Project Report
Advanced Machine Learning

Topic:Sentence Compression by deletion using LSTMs

Aditya Golatkar-14B030009
Rudrajit Das-140020012
Abhishek Gore-140040011

Goal of our project:
We aim to perform sentence compression by deletion using LSTMs in which a sentence is translated into a sequence of zeros and ones, corresponding to deleted or retained words. Our baseline is a sentence compression paper by Google NLP Research which uses an encoder-decoder kind of model for this task and obtain pretty decent results. We propose a simple yet powerful 3 layer LSTM based method (no encoding-decoding) with a greedy search approach which obtains better results than the baseline.

References:

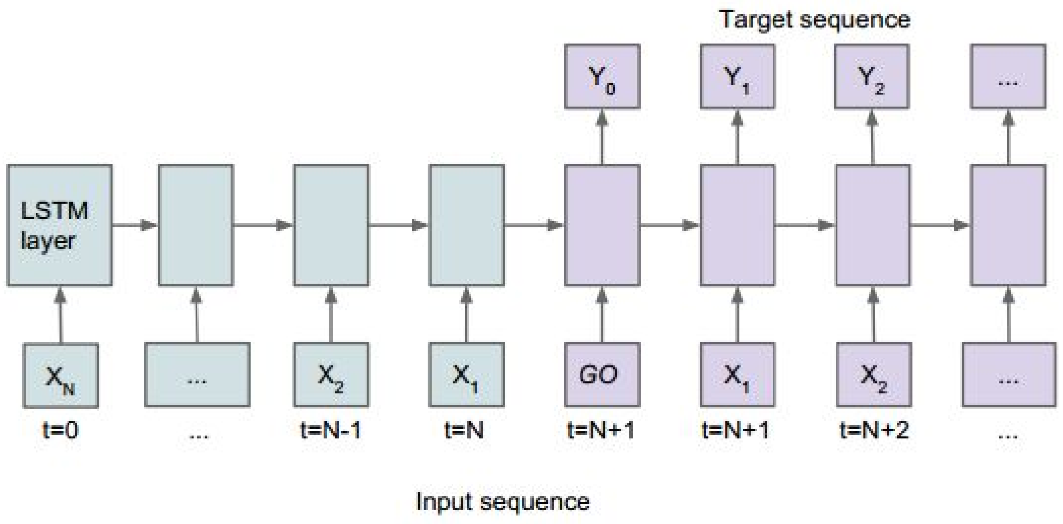
1) Sentence compression by deletion with LSTMs - Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser, Oriol Vinyals
2) Sequence to Sequence Learning with Neural Networks - Ilya Sutskever, Oriol Vinyals , Quoc V. Le
3) Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation
4) LONG SHORT-TERM MEMORY - Sepp Hochreiter et al.
5) Lecture Notes.

Description of Approaches:

The problem formulation we adopt in this paper is very simple: for every token in the input sentence we ask whether it should be kept or dropped, which translates into a sequence labeling problem with just two labels: one and zero. The deletion approach is a standard one in compression research, although the problem is often formulated over the syntactic structure and not the raw token sequence.

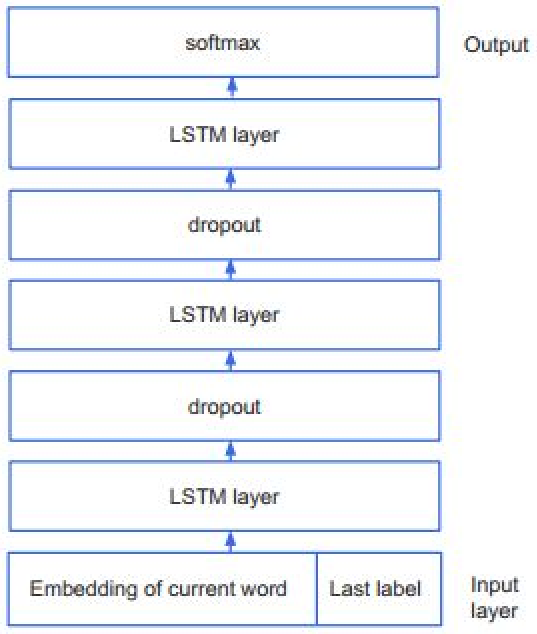
Baseline method :

We use the Encoder - Decoder approach proposed by our first reference as our baseline.



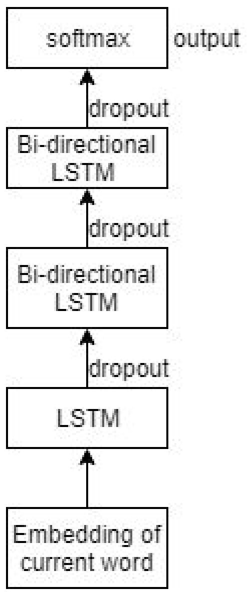
In the above figure the blue boxes represent the encoder LSTM and the purple boxes represent the decoder LSTM. The encoder and decoder both are 3 layer LSTM. N in the above figure represents the length of the sentence.
The input sentence is first fed to the encoder LSTM in the reverse order and then to the decoder in the normal order (Sutskever et al.). The decoder then generates binary tokens for each input word. Words in the sentences are embedded using Word2vec model. We have used three Word2vec models for word embedding: 1)trained on google news dataset, 2)trained on the training corpus and 3)trained on google news dataset and fine-tuned using the training corpus.

At the decoder we have used a greedy approach to label the words in the sentence, i.e. the input to the first LSTM is just the embedding vector for the current word without the label of the previous word appended to it.



This figure represents LSTM block used in the encoder and the decoder of the above model. However in our implementation we do not append the last label to the input. Also inspired from the GNMT paper we have used BiLSTM in the first layer of the encoder.

Our pure LSTM approach-



The above figure shows the schematic for our pure LSTM based approach. As mentioned before, we followed a greedy search approach. We have used a 3 layer LSTM as suggested by

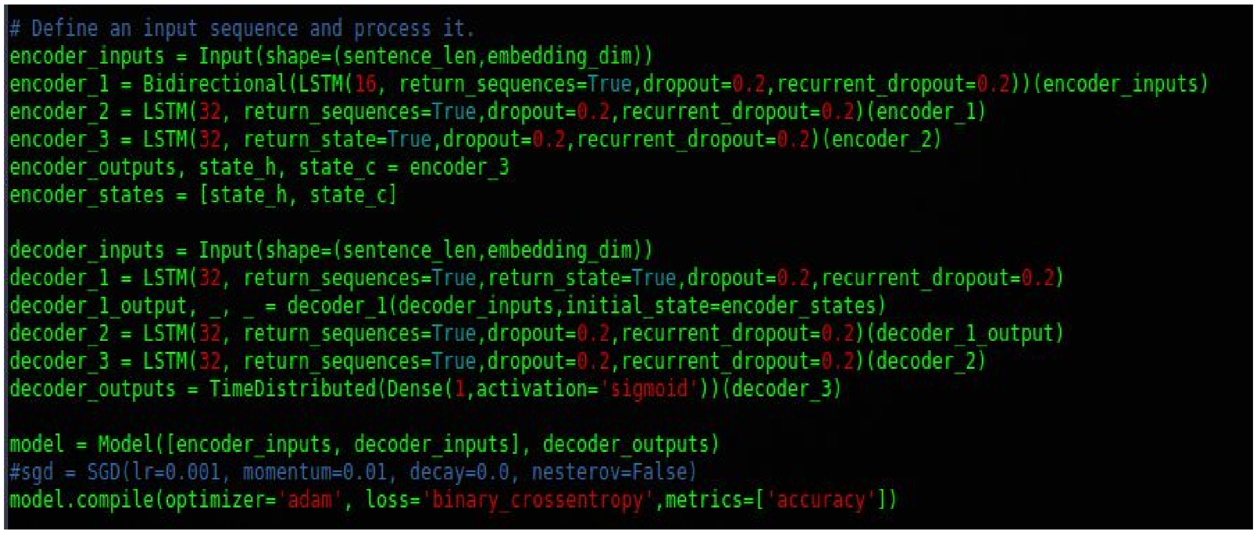
Sutskever et al. The bottom LSTM is unidirectional whereas the two on top of it are bidirectional LSTMs. The reason for not using bidirectional LSTM at the lowest layer is that during testing there would be ambiguity of which word corresponds to the output that is propagated forward at the k^{th} time instant - the k^{th} word or the $(n-k)^{th}$ word where n is the total number of words in the sentence. We have added dropout with probability 0.2 in between the LSTMs. The final bidirectional LSTM is connected to a feed forward neural network with a sigmoid output layer.

Experiments:

1.

The 3 main codes for our project are -

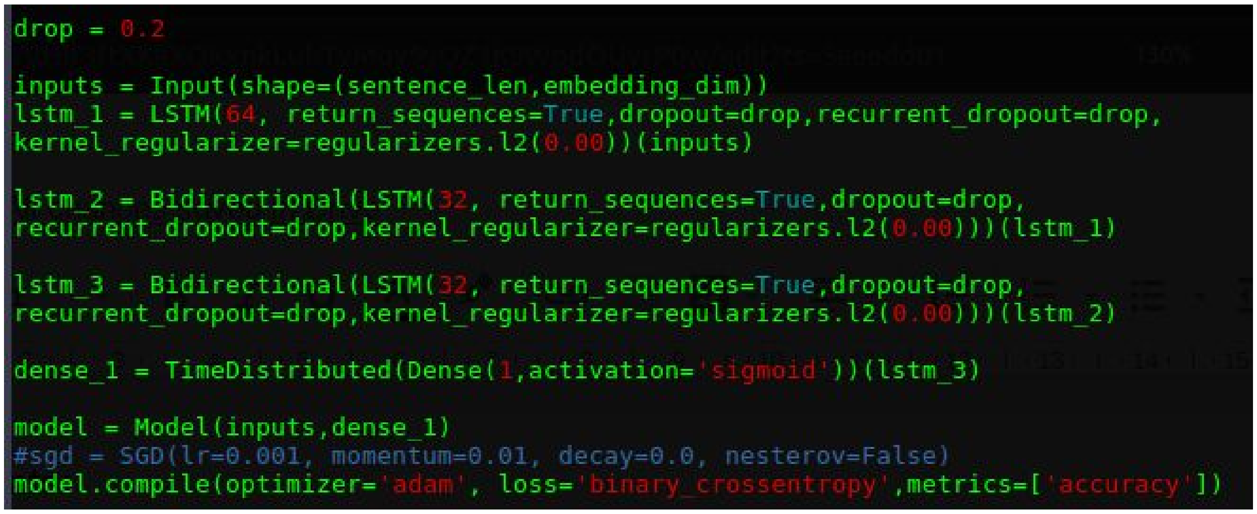
Code for baseline model:



```
# Define an input sequence and process it.
encoder_inputs = Input(shape=(sentence_len,embedding_dim))
encoder_1 = Bidirectional(LSTM(16, return_sequences=True,dropout=0.2,recurrent_dropout=0.2))(encoder_inputs)
encoder_2 = LSTM(32, return_sequences=True,dropout=0.2,recurrent_dropout=0.2)(encoder_1)
encoder_3 = LSTM(32, return_state=True,dropout=0.2,recurrent_dropout=0.2)(encoder_2)
encoder_outputs, state_h, state_c = encoder_3
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(sentence_len,embedding_dim))
decoder_1 = LSTM(32, return_sequences=True,return_state=True,dropout=0.2,recurrent_dropout=0.2)
decoder_1_output, _, _ = decoder_1(decoder_inputs,initial_state=encoder_states)
decoder_2 = LSTM(32, return_sequences=True,dropout=0.2,recurrent_dropout=0.2)(decoder_1_output)
decoder_3 = LSTM(32, return_sequences=True,dropout=0.2,recurrent_dropout=0.2)(decoder_2)
decoder_outputs = TimeDistributed(Dense(1,activation='sigmoid'))(decoder_3)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
#sgd = SGD(lr=0.001, momentum=0.01, decay=0.0, nesterov=False)
model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

The code is divided into two blocks. The first block is the encoder and the second block is the decoder. In the above code state_h and state_c are the hidden representations obtained from the encoder. They are used as initial states of the decoder. We use a three stage encoder and decoder. Final a sigmoid layer to obtain the probability of each word to be included in the summary.

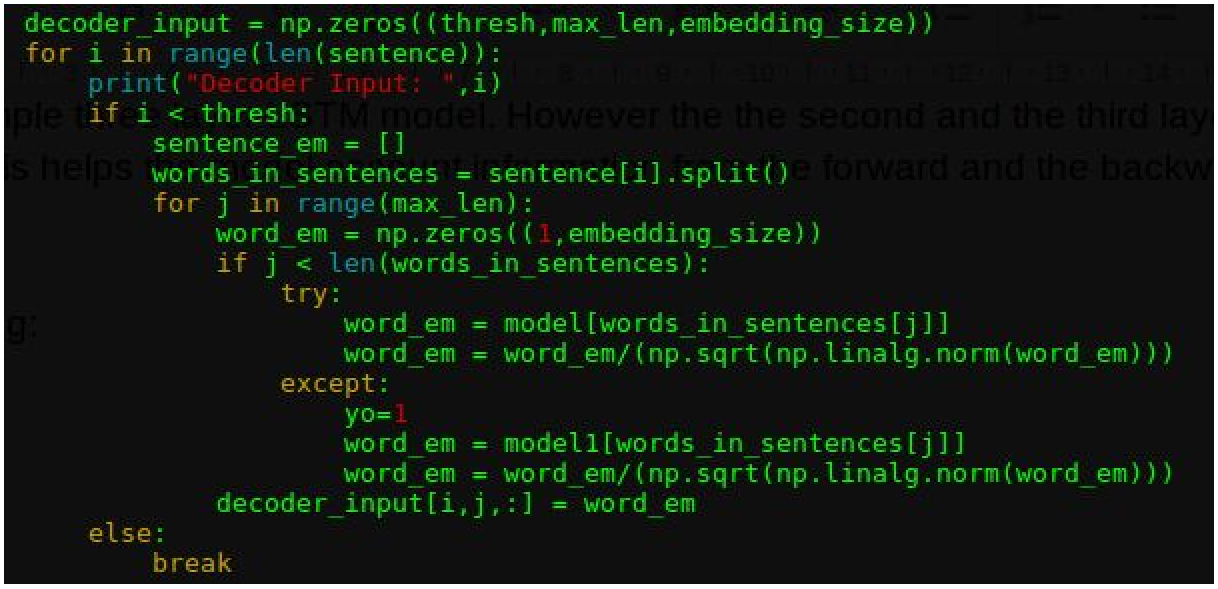
Code for our pure lstm model:



```
drop = 0.2

inputs = Input(shape=(sentence_len,embedding_dim))
lstm_1 = LSTM(64, return_sequences=True,dropout=drop,recurrent_dropout=drop,
kernel_regularizer=regularizers.l2(0.00))(inputs)

lstm_2 = Bidirectional(LSTM(32, return_sequences=True,dropout=drop,
recurrent_dropout=drop,kernel_regularizer=regularizers.l2(0.00)))(lstm_1)

lstm_3 = Bidirectional(LSTM(32, return_sequences=True,dropout=drop,
recurrent_dropout=drop,kernel_regularizer=regularizers.l2(0.00)))(lstm_2)

dense_1 = TimeDistributed(Dense(1,activation='sigmoid'))(lstm_3)

model = Model(inputs,dense_1)
#sgd = SGD(lr=0.001, momentum=0.01, decay=0.0, nesterov=False)
model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

We use a simple three layer LSTM model. However the the second and the third layer are BiLSTMs. This helps the model account information from the forward and the backward direction.

Preprocessing:



```
decoder_input = np.zeros((thresh,max_len,embedding_size))
for i in range(len(sentence)):
    print("Decoder Input: ",i)
    if i < thresh:
        sentence_em = []
        words_in_sentences = sentence[i].split()
        for j in range(max_len):
            word_em = np.zeros((1,embedding_size))
            if j < len(words_in_sentences):
                try:
                    word_em = model[words_in_sentences[j]]
                    word_em = word_em/(np.sqrt(np.linalg.norm(word_em)))
                except:
                    yo=1
                    word_em = model1[words_in_sentences[j]]
                    word_em = word_em/(np.sqrt(np.linalg.norm(word_em)))
            decoder_input[i,j,:] = word_em
    else:
        break
```

This code is a part of the preprocessing step. In this code we take words from the sentences and generate embedding for those words using Word2Vec.

4.

Description of code files:

Preprocessing:

1) Json_to_numpy : In this code we converted the data from json format and stored it in numpy format. (115 lines)
2) Binary_output_generator : In this code we generate binary outputs using the actual sentences and the summary sentences. (101 lines)
3) Refine_data: In this code we refine the data i.e. remove the sentence pairs which are very large as compared the mean sentence length of the data we have. (146 lines)
4) Embedded_data_gen : In this code we generate data embedding from the word2vec model in the format required by the LSTM model (153 lines)
5) Prepare_for_word2vec_train: In this code we train the word2vec model using our data corpus.(44 lines)

Models:

1) Encoder_decoder : In this code we have the encoder decoder model. This code trains the model and also generate the predictions for the test data. (118 lines)
2) Pure_lstm: In this code we have our proposed pure lstm model. This code trains the model and also generates the predictions for the test data. (118 lines)
3) Observe_results : This code generates the average F1 score over the test data and also decodes the predictions given by the LSTM to generate our predicted sentences. (59 lines)

2.
Language - Keras
Environment - Tensorflow

3.
References for code-
The preprocessing and the pure lstm code has been completely written by us. For the encoder decoder we taken some help from here :
https://github.com/keras-team/keras/blob/master/examples/lstm_seq2seq.py
This is basically the examples give on the Keras github page.

In addition we had to do a lot of pre-processing :

4.
Platform - Linux/Windows

Machine - First we ran the code on our laptops but we faced some external issues due to which we had to take permission to use the EE department VIP Lab Machines. Time for training ~ 5 hours

Experimental Results:

Dataset used:

We have used the google research dataset available at this link:
https://github.com/google-research-datasets/sentence-compression/tree/master/data

This dataset was also used by our baseline method.

Encoder-Decoder model -

Training time is a bit longer for this model because we have an encoder-decoder kind of model.

Embedding type	**Exp No.**	**Sentences size**	**Inner Representation of LSTM**	**Dropout**	**Train samples**	**Test samples**	**Average F1 Score**	**Time per epoch (sec)**	**No, of Epochs**	**No. of Parameters**
Word2vec	1	10-15	32	0.2	15000	500	**83.96**	180	50	~300000
Word2vec	2	10-15	64	0.2	15000	500	**84.86**	180	50	~310000
Trained on the corpus	3	10-15	64	0.2	15000	500	**84.09**	180	50	~310000
Word2vec fine tuned on trained on the corpus	4	10-15	64	0.2	15000	500	**84.59**	180	50	~310000
Word2vec fine tuned on trained on the corpus	5	20-25	64	0.2	20000	1000	**78.9**	190	50	~310000

Sample sentences+gold summaries+generated summaries -

1.

Actual sentence:
The Phillies want to keep Chase Utley but other teams are interested in acquiring him

Actual Summary:
The Phillies want to keep Chase Utley

Our model's Summary :
The Phillies want to keep Chase Utley

Comments: If you notice in this example the summary is essentially the first six sentences of the sample sentence. So the encoder decoder model is able to neglect the last few unimportant words.

2.
Actual sentence:
After weeks of rumors and expectations Google has officially introduced Android 4 3 Jelly Bean

Actual Summary:
Google has officially introduced Android 4 3 Jelly Bean

Our model's Summary :
Google has officially introduced Android 4 3 Jelly Bean

Comments: In this example the summary is the last few words of the actual sentence. The encoder decoder model has also learnt to neglect the first few unimportant words. This example is the exact opposite of the first example.

3.
Actual sentence:
Pita restaurant chain Pita Pit has closed in Edwardsville

Actual Summary:
Pita Pit has closed in Edwardsville

Our model's Summary :
Pita Pit has closed in Edwardsville

Comments : In this sentence the summary consists of words picked from the start, middle and the end of the actual sentence.

4.
Actual Sentence :
Mayors from cities across Kansas urge members of Congress to pass comprehensive immigration reform

Actual Summary :
Mayors urge members of Congress to pass immigration reform

Our model's Summary :
Mayors urge members of Congress to pass immigration reform

Pure LSTM model -

Embedding	**Exp No.**	**Sentences size**	**Inner Representation**	**Dropout**	**Train samples**	**Test samples**	**Average F1 Score**	**Time per epoch(sec)**	**No. of Epochs**	**No. of Parameters**
Word2vec	1	10-15	16	0.2	15000	500	**84.64**	150	50	~140000
Word2vec	2	10-15	32	0.2	15000	500	**85.12**	150	50	~143000
Trained on the corpus	3	10-15	32	0.2	15000	500	**84.77**	150	50	~143000
Word2vec fine tuned on trained on the corpus	4	10-15	32	0.2	15000	500	**85.5**	150	50	~143000
Word2vec fine tuned on trained on the corpus	5	20-25	32	0.2	20000	1000	**80**	160	50	~143000

Sample sentences+gold summaries+generated summaries -

All the sentence which we showed above were also correctly decoded using our LSTM model.

1.

Actual sentence:
Mayors from cities across Kansas urge members of Congress to pass comprehensive immigration reform

Actual Summary:
Mayors urge members of Congress to pass immigration reform

Our model's Summary :
Mayors urge members of Congress to pass immigration reform

Comments:
Our simple yet powerful LSTM model is able to correctly summarize the sentence with words coming from the start, mid and end.

2.

Actual sentence:
The UPA chairperson alleged that the BJP only makes promises while the Congress delivers

Actual Summary:
The BJP makes promises while the Congress delivers

Our model's Summary :
The BJP makes promises while the Congress delivers

3.

Actual Sentence :
George Michael has topped the UK album charts with his new album Symphonica

Actual Summary :
George Michael has topped the UK album charts

Our model's summary:
George Michael has topped the UK album charts

4.
Actual Sentence :
Alyssa Milano revealed Friday that she and husband David Bugliari are expecting their second child

Actual Summary :
Alyssa Milano and David Bugliari are expecting their second child

Our model’s Summary :
Alyssa Milano and David Bugliari are expecting their second child

Examples where our model performed better than the encoder decoder model:

1.
Actual Sentence :
It s tax day the deadline to file your federal taxes

Actual Summary :
It s tax day

Encoder decoder summary:
S tax day the deadline taxes

Pure LSTM summary:
It s tax day

2.
Actual Sentence :
Big Cinemas a division of Reliance MediaWorks has entered washroom advertising

Actual Summary :
Big Cinemas has entered washroom advertising

Encoder decoder summary:
has entered washroom advertising

Pure LSTM summary:
Big Cinemas has entered washroom advertising

3.
Actual Sentence :

DeLaet tied second at The Barclays a new career high for the former Weyburn resident

Actual Summary :
DeLaet tied second at The Barclays a new career high

Encoder decoder summary:
DeLaet tied The Barclays a new career high

Pure LSTM summary:
DeLaet tied second at The Barclays a new career high

Effort-

Fraction of time spent on different parts-

1.Pre-processing the data - 40%

2.Word embeddings - 10%

3.Training the Encoder-Decoder model - 30%

4. Training our LSTM model - 20%

Most challenging part -

1) In the dataset, we had to exclude the large sentences as we had to make all the sentences of the same length. If the largest sentence is too large, then large number of zeros would be appended to majority of the sentences.
2) Since our task is binary classification, to train the model efficiently the data set needs to be balanced. However in our case the the number of zeros will be larger than the number of ones. As a result the model will get biased to predict zeros. To overcome this problem we choose sentences such that the number of zeros and the ones were close by so that we get a well balanced dataset.

Future Work:

1) We plan to use our model along with initial hidden state representation inspired from the work : A simple but tough to beat baseline for sentence embedding - Arora et al.
2) We also plan to use our model along with an attention based approach with attention inspired from : A simple but tough to beat baseline for sentence embedding - Arora et al.

Fraction of work done by team members-

Aditya : Encoder-decoder model + Pure LSTM + Pre-processing
Rudrajit : Encoder-decoder model + Pure LSTM + Pre-processing
Abhishek : Pre-processing + Encoder-decoder model