Gautami Gupta
(IMT2016069)
Project Elective Report

Problem Statement : Extractive and Abstractive Text Summarisation

Abstract : Nowadays, there is a lot of information available on the Internet which requires mechanisms which can extract relevant information efficiently. Automatic text summarisation is the solution for the above problem. Text summarisation is a process of identifying the most important information in a transcript and compressing them to a shorter transcript preserving its meaning.[1] I have implemented both Extractive and Abstractive text summarisation. Extractive summarisation is the technique of identifying important sentences within the transcript. It contains same sequence of words as in the original transcript. On the other hand, abstractive text summarisation understands the meaning of the transcript and displays the summary in natural language. It may or may not contains sentences from the transcript, giving a relevant summary. I have focused on indicative summaries like generating headlines for a transcript which has a summary length of 15-20 words.

Algorithms/ techniques implemented:

- TextRank for Extractive text summarisation
- Encoder-Decoder model with attention mechanism for Abstractive text summarisation.

TextRank for Extractive text summarisation :

I have implemented TextRank algorithm for extracting significant key phrases from the transcript. The basic idea of the TextRank algorithm is assigning weights to every word(except the stop_words) and then building a weighted undirected graph of words followed by grouping words which are connected to form key phrases. The importance of the key phrases is decided by its total weight(calculated by the summation of weights of each word in a phrase). It is an unsupervised algorithm and does not uses any deep learning approach

Dataset :Videoken dataset

Libraries used : nltk, numpy, pandas

Input : “We've learned about methods for regression and for classification involving predictors and for making predictions from our data. Well, ideally, we'd like to get a new sample from the population and see how well our predictions do. Well, we don't always have new data. And we can't use our training data just straight off, because it's going to be a little bit optimistic. So we're going to tell you about cross-validation which is very clever device for using the same training data to tell you how well your prediction method works.”

---

[1] "(PDF) Text Summarization:An Overview - ResearchGate." https://www.researchgate.net/publication/257947528_Text_SummarizationAn_Overview.

Data Preprocessing :

- Tokenize each word using word_tokenize() (a function in nltk library)
- Add part-of-speech tagging to each word. (list of pos_tag)
- Lemmatize nouns and adjectives in the transcript. Here different grammatical forms change to single one. E.g. “methods” change to “method”
- Removing stop words: Any word which is not a noun, adjective, verb or a foreign word is considered as a stop word. I am assuming here that keywords are generally nouns, adjective or foreign words.

Algorithm implementation :

Building a graph and assigning weights to each vertex:
After removing stop words and repeated words, we get a list of words which can be used to build the word graph. Each word serves as vertex. We represent the graph(weighted_graph) in the form an adjacency matrix, which contains details about the connected vertices and their weights.
Weighted_graph[i][j] contains the weight of the edge from vertex i to vertex j. It is zero if there is no edge between the two vertices.
There is a semantic connection between words of they fall within the same window. I have defined window_size as 3. Thus, words in the same window will be have a connected edge between them.
Now, I will slide the window of size three across the words. If words(i,j) fall in the same window, the value of weighted_graph[i][j] will be 1/math.fabs(index_of_i-index_of_j)
This is how a weighted undirected graph is build for TextRank.

Calculate importance of each word :
Next, we calculate the weighted connection of each vertex i, which is the summation of weights of each edge connected to vertex i. Stored in inout[i]
To calculate the importance of each word, we calculate the score of each vertex. The formula used for scoring a vertex represented by i is:
score[i] = (1-d) + d x [Summation(j) ((weighted_edge[i][j]/inout[j]) x score[j])] where j belongs to the list of vertices that has a connection with i. d is the damping factor. I have taken the value of damping factor as 0.85.

Build and scoring phrases:
I will form phrases from the lemmatized text using stop words as delimiters. Phrases will be formed if there are no stop words between the two words.
The score of each phrase will be the summation of the score of each word in that phrase. Thus, we get a list of key phrases together with its score of importance.

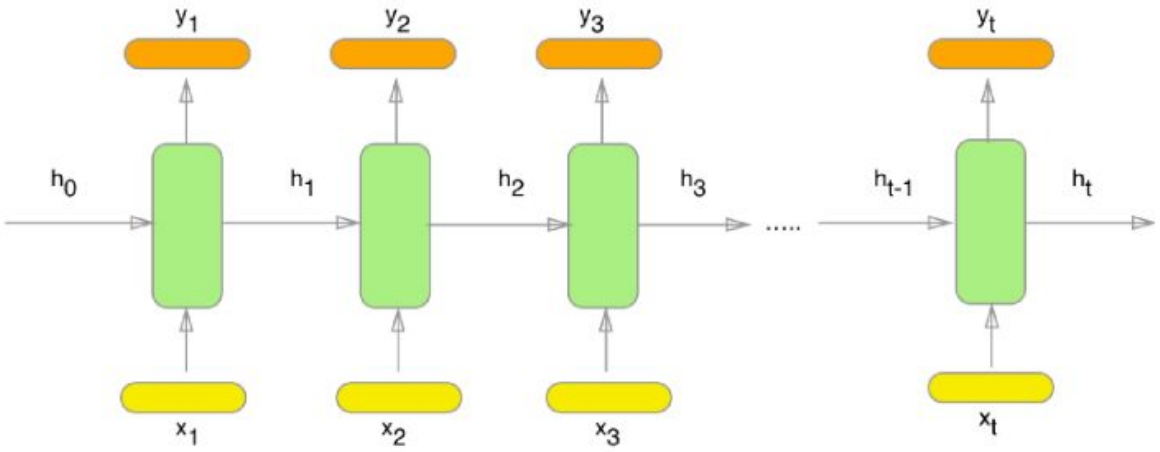
Output :
prediction method work (score = 2.76)
classification involving predictor (score = 2.59)
making prediction (score = 2.59)

Encoder-Decoder Sequence to Sequence model with Attention mechanism for Abstractive text summarisation :

What is Recurrent Neural Network(RNN) :

RNNs are algorithms used for sequential data owing to their feature of internal memory. In sequence data, every state is dependent upon the previous state and therefore, it is important to consider the previous state when predicting the next state.

RNN algorithms work in loops. When predicting the output, it takes in both the current input and the features it learned from the previous inputs.



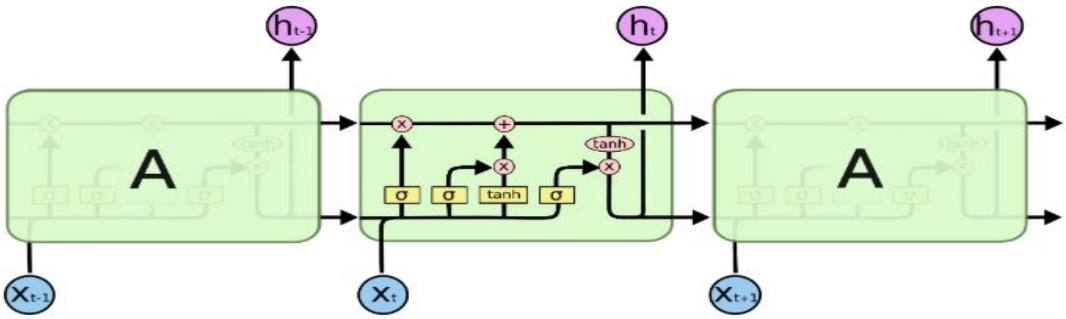
Here, x_i is the input words, h_i is the features form the previous input words and y_i is the output word.

Another feature of RNN which makes it suitable for seq-to-seq input is the Backward-propagation through time. It means going backward through the network and finding the derivative of errors wrt to weights and then adjusting the weights to minimize the error.

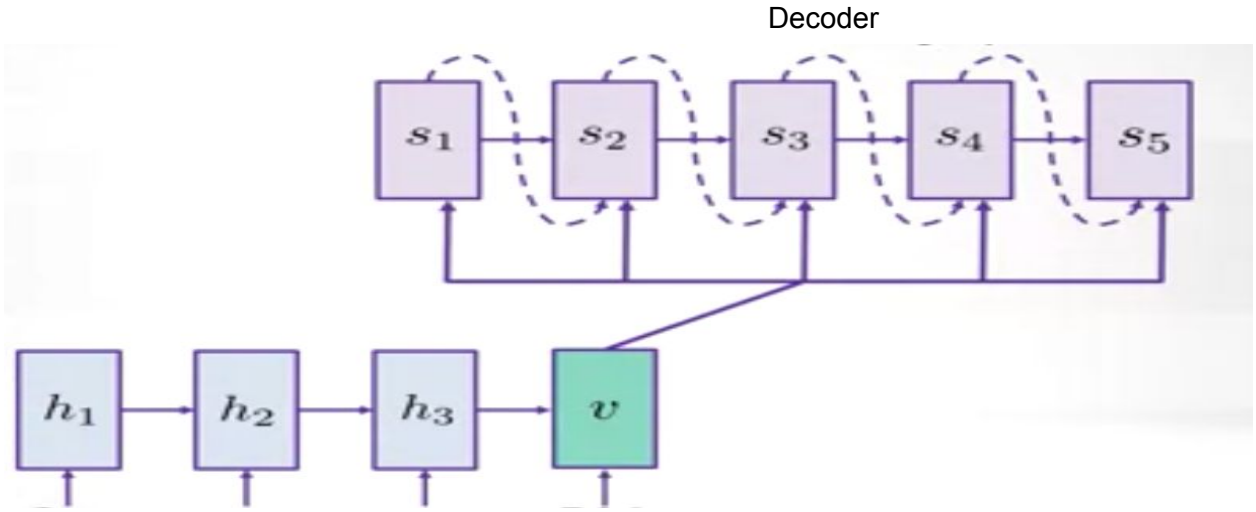
Thus, RNNs with internal memory and backward-propagation technique is the best algorithm for sequence data and gives best results.

What are Long short-term memory(LSTM) networks :

LSTMs are a special kind of RNN, capable of learning long-term dependencies. The learning module has four instead of one neural network layer. It has forget gate layer, input layer and output layer.The forget gate layer decides what information to throw away. The input gate layer decides what information we are going to store in cell state. The output gate decides what output should be sent from the cell.

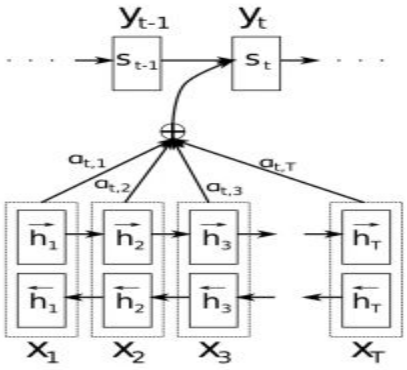


What are Encoder to Decoder Sequence to Sequence Model :
It uses two Recurrent Neural Networks. One encodes the input sequence of symbols to vector representation and the other decodes the vectors to another sequence of symbols.



Here, we provide input as words to the encoder which learn features and passes it to the next state together with the new input. The cell 'V' called as the Thought Vector, stores the vector representation of the learned features from the entire input sequence. V is passed on to the decoder. We pass it to all the cells of the decoder for a better prediction. The decoder takes in the vector representation and convert it back to the sequence of words.

Attention Mechanism :
It seems somewhat unreasonable to assume that we can encode all information about a potentially very long transcript into a single vector i.e. V(thought vector) and then have the decoder produce a good translation based on only that. This does not gives quite satisfactory results. This problem can be solved using the attention mechanism.
With an attention mechanism, we do not encode the entire input transcript into a single vector. Rather, we allow the decoder to extract features from different parts of the input transcript at each step of the output generation.



Implementation : Supervised learning using Bidirectional encoder and Basic decoder on LSTM cells; Bahdanau Attention with weight normalization.

Dataset : CNN News Dataset

Libraries used: Tensorflow, gensim, nltk, numpy

Data preprocessing :

- Tokenizing the words in a transcript using word_tokenize().
- Build dictionary with keys as the word and item as its index.
- Here, we do need to remove the stop words as in Extractive text summarisation.
- The most important step here is to initialize word-embeddings. I have used 'GloVe(Global Vectors)' for that purpose. It is a pre-trained algorithm for converting words to their semantic vector representation. The model is trained on the co-occurrence matrix of words. Glove adds some more meaning to word vectors by considering the relationships between word pair and word pair rather than word and word. Thus proving better than word2vec. Also, it gives low weight to frequently occurring words thus prevents stop words like, a, the, is etc. from dominating the training.
- I have set the maximum transcript size to 150 and maximum summarization size to 30.

Training model :

I will be training the model on news dataset of size 2,00,000. I am using Tensorflow library. Using tensorflow session as a context manager.

- Defining tensorflow parameters :
    - embedding size = 300
    - learning rate = 0.001
    - epochs = 5
    - num_layers =2
    - num_hidden = 150
- Defining LSTM cell using `tf.nn.rnn_cell.BasicLSTMCell`
- Encoder :
    - Encoder has a Input layer which takes the english transcript and passes it to the embedding layer.
    - Embedding layer converts each word to a fixed sized vector.

    - Creating a dynamic bidirectional recurrent neural network with LSTM cell using tf.contrib.rnn.stack_bidirectional_dynamic_rnn(fw_cells, bw_cells, encoder_embeddings, sequence_length) where fw_cells are LSTM cells for forward propagation and bw_cells are LSTM cells for backward propagation.
- Attention Mechanism : Apply Bahdanau Attention using tf.contrib.seq2seq.BahdanauAttention
- Decoder :
    - After the attention mechanism, build a decoder cell, using tf.contrib.seq2seq.AttentionWrapper with the implemented attention mechanism.
    - Projection/Dense layer : Outputs the vector representing the target.

- Using cross entropy for loss calculation. I have also applied AdamOptimizer which further improves the prediction.
- Also, I have saved the model checkpoints here

Results :
Loss calculated by cross entropy is 6.6 to 8.9 for different batches.

Input1 : five-time world champion michelle kwan withdrew from the us figure skating championships on wednesday , but will petition us skating officials for the chance to compete at the turin olympics .
Output1: world figure skating cycling results.

Input2 : several thousand people gathered on wednesday evening on the main square in zagreb for a public draw and an open air party to celebrate the croatian capital 's second chance to host the women 's slalom world cup .
Output2 : croatia 's world cup super-g results

Input3 : us business leaders lashed out wednesday at legislation that would penalize companies for employing illegal immigrants .
Output3 : us business leaders lash out at illegal immigrants

Future Improvements :

- The videoken dataset was had a lot of junk data due to which, the summaries generated were not relevant. e.g.
  Input : We've learned about methods for regression and for classification involving predictors and for making predictions from our data. Well, ideally, we'd like to get a new sample from the population and see how well our predictions do. Well, we don't always have new data. And we can't use our training data just straight off, because it's going to be a little bit optimistic. So we're going to tell you about cross-validation which is very clever device for using the same training data to tell you how well your prediction method works.
  Output : beijingers distance out of colleagues after being used to inspire
  Thus, I think it is necessary to clean data to get relevant summary for videos.

- I am currently studying about Deep transfer reinforcement learning. Text summarisation for small text datasets fail to give good results with above mechanism. Since, videoken dataset is small, I think we can achieve better results using reinforcement learning. Also, attentional, RNN-based encoder-decoder models for abstractive summarization have achieved good performance on short input and output sequences. For longer documents and summaries however these models often include repetitive and incoherent phrases. This problem can also be overcome by reinforcement learning.
- I studied about Bert model which is a pre-trained transformer model. We can use for Bert for extractive text summarisation and table of content generation. Bert model has not given significant results for abstractive text summarisation, therefore we decided to go with attentional, RNN-based encoder-decoder models.

Readings :

- "(PDF) Text Summarization:An Overview - ResearchGate." https://www.researchgate.net/publication/257947528_Text_SummarizationAn_Overview.
- https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/
- Generating News Headlines with Recurrent Neural Networks by Konstantin Lopyrev https://arxiv.org/pdf/1512.01712.pdf
- https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf
- https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjvnKGropngAhVIiHAKHR2EBVgQFjAAegQIAhAC&url=https%3A%2F%2Fwww.researchgate.net%2Fprofile%2FSamrat_Babar%2Fpublication%2F261174110_Improving_Text_Summarization_Using_Fuzzy_Logic%2Fdata%2F004635336b83780261000000%2FREPORT-II.pdf&usg=AOvVaw033aOMwzkkY-y34Hj8Dwo9
- https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f
- Neural Abstractive Text Summarization with Sequence-to-Sequence Models https://arxiv.org/pdf/1812.02303v2.pdf
- Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond https://arxiv.org/abs/1602.06023
- Abstractive Text Summarization with Attention-based Mechanism https://upcommons.upc.edu/bitstream/handle/2117/119051/131670.pdf
- https://paperswithcode.com/paper/deep-reinforcement-learning-for-sequence-to
- Generative Adversarial Network for Abstractive Text Summarization∗ https://arxiv.org/pdf/1711.09357.pdf
- Deep Transfer Reinforcement Learning for Text Summarization https://arxiv.org/abs/1810.06667