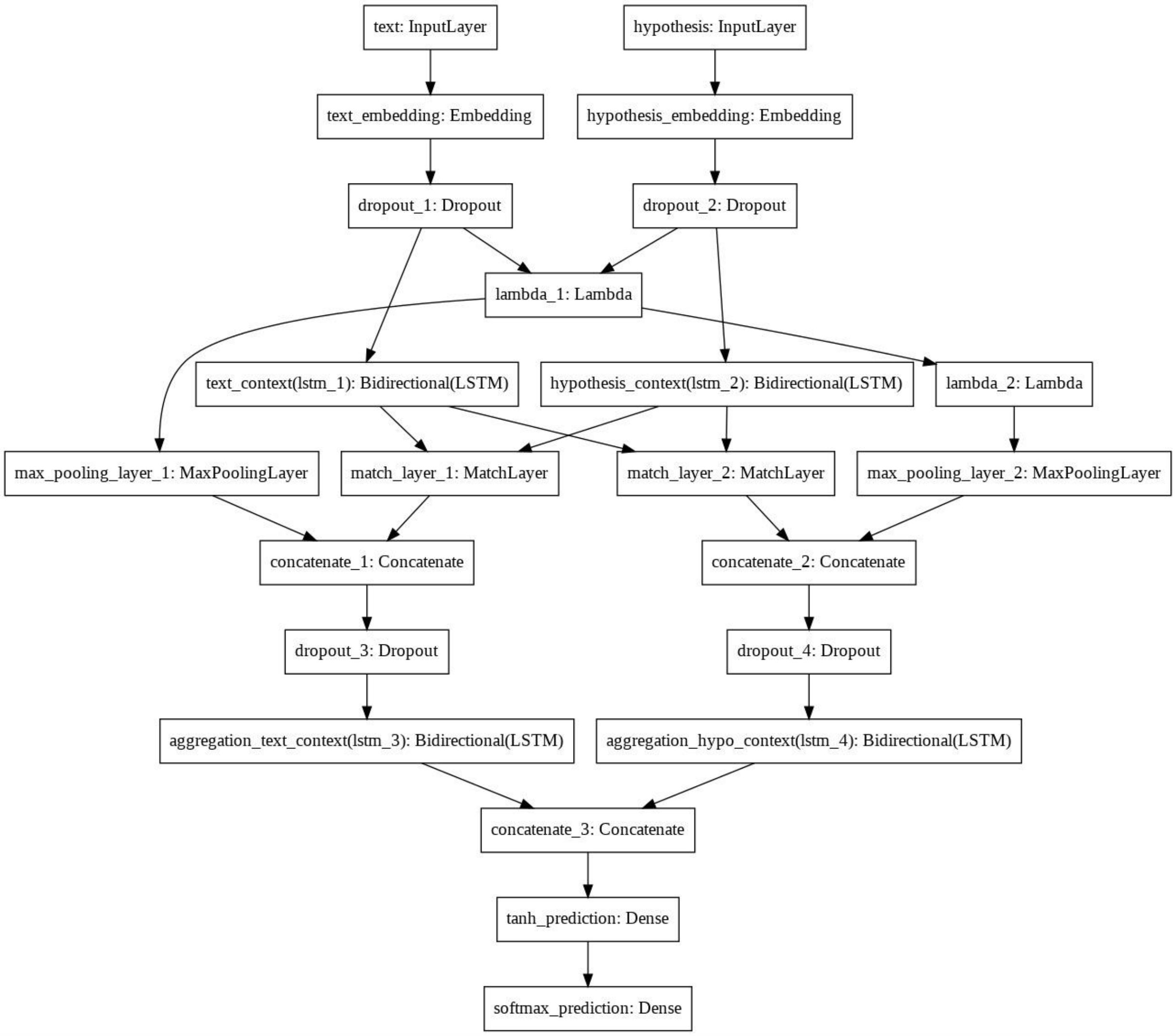
- Recurrent Neural Network (RNN)



- <u>Convolution Neural Network (CNN)</u>


embedding_9_input: InputLayer
embedding_9: Embedding
conv1d_29: Conv1D
max_pooling1d_29: MaxPooling1D
conv1d_30: Conv1D
max_pooling1d_30: MaxPooling1D
conv1d_31: Conv1D
max_pooling1d_31: MaxPooling1D
conv1d_32: Conv1D
max_pooling1d_32: MaxPooling1D
conv1d_33: Conv1D
max_pooling1d_33: MaxPooling1D
conv1d_34: Conv1D
max_pooling1d_34: MaxPooling1D
conv1d_35: Conv1D
max_pooling1d_35: MaxPooling1D
conv1d_36: Conv1D
max_pooling1d_36: MaxPooling1D
conv1d_37: Conv1D
max_pooling1d_37: MaxPooling1D
embedding_10_input: InputLayer
embedding_10: Embedding
conv1d_38: Conv1D
max_pooling1d_38: MaxPooling1D
conv1d_39: Conv1D
max_pooling1d_39: MaxPooling1D
conv1d_40: Conv1D
max_pooling1d_40: MaxPooling1D
conv1d_41: Conv1D
max_pooling1d_41: MaxPooling1D
conv1d_42: Conv1D
max_pooling1d_42: MaxPooling1D
conv1d_43: Conv1D
max_pooling1d_43: MaxPooling1D
conv1d_44: Conv1D
max_pooling1d_44: MaxPooling1D
conv1d_45: Conv1D
max_pooling1d_45: MaxPooling1D
conv1d_46: Conv1D
max_pooling1d_46: MaxPooling1D
concatenate_5: Concatenate
flatten_3: Flatten
final_den_1: Dense
final_output: Dense

- <u>Bidirectional LSTM</u>
  This model is used to retain the contextual information from embeddings. It uses two LSTM that run in parallel. The first BiLSTM is used to encode the premise sentences and the second one for the hypothesis. Then using the matching function each premise was matched with the hypothesis. Another BiLSTM is then used to combine the result of the first two. The output is then passed through a fully connected layer for classification followed by a softmax function as we had three classes
  Other Details –
  Dropout - 0.2 was applied throughout the model; Batch size – 25; Stochastic Optimizer – ADAM, Learning Rate – 0.001; Epsilon - 1e-08; beta_1 - 0.9, beta_2 - 0.999 and loss function – categorical cross entropy. **This model led to the best results**.


text: InputLayer
hypothesis: InputLayer
text_embedding: Embedding
hypothesis_embedding: Embedding
dropout_1: Dropout
dropout_2: Dropout
lambda_1: Lambda
text_context(lstm_1): Bidirectional(LSTM)
hypothesis_context(lstm_2): Bidirectional(LSTM)
lambda_2: Lambda
max_pooling_layer_1: MaxPoolingLayer
match_layer_1: MatchLayer
match_layer_2: MatchLayer
max_pooling_layer_2: MaxPoolingLayer
concatenate_1: Concatenate
concatenate_2: Concatenate
dropout_3: Dropout
dropout_4: Dropout
aggregation_text_context(lstm_3): Bidirectional(LSTM)
aggregation_hypo_context(lstm_4): Bidirectional(LSTM)
concatenate_3: Concatenate
tanh_prediction: Dense
softmax_prediction: Dense

<u>Experiments</u>

The following table shows the evaluation result, in terms of accuracy, on the SICK dataset we used in our experiments with different models.

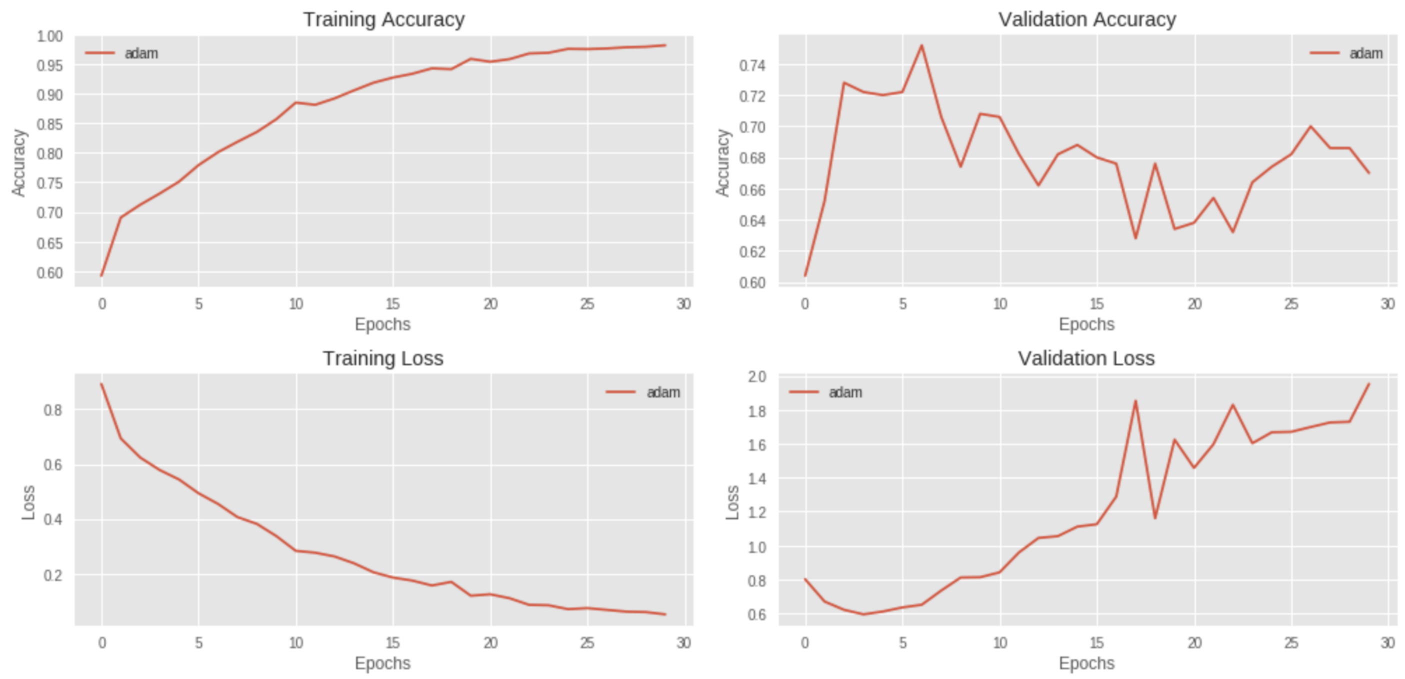
Model	Epoch	Batch Size	Embeddings	Dropout rate	Accuracy (%)
LSTM	8	240	Keras	0.2	49.9
LSTM	10	240	Keras	0.2	54.88
LSTM	8	124	Keras	0.2	55.26
LSTM	8	124	Keras	0.25	56.50
LSTM	8	240	Custom	0.25	58.04
LSTM	10	240	Custom	0.2	54.88
LSTM	8	124	Custom	0.2	56.98
LSTM	8	124	Custom	0.25	57.58
LSTM	10	124	GloVe	0.25	**61.10**
CNN	20	124	Keras	0.2	48.04
CNN	7	124	Keras	0.2	50.98
CNN	2	124	Custom	0.25	56.68
CNN	5	124	GloVe	0.25	56.78
CNN	8	124	GloVe	0.25	**59.10**
RNN	10	124	Glove	0.25	51.8
Bi-directional LSTM	**30**	**124**	**GloVe**	**0.2**	**69.69**

The best results were obtained from bi-directional LSTM. We have submitted the results from the same in Results.txt

The following table shows the evaluation result on the SICK dataset on Bi-directional LSTM for Text Entailment task.

	Precision	Recall	F1-Score	Support
CONTRADICTION	0.67	0.72	0.69	720
ENTAILMENT	0.66	0.48	0.56	1414
NEUTRAL	0.72	0.80	0.76	2793
Micro avg	0.70	0.70	0.70	4927
Macro avg	0.68	0.67	0.67	4927
Weighted avg	0.69	0.70	0.69	4927

Below are the graphs depicting training behavior of our model.



Other Method:

<u>Oversampling of data</u>: We observed in our exploration step that the majority of samples were of the class 'Neutral'. We handled this imbalance is our dataset by using oversampling technique called SMOTE.

SMOTE: It creates new instances of the minority class by forming convex combinations of the neighboring instances. This allows us to balance our dataset without overfitting, as we create new synthetic examples instead of using duplicate samples. We applied SMOTE twice, in the first SMOTE oversampling, we were able to generate synthetic samples for the class 'Contradiction'. After the first set of oversampling, the data set was still imbalanced with minority class 'Entailment'. So, we had to apply SMOTE once more so that we have enough samples for all the three classes.

Using this technique with LSTM, RNN and CNN, gave better results. However, the results were not that impressive when used with Bi-directional LSTM. Note: The best accuracy obtained and reported in result.txt is without using this technique.

Task 2: Semantic Relatedness

Given two sentences, one named sentence_A (Premise) and one named sentence_B (Hypothesis), predict the degree of relatedness between two sentences.

Steps:

1. Preparing the data
   For task 2, we extracted premise and hypothesis sentences as X train and semantic_relatedness as Y train. As we have used sigmoid as our activation function which outputs results between 0 to 1, we have to normalize the target value in range of 0 to 1 using min-max scaling technique.

2. Word Embedding – *Same as Task 1*

3. Data Representation – *Same as Task 1*

4. Model
   We used the same model here that gave us the best results for task 1. We changed the prediction layer from softmax to sigmoid as here we do have three target variables. Rest all the layers as same as we had for task 1.

5. Prediction
   The output of sigmoid function lies between 0 to 1 and hence we need to rescale our output in range of 1 to 5.

6. Evaluation
   We have used Pearson Correlation and Spearman Correlation as metrics for evaluation of our model. We also considered mean square error while training and testing our model.

The following table shows the evaluation result on the SICK dataset using Bi-directional LSTM for semantic relatedness task.

	Pearson Correlation	**Spearman Correlation**
Bi-directional LSTM	0.7249	0.661

Resources / Tools used

- We used pandas, numpy, matplotlib, sckit learn, NLTK, Keras and TensorFlow for this assignment.
- For word embeddings, we used GloVe and Keras Word Embeddings.
- We used Google Colab for developing and training our models.
- We used the Bidirectional layer wrapper provided in keras to implement a bidirectional LSTM.
- We used the min-max scale provided by sklearn library.
- https://www.oreilly.com/learning/textual-entailment-with-tensorflow
- https://github.com/kiril-me/rep-task/blob/master/Recognizing%20Textual%20Entailment.pdf

How to run?

1. Download the code (Jupyter Notebook or .py file)
2. Download the dataset (SICK_train.txt, SICK_trial.txt, SICK_test_annotated.txt)
3. Make sure all the files downloaded in step 1 & 2 are in same directory.
4. Run the code.
5. We used Google Colab for developing the code. However, the code can also be ran using any python supported IDE.

Please note, the code has steps to download the GloVe Embeddings (the downloaded files will be saved in a folder name datasets) and hence may get slow while executing those steps.

Distribution of Tasks

	Task	Owner
	Dataset Exploration	Nikhil, Deepshi
Task 1 Text Entailment	Features Engineering	Nikhil, Deepshi
	Embeddings - Keras	Deepshi
	Embeddings - Dataset	Nikhil
	Embeddings - Glove	Nikhil
	Oversampling	Nikhil, Deepshi
	Models - LSTM	Deepshi
	Models - CNN	Nikhil, Deepshi
	Models - RNN	Nikhil
	Models – Bidirectional LSTM	Nikhil, Deepshi
	Evaluation Metrics	Nikhil
Task 2 Semantic Relatedness	Feature Engineering	Deepshi
	Model – Bidirectional LSTM	Deepshi
	Evaluation Metrics	Deepshi
	Report	Nikhil, Deepshi