

# NLP Final Report

2019101008 - Raghav Raj Dwivedi

2019101028 - Aakash Jain

2019101029 - Y. Sai Sriram

## 1 Introduction

### 1.1 Problem Statement

A fundamental phenomenon of natural language is the variability of semantic expression, where the same meaning can be expressed by or inferred from different texts. Textual entailment recognition is the task of deciding, given two text fragments, whether the meaning of one text is entailed (can be inferred) from another text. Given two text fragments, one named text (t) and the other named hypothesis (h), respectively. The task consists in recognizing whether the hypothesis can be inferred from the text. Textual entailment has a three-class balanced classification problem over sentence pairs:

- Entailment
- Neutral
- Contradiction

### 1.2 Datasets

- SNLI: <https://nlp.stanford.edu/projects/snli/>
  - 530k sentence pairs
  - Stanford had custom sentence pairs manually written out by labelled by humans, that being SNLI.
- MultiNLI: <https://cims.nyu.edu/~sbowman/multinli/>
  - 433k sentence pairs
  - MultiNLI is a crowd-sourced dataset, which differs from SNLI where it has more cross-genre generalization, and a more varied dataset.

- SciTail: <https://allenai.org/data/scitail>
  - 27k sentence pairs
  - SciTail was created from science multiple choice quizzes, where each question and correct answer pair is labelled as entailment.
- SICK: <https://paperswithcode.com/dataset/sick>
  - 10k sentence pairs
  - SICK was created by taking datasets which have sentences that describe particular images, and labelling these descriptions.

### 1.3 Relevant Readings

<https://aclanthology.org/W17-5301/>

<https://arxiv.org/pdf/1509.06664v1.pdf>

[https://nlp.stanford.edu/pubs/snli\\_paper.pdf](https://nlp.stanford.edu/pubs/snli_paper.pdf)

In order to begin the project, we had to read up on NLI, understand the basis of textual entailment, and the methods and standard practices that go into this classification task. We looked into papers that did specific implementations using Bowman models and LSTMs. We looked into how the construction of these attention models is usually done in both TensorFlow and PyTorch.

We looked into preprocessing of the data in various ways, such as BERT, RoBERTa, GloVe, and word2vec. We used BERT in our attention model for the mid-semester evaluations but had switched over to Glove for our next two versions of the implementation, and word2vec for our final implementation.

We upgraded from our mid-semester attention model to a Bowman model that uses a simple summation layer. We then replaced the summation layer with LSTMs and got better performance across all models. These LSTMs run in parallel for both premise and hypothesis vectors. However, in our final implementation, we used two sequential LSTMs, the first one inputting the premise vector and outputting its internal states as the initialisation to the next LSTM, which also inputs the hypothesis vectors.

For our larger datasets, the train accuracies have gone up from the mid 70% range to the lower 80% range while our test accuracies are also the same. For our smaller datasets, our train accuracies are in the 90% range while our test accuracies are in the

low 70% range, a sign that the smaller datasets make the model overfit on the training data slightly.

## **2 Implementation**

### **Model 1: Using a Summation Layer**

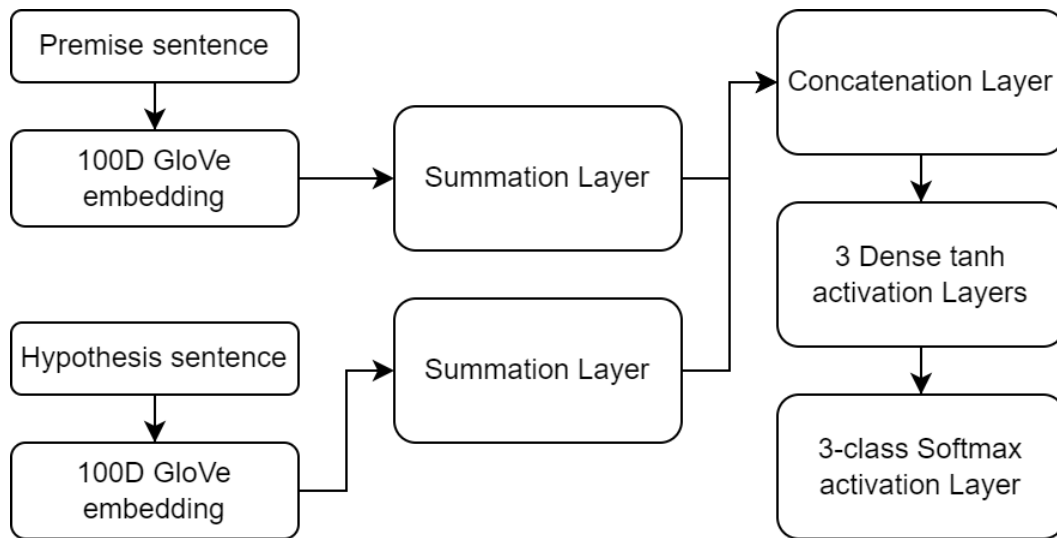
Our first model is a Bowman model using a simple summation layer.

The first component of the model is a word embedding layer that is set to be non trainable. The embedding matrix for this layer is initialized with the 300 dimensional glove embeddings trained on 840 billion tokens. It converts the vocabulary sized input dimension (around 30K) into 300 dimensions.

The update to the model from the previous submission is that previously, 100D glove vectors were directly used. Now, the embedded premise and hypothesis are sent into another layer that projects 300D vectors into a 100 dimensional space, followed by the application of tanh activation.

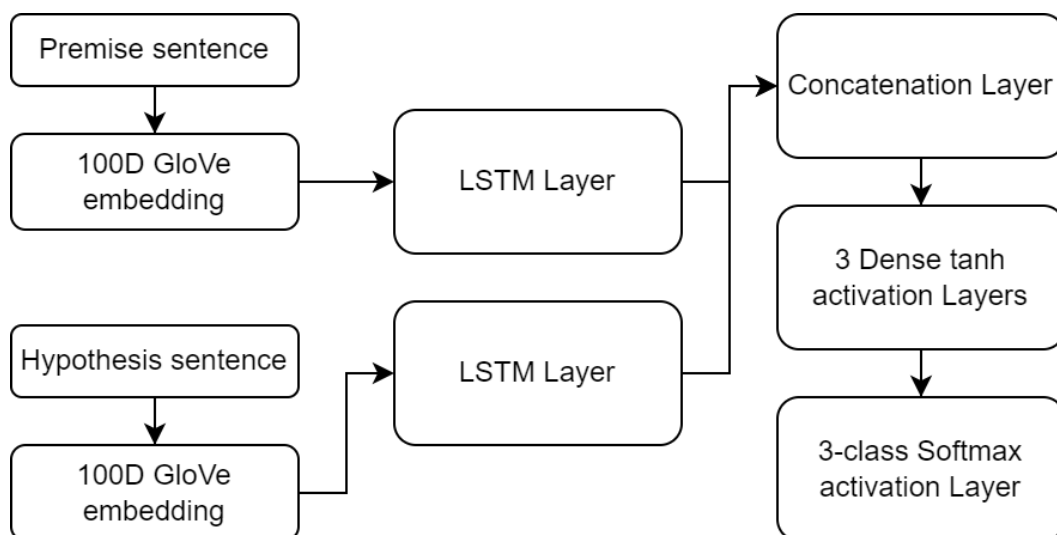
In the case of the simple summation layer, for both the premise and hypothesis the vectors along the sentence axis are summed. That is, maxLen number of tokens are summed to give a single 100 dimensional representation of the premise/hypothesis.

Followed by this is a concatenation layer that simple appends the hypothesis to the premise. A combination of fully connected layers with tanh activation and dropout layers are applied successively, before finally feeding the resultant 200 dimensional vector into a fully connected layer with softmax activation over 3 labels.



## Model 2: Using an LSTM

We replace the summation layer from the previous model with an LSTM to make our second model. In this case, both the premise and hypothesis are sent into the LSTM with dropout and recurrent dropout both set to 0.2. Everything else regarding the model remains the same.



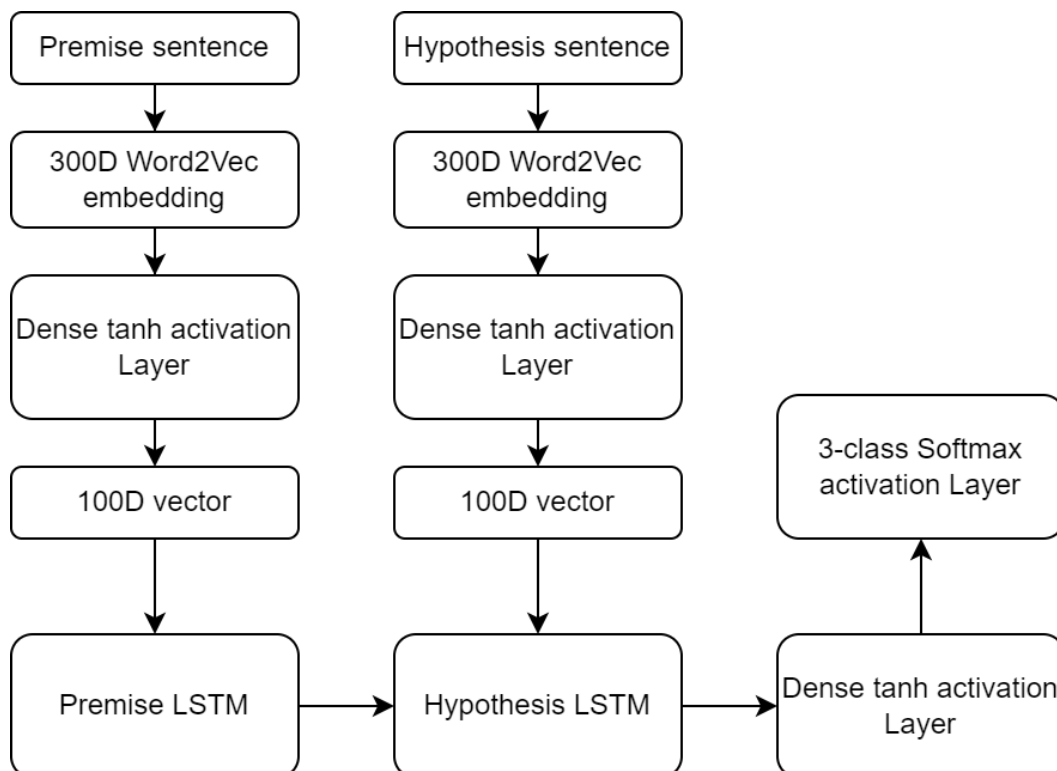
## Model 3: Using two LSTMS

In our final model, we utilise two LSTMs as well as word2vec instead of glove.

The first component of the model is a word embedding layer that is set to be non trainable. The embedding matrix for this layer is initialized with the 300D word2vec vectors trained on Google News. It converts the vocabulary sized input dimension (around 30K) into 300 dimensions.

The embedded versions of the premise and hypothesis are then sent into a fully connected layer activated by the tanh function, to convert the 300-dimensional representations into 100 dimensions.

The first LSTM in this model is for premise and the second is for hypothesis. The one for the premise is set to return its final internal states. These states are then used to initialize the second LSTM - this is an attempt to make the hypothesis' LSTM be aware of the context given by the premise. Unlike the previous models, there is no concatenation here; the final 100 dimensional representation of the hypothesis is fed into a fully connected layer of tanh activation, followed by another fully connected layer with softmax activation over three output labels.



## 3 Conclusion

### 3.1 Results

- Summation Model:

Accuracy	Training	Validation	Test
SNLI	74%	77%	76%
MultiNLI	65.39%	67.23%	65.74%
SciTail	87.88%	70.68%	71.62%
SICK	88.21%	70.84%	70.49%

- LSTM Model:

Accuracy	Training	Validation	Test
SNLI	75.8%	78.21%	77.3%
MultiNLI	67.41%	65.98%	65.76%
SciTail	83.77%	72.83%	74.58%
SICK	95.66%	73.14%	72.65%

- Two LSTMs Model:

Accuracy	Training	Validation	Test
SNLI	81%	80%	80.13%
MultiNLI	70.29%	68.65%	67.47%
SciTail	97.97%	70.53%	74.58%
SICK	98.60%	58.79%	60.57%

- Test accuracies across models:

Model	Summation	LSTM	2LSTM
SNLI	76%	77.3%	80.13%
MultiNLI	65.74%	65.76%	67.47%

Model	Summation	LSTM	2LSTM
SciTail	71.62%	74.58%	74.58%
SICK	70.49%	72.65%	60.57%

## 3.2 Inferences and Limitations

- We can see the continuous improvements that the LSTM and shared LSTM models have over the baseline bowman model. The implementations are based on the papers that they were explained in, and we have very similar results for all models.
- Currently, most of the models are likely to overfit in the smaller datasets, SciTail and more significantly SICK. While the training and test accuracy values are comparable and expected for the other datasets, we see that our models, particularly the LSTM ones, have quite a high value for train accuracy, while the test accuracy is much lower.
- The reasoning for the high overfit and low performance of the LSTM models in SICK and SciTail is likely due to two factors:
  - Lack of Sufficient Data: The datasets are significantly smaller than both the SNLI and MultiNLI datasets, and there isn't enough data present to populate the model adequately
  - Nature of the Dataset: The SciTail and SICK dataset are structured significantly differently from the other two, with different contexts and origins, and the models presented, that were based on the SNLI dataset, may not be optimal in this case.
- The MultiNLI dataset has inconsistent results. We see that it is steadily poorer than SNLI, and the accuracy gain of even the shared LSTM model isn't significant. This, again, is likely due to two reasons:
  - Nature of the Dataset: The MultiNLI dataset differs from SNLI in that it is much broader in terms of domain and genres of samples. As a result, it is harder for the model to pick up on the context and semantics of the sentences, and thus entailment becomes a more challenging task.
  - Word Embeddings Used: Here, the Google News Word2Vec embedding has been used. It is likely that this embedding isn't suitable for the task, and sentence transformers like BERT or RoBERTa, are more suitable

- Due to the extensive training time that is required, particularly for the large datasets, and also the large number of models that are being run, to have a comparative analysis of all models over all datasets, the project doesn't delve sufficiently into hyperparameter optimization of the models, or altering of its structure for each dataset. Optimal patience values for early stopping have been experimented with.