NLA Project Report

Pulle Sri Satya Sravya
Sushmitha Rajeswari Muppa

A. Project Description:
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.

B. Implementation choices
There are two baselines that can be implemented (as per the given project list):

1. Structured Self-Attentive Sentence Embedding
2. A Decomposable Attention Model for Natural Language Inference

We chose to implement “Structured Self-Attentive Sentence Embedding” as the approach of this paper seems novel and interesting.

C. Algorithm of Structured Self-Attentive Sentence Embedding:

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.

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

    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.

    $$h_{tf} = fLSTM (w_t , h_{(t-1)f})$$
    $$h_{tb} = bLSTM (w_t , h_{(t+1)b})$$

    fLSTM -> forward LSTM
    bLSTM -> backward LSTM

    e. And then, we concatenate each 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.

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

    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.

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

    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,

$$A = \mathrm{softmax}(W_{s2}(\tanh(W_{s1}H^T)))$$

j. The embedding vector m then 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:

$$M = AH$$

k. u and r are independent of sentence length. Therefore, we get a fixed-length matrix for any given sentence once we fix r(The number of sentence parts we want to extract)

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

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

d. 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)

D. Implementation details:

1. We word-tokenized all the train,validation and test data using nltk and then obtained word embeddings of size 300 from Glove. Some words have no embeddings in glove, for such words we calculated embeddings using neighbouring words. Since there are less un-embedded words in the whole dataset, we didn’t use any neural network to get embeddings. Instead, we obtained embedding by averaging surrounding words embeddings.
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 2-3 epochs.
8. Dropout of 0.5 is used on Gated Encoder input to get better results. The optimizer used is ADA.
9. Self-attention mechanism is divided into two parts Attention1 and Attention2. Attention1 is A1 = $\tanh(W_{s1}H^T)$ and Attention2 is A2 = W_{s2}A1. Final attention output is A=softmax(A2)
10. Finally, putting all of it together:

```
Training:
Input                  : bsz pairs of sentences
BiLSTM Input           : (bsz, len(sent[i]), WE_DIM)
BiLSTM Ouput           : (bsz, len(sent[i]), 2*LSTM_HIDDEN)
```

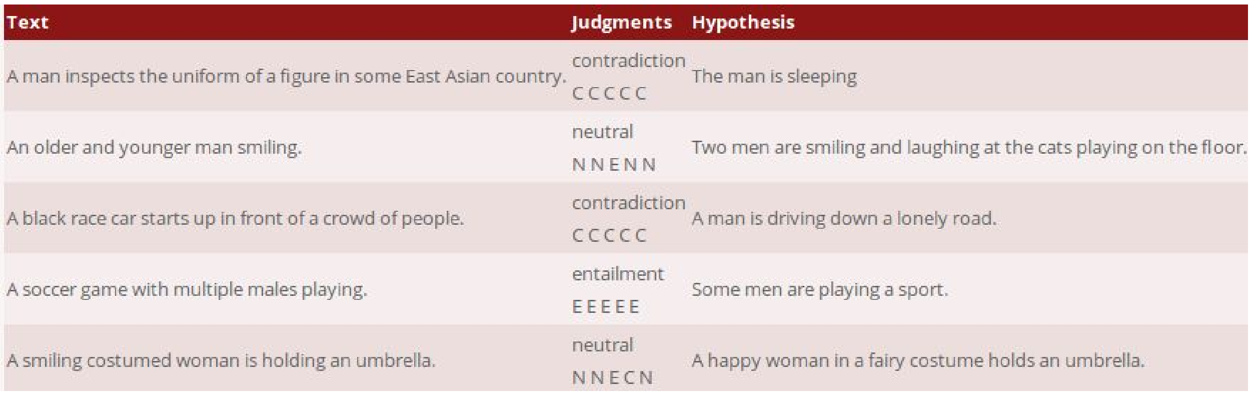
```
Attention1 Input       : (bsz, len(sent[i]), 2*LSTM_HIDDEN)
Attention1 Output      : (bsz, len(sent[i]), ATTENTION_HIDDEN)
Attention2 Input       : (bsz, len(sent[i]), ATTENTION_HIDDEN)
Attention2 Output      : (bsz, len(sent[i]), N_ROWS)
Sentence embeddings(M) : (bsz, N_ROWS, 2*LSTM_HIDDEN)
Gated Encoder Input    : [(bsz, N_ROWS, 2*LSTM_HIDDEN),
                           (bsz,N_ROWS, 2*LSTM_HIDDEN)]
Gated Encoder Output   : (bsz, N_ROWS, 2*LSTM_HIDDEN)
MLP Input              : (bsz, N_ROWS, 2*LSTM_HIDDEN)
MLP Output              : (bsz, 3)
Softmax layer           : (bsz, 1)

where
bsz                => Batch Size
LSTM_HIDDEN        => Number of hidden units in an LSTM
ATTENTION_HIDDEN  => Number of hidden units in attention
model, i.e. size of each output in A1.
N_ROWS             =>  Number of rows in the matrix M(M=AH)
```

E. Challenges faced:
There are 2 datasets used in this project till now, namely:

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), also known as recognizing textual entailment (RTE). 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.



Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

As the SNLI dataset is very large and this approach is known to converge after 4 epochs as per the research paper "Structured Self-Attentive Sentence Embedding", the time taken to run it was considerably large (easily more than 6 hours).

To execute it requires:

1. First run python2 tokenization.py. It generates

```
python train.py 300 150 4000 30 0.01 0.1 0.5 300 50 100 12 0.1:
```

So we had to run on ADA server, but still due to the restricted time limit provided to each user we were only able to run 1 epoch per login to ADA server. Therefore, it took long time to run train the model and test it (on the whole to run the code).

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.

   Here the dataset is relatively smaller so it was easy to train. But at the same time due to smaller training data, the model is not trained properly to classify correctly as *entailment*, *contradiction*, or *neutral*. Thereby, gave lower accurate results.

F. Accuracies on SNLI and SICK datasets:

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. Hence, we randomly shuffled the whole dataset and took one half of it and trained on only that half. In first epoch, we got an accuracy of approximately 77.2% on test dataset, 77% on validation dataset and 70.6% on train dataset.
   d. After second epoch, the accuracies are as following:

Train	: 78%
Validation	: 80.2%
Test	: 79.5%

   e. The accuracies after third epoch didn’t deviate much from second epoch, hence we are reporting the values after second epoch itself.
   f. With full dataset, the accuracies after second epoch are as follows:

Train	: 80%
Validation	: 83%
Test	: 82.1%

   g. We doubt that the error didn’t converge after second epoch, and we are running third epoch currently.
   h. 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.
   i. 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 training SNLI dataset

G. Instructions to run the code

a. SNLI

i. First run *python2 ./SNLI/SNLISNLI_train&test/tokenization.py*.

ii. Next run *python2 ./SNLI/SNLISNLI_train&test/train.py <LSTM_HIDDEN> <ATTENTION_HIDDEN> <MLP_HIDDEN> <N_ROWS> <LEARNING_RATE> <ATTENTION_PENALTY> <DROPOUT> <WE_DIM> <bsz> <GRADIENT_CLIP> <epochs> <STD>*

iii. Th command we ran is
python2 ./SNLI/SNLISNLI_train&test/tokenization.py && python2 ./SNLI/SNLISNLI_train&test/train.py 150 350 3000 10 0.01 1 0.5 300 50 100 3 0.1

iv. **LSTM_HIDDEN** : Number of hidden units in single LSTM
ATTENTION_HIDDEN : Number of hidden units in attention model, i.e. size of each output in $A1=\tanh(W_{s1}H^T)$.
N_ROWS : Number of rows in the matrix M(M=AH)
LEARNING_RATE : Learning rate eta
ATTENTION_PENALTY : Penalty used to avoid giving attention to same word again and again
DROPOUT : Dropout value used in MLP layer(Here, Dropout is used in only MLP layer. One can also use it in LSTM and Attention layers)
<WE_DIM> : Dimension of word embeddings
<bsz> : Batch size
<GRADIENT_CLIP> : Gradient is clipped after it reaches certain value to avoid exploding gradient problem
<epochs> : Number of epochs we intend to run
<STD> : Standard deviation of weights in initialization

b. **SICK**
   i. First run *python2 ./SICK/tokenization_SICK.py*.

   ii. Next run *python2 ./SICK/train_SICK.py <LSTM_HIDDEN> <ATTENTION_HIDDEN> <MLP_HIDDEN> <N_ROWS> <LEARNING_RATE> <ATTENTION_PENALTY> <DROPOUT> <WE_DIM> <bsz> <GRADIENT_CLIP> <epochs> <STD>*

   iii. Th command we ran is **python2 ./SICK/tokenization_SICK.py && python2 ./SICK/train_SICK.py 150 350 3000 10 0.01 1 0.5 300 50 100 3 0.1**

c. **SNLI only test:**
   i. Since training SNLI takes a lot of time, we stored parameters/weights for all neural networks. So, we can directly do testing to get accuracies in less time.
   ii. To only_test the model on SNLI dataset, run **python2 ./SNLI/SNLI_onlytest/to_test.py 150 350 3000 10 0.01 1 0.5 300 50 100 3 0.1** .
   iii. Accuracies can be obtained in less than 5 minutes.

H. Some future improvisations/optimizations(Final deliverable):

A few other optimizations can be made to the above algorithm to improve its accuracy:

1. Adding inter-sentence level attention
2. Augmenting word representations with character level features. This method improves the word embeddings efficiency and hence, sentence embeddings can be closer to the real meaning of the sentence.
3. Using Dynamic Coattention Network(DCN) approach. Normal deep learning models have no way to recover from local maxima corresponding to incorrect answers. To address this problem, DCN was proposed. DCN was actually proposed to solve question-answering problems. We have to check if there is a way to adapt this model into textual entailment problem.

Since Baseline is implemented, we will check which of the above models can be effectively adapted to textual entailment context. We will implement the selected model and deliver it as the final deliverable.