Final Project report

1. Abstract / Problem definition:

Natural language inference (NLI) refers to the problem of determining entailment and contradiction relationships between a premise and a hypothesis. NLI is a central problem in language understanding. The task is that of comparing two sentences and identifying the relationship between them.

Textual entailment is a directional relation between text fragments. The relation holds whenever the truth of one text fragment follows from another text. In the TE framework, the entailing and entailed texts are termed *text (t)* and *hypothesis (h)*, respectively.

2. Literature survey and Related work:

1. Structured Self-Attentive Sentence Embedding
2. Augment word representations with character level feature
3. DCN (Dynamic Co-Attentive Networks for Question Answering)
4. Neural machine translation by jointly learning to align and translate
5. Effective Approaches to Attention-based Neural Machine Translation

3. Datasets used:

1) SNLI:

The SNLI corpus (version 1.0) is a collection of nearly 570k human-written English sentence pairs manually labeled for balanced classification with the labels *entailment*, *contradiction*, and *neutral*, supporting the task of natural language inference (NLI).It known to serve both as a benchmark for evaluating representational systems for text, especially including those induced by representation learning methods, as well as a resource for developing NLP models of any kind. SNLI dataset looks as follows.

2) SICK:

The SICK data set consists of about 10,000 English sentence pairs, generated starting from two existing sets: the 8K ImageFlickr data set and the SemEval 2012 STS MSR-Video Description data set. Each sentence pair was annotated for relatedness and entailment by means of crowdsourcing techniques. The sentence relatedness score (on a 5-point rating scale) provides a direct way to evaluate Computational Distributional Semantic Models(CDSMs), insofar as their outputs are meant to quantify the degree of semantic relatedness between sentences; the categorizations in terms of the entailment relation between the two sentences (with *entailment*, *contradiction*, and *neutral* as gold labels) is also a crucial aspect to consider, since detecting the presence of entailment is one of the traditional benchmarks of a successful semantic system.

4. Approach:

1) The algorithm consists of three main parts:
    a. Getting fixed-length embeddings for hypothesis and premise sentences using ***Embedding Model***(BiLSTM and self-attention mechanism)
    b. Extracting the relation between the two sentence embeddings by using ***Gated AutoEncoder***
    c. Using ***MLP*** on the output of Gated AutoEncoder to classify the pair of sentences into one of the three types(Entailment, Neutral, Contradiction)

2) Embedding Model:
    a. The first part is a bidirectional LSTM(BiLSTM), and the second part is the self-attention mechanism, which provides a set of summation weight vectors for the LSTM hidden states.
    b. These set of summation weight vectors are dotted with the LSTM hidden states, and the resulting weighted LSTM hidden states are considered as an embedding for the sentence.
    c. Suppose we have a sentence, which has n tokens,represented in a sequence of word embeddings.

    $$S = (w_1 , w_2 , \cdots w_n)$$

    Here wi is a vector standing for a d dimensional word embedding for the i-th word in the sentence. S is thus a sequence represented as a 2-D matrix, which concatenates all the word embeddings together. S should have the shape n-by-d.
    d. Now each entry in the sequence S are independent with each other. To gain some dependency between adjacent words within a single sentence, a bidirectional LSTM can be used.

    $$h_{tf} = fLSTM (w_t , h_{(t-1)f})$$

$$h_{tb} = \text{bLSTM}(w_t, h_{(t+1)b})$$

e. And then, we concatenate h_{tf} with h_{tb} to obtain a hidden state h_t. Let the hidden unit number for each unidirectional LSTM be u. For simplicity, we note all the n h_ts as H, who have the size n-by-2u.

$$H = (h_1, h_2, \cdots h_n)$$

f. Our aim is to encode a variable length sentence into a fixed size embedding. We achieve that by choosing a linear combination of the n LSTM hidden vectors in H. Computing the linear combination requires the self-attention mechanism. The attention mechanism takes the whole LSTM hidden states H as input, and outputs a vector of weights a.

$$a = \text{softmax}(w_{s2}(\tanh(W_{s1}H^T)))$$

g. Here W_{s1} is a weight matrix with a shape of d_a-by-2u and w_{s2} is a vector of parameters with size d_a ,where d_a is a hyperparameter we can set arbitrarily. Since H is sized n-by-2u, the annotation vector a will have a size n.

h. This vector representation usually focuses on a specific component of the sentence, like a special set of related words or phrases. So it is expected to reflect an aspect, or component of the semantics in a sentence. However, there can be multiple components in a sentence that together forms the overall semantics of the whole sentence, especially for long sentences. (For example, two clauses linked together by an ”and.”)

i. Thus, to represent the overall semantics of the sentence, we need multiple m’s that focus on different parts of the sentence.Thus we need to perform multiple hops of attention. Say we want r different parts to be extracted from the sentence, with regard to this, we extend the w_{s2} into a r-by-d_a matrix, note it as W_{s2} , and the resulting annotation vector a becomes annotation matrix A. Formally, $A = \text{softmax}(W_{s2}(\tanh(W_{s1}H^T)))$

j. The embedding vector m now becomes an r-by-2u embedding matrix M . We compute the r weighted sums by multiplying the annotation matrix A and LSTM hidden states H, the resulting matrix is the sentence embedding:

$$M = AH$$

3) Gated AutoEncoder:

a. For both hypothesis and premise, we extract their embeddings (M_h and M_p in the figure) independently, with the same LSTM and attention mechanism.

b. Doing the batched dot for both hypothesis embedding and premise embedding, we have F_h and F_p.

F_h = batcheddot(M_h , W_{fh})

F_p = batcheddot(M_p , W_{fp})

Here W_{fh} and W_{fp} are the two weight tensors for hypothesis embedding and premise embedding.

c. The factor of the relation (F_r) is just an element-wise product of F_h and F_p

$$F_r = F_h \,.\, F_p$$

F_r captures the relation between the hypothesis and premise

4) MLP:

a. On top of F_r layer, we use an MLP with softmax output to classify the relation into different categories(Entailment, Neutral, Contradiction)

5. Improvisations:

A. Modified Word Embeddings using FastText

1. We improvised the existing model by changing the approach to computing the word embeddings.
2. Previously we used Glove embeddings (GloVe: Global Vectors for Word Representation). GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. We used pre-trained model of GloVe previously.
3. In a character embedding model, the vector for a word is constructed from the character n grams that compose it.
4. Since character n grams are shared across words, these models do better than word embedding models for out of vocabulary words - they can generate an embedding for an OOV word. Word embedding models like word2vec cannot since they treat a word atomically. In english, all words are formed by 26 (or 52 if including both upper and lower case character, or even more if including special characters)characters. Having the character embedding,

every single word's vector can be formed even it is out-of-vocabulary words . On the other hand, word embedding can only handle those seen words.

5. It handles infrequent words better than other embedding mechanisms as models like word2vec or GLove suffer from lack of enough training opportunity for those rare words.
6. **Embeddings using Fasttext:** FastText is an extension to Word2Vec proposed by Facebook in 2016. It is a character embedding model. Instead of feeding individual words into the Neural Network, FastText breaks words into several n-grams (sub-words). For instance, the tri-grams for the word *apple* is *app*, *ppl*, and *ple* (ignoring the starting and ending of boundaries of words). The word embedding vector for *apple* will be the sum of all these n-grams. After training the Neural Network, we will have word embeddings for all the n-grams given the training dataset. Rare words can now be properly represented since it is highly likely that some of their n-grams also appears in other words.
7. It is helpful to find the vector representation for rare words. Since rare words could still be broken into character n-grams, they could share these n-grams with the common words
8. It can give the vector representations for the words not present in the dictionary (OOV words) since these can also be broken down into character n-grams. word2vec and glove both fail to provide any vector representations for words not in the dictionary.
   For example, for a word like *stupedofantabulouslyfantastic*, which might never have been in any corpus. FastText can produce vectors better than random by breaking the above word in chunks and using the vectors for those chunks to create a final vector for the word. In this particular case, the final vector might be closer to the vectors of fantastic and fantabulous.

B. Modified Architecture using Dynamic Co-attention Networks:

1. We improved the current accuracy by changing the architecture similar to the model used in Dynamic Co-attention Network.
2. Instead of using hidden state of LSTM of only one sentence for calculating attention, we now used a function of hypothesis and premise LSTM hidden states

3. i.e. previously, A_h = softmax(W_{s2}(tanh($W_{s1}H_h^T$))) and A_p = softmax(W_{s2}(tanh($W_{s1}H_p^T$))); now
    a. A_h = softmax(W_{s2}(tanh($W_{s1}H_{h\_new}^T$))) where $H_{h\_new} = H_pH_h^T$
    b. A_p = softmax(W_{s2}(tanh($W_{s1}H_{p\_new}^T$))) where $H_{p\_new} = H_hH_p^T$
4. The rest of the steps are same as the original model.

6. Experimental settings:

1. We word-tokenized all the train,validation and test data using nltk and then obtained word embeddings of size 300 from fastext(in improvised model), glove(in baseline model)
2. Next, we trained a Bidirectional LSTM , each of 150 hidden units and 1 layer on each sentence pair of training data. In training data, some sentence pairs are not classified into any of the three categories, such sentences are not considered while training. Size of each hidden state in BiLSTM is 150+150=300
3. Next, we trained a self-attention model on the hidden states of the BiLSTM. We set the number of hidden states in attention layer as 350 and the number of output rows(r) as 10. Attention penalty is set to 1.0
4. Sentence embedding is calculated by doing product of BiLSTM output and Self-Attention model output
5. A gated encoder is trained on sentence embeddings obtained using above method. The output of Gated Encoder module is sent to MLP with number of hidden states 3000 and output size 3. A softmax layer is applied on output to get the category to which the pair of sentences belong.
6. The two hidden states(matrices W_{fh}, W_{fp}) in Gated Encoder are each of shape r-by-2u-by-2u (2u=Size of each hidden state in BiLSTM) i.e. 10-by-300-by-300
7. Training is done in batches with batch-size as 50, learning rate as 0.01 and the error converges after 2 epochs.
8. Dropout of 0.5 is used on Gated Encoder input to get better results. The optimizer used is ADA.

7. Results for baseline model:

1) SNLI:

a. In SNLI, the total training dataset contains approximately 550k valid sentence pairs, validation set and test set have 10k pairs each.
b. 550k pairs is a large dataset and it is taking more than 8 hours to run one epoch on 550k pairs. So, we were able to run only two epochs on full dataset.
c. The final accuracies are as follows:

Train	80%
Validation	83%
Test	82%

d. We didn't use the hyperparameters mentioned in paper(which give maximum possible accuracy according to authors), because, we couldn't even run even on half-dataset when we used those hyperparameters. Hence, we deviated from the parameters mentioned in the paper.
e. The hyper-parameters values used are:

LSTM_HIDDEN	150 instead of 300
ATTENTION_HIDDEN	350 instead of 150
N_ROWS	10 instead of 30
MLP_HIDDEN	3000 instead of 4000

2) SICK:

a. After running 20 epochs on approximately 4100 train and 4100 test data, the accuracies obtained are as follows:

Train	99.8%
Validation	75%
Test	74.5%

b. Hyperparameters used are same as those used for SNLI dataset.

8. Results after improvisations:

a. On the improved model, for SNLI dataset, the final accuracies are as follows:

Train	89%
Validation	84%
Test	83.5%

b. On the improved model, for SICK dataset, the final accuracies are as follows:

    Train : 99.84%
    Validation : 76%
    Test : 75.5%

c. Hence, after the introduction of fastext and inter-sentence level attention into the baseline model, the accuracies improved significantly.