# Abstracting the summary from audio transcripts of videos

*Project Elective*

Atibhi Agrawal

*Under Prof. Manish Gupta*

*International Institute of Information Technology, Bengaluru*

Submitted 13 May 2019

## Abstract

All of us watch or have watched educational videos online. However, we cannot fully leverage the content. To help us to better leverage the content Videoken has come up with an innovative solution of generating table of contents. . This allows a textbook-like facility for non-linear search and navigation through the video, enables extraction of semantically coherent clips from within a video and improves video search through better semantic indexing. The platform also allows new ways of course creation and sharing of learning modules; and can be both integrated with existing Learning Management Systems and used independently. (Debabrata Mahapatra, 2018; et al) During this project elective we have tried to summarize the text in between the table of contents. The task of the text summarization is to condense long documents into short summaries while preserving the important information and meaning of the documents. Having the short summaries, the text content can be retrieved, processed and digested effectively and efficiently. Our work includes exploring both abstractive and extractive text summarization and the challenges that this task poses.

*Keywords:* Abstractive Text Summarization, Bleu Scores, Rouge Scores, LSTMs

## Introduction

Text Summarization is one of the most challenging and interesting problems in the field of Natural Language Processing.There are two main ways of summarizing text in NLP. Extractive text summarization and Abstractive text summarization. During this semester we have explored both the methods.

## Extractive Text Summarization

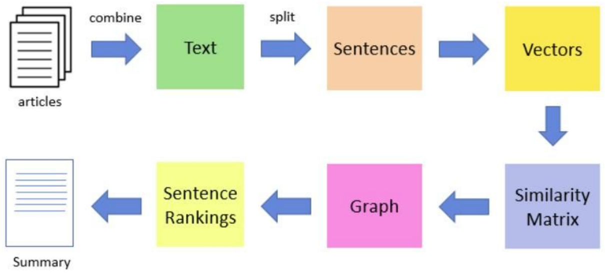
A method is considered to be extractive if words, phrases, and sentences in the summaries are selected from the source articles. They are relatively simple and can or cannot produce grammatically correct sentences. The generated summaries usually persist salient information of source articles and have a good matching with human-written summaries.

Here is an example :

**Source text:** ***Joseph and Mary*** *rode on a donkey to* ***attend*** *the annual* ***event*** *in* ***Jerusalem****. In the city,* ***Mary*** *gave* ***birth*** *to a child named* ***Jesus****.*

**Extractive summary:** *Joseph and Mary attend event Jerusalem. Mary birth Jesus.*

We have used textrank algorithm for implementing extractive text summarization. It is an unsupervised learning algorithm which is very similar to pagerank.



What textrank basically does is in the first step it basically concatenates all the text contained in the articles. Then splits the text into individual sentences. In the next step, we find vector representation (word embeddings) for each and every sentence. Similarities between sentence vectors are then calculated and stored in a matrix. The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation Finally, a certain number of top-ranked sentences form the final summary.

In our implementation, we have also performed pre-processing on the data before generating summaries. First converted the summary to lowercase and tokenized it. Then applied POS tagging to the same. POS tagging simply means labelling words with their appropriate part of speech. POS tagging is a supervised learning solution that uses features like the previous word, next word, is first letter capitalized etc. NLTK has a function to get pos tags and it works after tokenization process. Then we lemmatize the tokens. It reduces the inflectional forms of each word into a common base or root. Then we removed some of the stopwords and finally created a dictionary of unique words. Next, we created a weighted undirected graph and if weighted_edge[i][j] is zero, it means no edge or connection is present between the words represented by index i and j. We gave scores to each vertex depending on the number of connections it has. Let d be the damping factor and inout[i] contain the total no. of undirected connections\edges associated with the vertex represented by i. The score can be calculated as follows:

score[i] = (1-d) + d x [ Summation(j) ( (weighted_edge[i][j] / inout[j]) x score[j] ) ]

Then we score keyphrases. Scoring is done on the basis of score of each vertex calculated above. Here is the outcome of a summary taken from the Videoken dataset:

Below is the non-pre processed initial text :

```
Hello, in our lecture we looked at
the corporate social responsibility
and also the ethics, ethics and
social responsibility takes us to an
another important area of concern
that is the human resource
management or the personal
management. In this lecture, I
intend to talk about the various
areas of human resource management,
personal management and would like
to give you quickly an overview of
the history of human resource
management, the definitions and
concepts of personal management, the
basic functions of personal
management and the strategic role of
human resource management in the
organization and also the new trends
facing human resources and
particularly, the human resource
management departments and also we
will see what are the issues before
one has to think about the human
resource and people management.
```

The summaries generated after textrank are :

```
human resource management
department,
human resource management,
new trend facing human resource,
human resource,
personal management,
corporate social responsibility,
social responsibility,
important area,
various area,
```

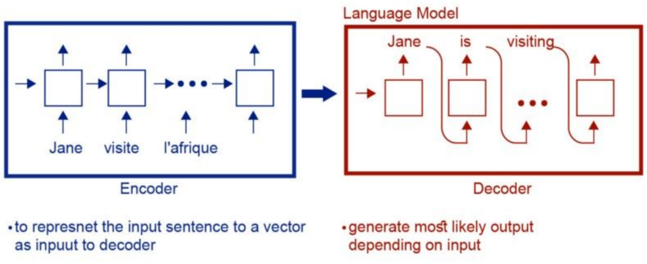
```
issue beforeone
```

Even though extractive text summarization is helpful, abstractive text summarization generates human like summaries with novel words using language generation models grounded on representations of source documents. Thus, they have a strong potential of producing high-quality summaries that are verbally innovative and can also easily incorporate external knowledge. Next we have focused on seq2seq models for text summarization.

## Abstractive Text Summarization

We have used a Recurrent Neural Network as the baseline model. We have done so because we cannot use a simple neural network because we need to share features across different parts of a text. RNN is the base of seq2seq.

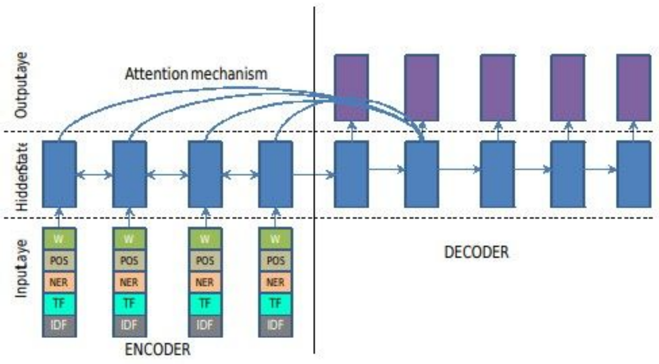
The simplest RNN has a many to many architecture with same lengths for input as well as output. For text summarization, we need a network that takes input of length (Tx) , and generates another output of another different length (Ty) , this architecture is called Encoder Decoder. Both Encoder Decoder are RNN networks.


Language Model
Jane
is
visiting
Jane
visite
l'afrique
Encoder
Decoder
•to represnet the input sentence to a vector as inpuut to decoder
•generate most likely output depending on input

We have made modifications to the core of the encoder-decoder meaning the RNNs to increase the efficiency. We have used beam search, attention and pointer generator. There are two main problems with the RNN, exploding gradients which occurs during back propagation when the gradients get too large and vanishing gradients which happen due to large number of layers and inability of the RNN to remember old values. This is very important in an NLP problem as some words depend on a word that appeared very early in a sentence. Two modifications to the RNN is GRU and LSTM. We have used a bidirectional GRU for the encoder. We apply forward propagation 2 times , one for the forward cells and one for the backward cells. Both activations (forward , backward) would be considered to calculate the output. The decoder is a unidirectional GRU with attention mechanism over source, beam search and softmax over target variable. A GRU does not have a cell state and has two gates instead of three. It uses update and reset gate. Update gate decides how much information should be let through and the reset gate decided how much should be discarded.

We use our feature rich encoder to capture keywords. We use one embedding vector each for POS, NER tags and discretized TF and IDF values, which are concatenated together with word-based embeddings as input to the encoder. Finally, for each word in the source document, we simply look-up its embeddings from all of its associated tags and concatenate them into a single long vector. On the target side, we continue to use only word-based embeddings as the representation.

We used pointer-generator to model unseen or rare words and captured hierarchical document structure using attention mechanism.

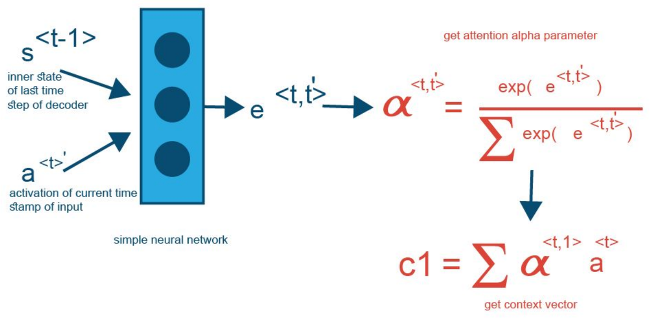


## Attention Mechanism

When we as humans summarize text , we actually look at couple of words at a time , not the whole text to summarize at a given instance , this is what we are trying to teach our model . We try to teach our model to only pay attention to the neighboring words not the whole text. We create a new interface between the encoder and decoder called the context vector. The attention actually depends on

1. the current activation of the input
2. the **previous state of the last time step in the decoder**

but we actually don't know the real function between them , so **we simply build a simple neural network to learn this relation** for us. The output from this network would be 'e' parameter that would be used to calculate the alpha attention parameter.


$s^{<t-1>}$
inner state of last time step of decoder
$a^{<t>'}$
activation of current time stamp of input
simple neural network
$e^{<t,t'>}$
get attention alpha parameter
$\alpha^{<t,t'>} = \frac{\exp( e^{<t,t'>} )}{\sum \exp( e^{<t,t'>} )}$
$c1 = \sum \alpha^{<t,1>} a^{<t>}$
get context vector

## Beam Search

Our task of text summarization can be seen as a conditional language model ,meaning that we generate an output given an input sentence , so the output is conditioned on the input sentence , so this is why it is called conditional. This can be seen as a contrary to a simple language model , as a normal language model only outputs the probability of a certain sentence , this can be used to generate novel sentences but in our case , it would select the most likely output given an input sentence. So our architecture is actually divided into 2 parts , an encoder , and a decoder as discussed earlier. The encoder would represent the input sentence as a vector , and pass it to the next part of the architecture which is the decoder. But a problem arises in the decoder , which sentence to select , as there could be a huge number of outputs for a certain input sentence. Then there arises a problem at the decoder, which sentence to choose. We can use greedy search which is generate the most likely word, then next most likely and so on. But in practice this is not the most optimum and we will use beam search. Simply beam search differs from basic greedy search by considering multiple options in every step , not just 1 option , these number of options is controlled by a variable called Beam Width.

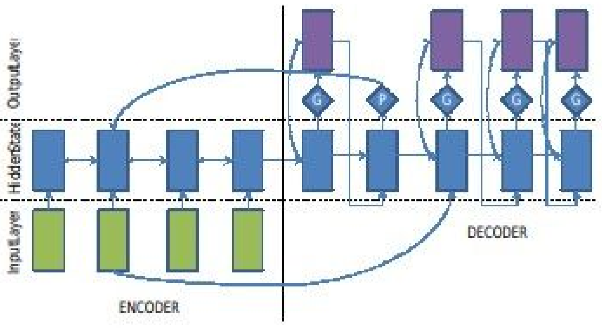
## Pointer Generator

It is a method of how to handle the novel/rare word in source document. The mathematical equation for it is shown next.

$$\begin{aligned} P(s_i = 1) &= \sigma(\mathbf{v}^s \cdot (\mathbf{W}^s_h \mathbf{h}_i + \mathbf{W}^s_e \mathbf{E}[o_{i-1}] \\ &+ \mathbf{W}^s_c \mathbf{c}_i + \mathbf{b}^s)), \end{aligned}$$

*i* = decoder timestep
$h_i$ = hidden state
$E[O_i$ *-1]* = embedding vec of emission from previous time step
$c_i$ = attention weighted context vector
Ws are switch parameters

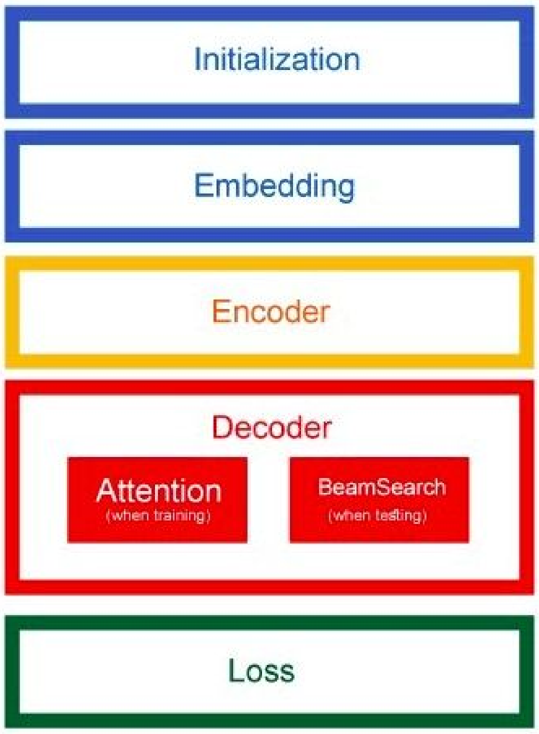
In this model, the decoder is equipped with a 'switch' that decides between using the generator or a pointer at every time-step. If the switch is turned on, the decoder produces a word from its target vocabulary in the normal fashion. However, if the switch is turned off, the decoder instead generates a pointer to one of the word-positions in the source. The word at the pointer-location is then copied into the summary. The switch is modeled as a sigmoid activation function over a linear layer based on the entire available context at each timestep as shown above. The pointer mechanism may be more robust in handling rare words because it uses the encoder's hidden-state representation of rare words to decide which word from the document to point to. Since the hidden state depends on the entire context of the word, the model is able to accurately point to unseen words although they do not appear in the target vocabulary.



**Implementation**

We have used tensorflow for implementation. The dataset that we used was the CNN new dataset. Even though we were given the dataset by Videoken of audio transcripts, we could not train on it as the datasets were of a poor quality and however much we tried prepare it, the data was not fit for the model. In fact, we even talked with an ex intern of Videoken who told us to use the code of the punctuator he built. We also felt that the semester was too short to train on the Videoken dataset after pre processing. However, we have tested on the dataset after training on the CNN dataset.

Our model's overview can be seen here :



In the initialization block, we defined the paths, placeholders and variables. We also defined the RNN cell here, that would be used throughout the model. We built our Model so that it would contain multiple parameters like:

1. embedding size (size of word2vector)
2. num_hidden (size of RNN)
3. num_layers (layers of RNN)
4. Learning Rate
5. BeamWidth
6. Keep Prob (dropout)

We also initialized the model with other parameters like:

1. reversed dict (dict of keys , each key which is a num points to a specific word)
2. Article_max_len & article_summary_len (max length of article sentence as input and max length of summary sentences output)
3. Forward Only (bool value to indicate training or testing phase) (Forward Only = False → training phase)

In the embedding block, we represent both our inputs articles that would be the embedded inputs and the decoder inputs using GLOVE word embeddings. We chose this over word2vec because Word2Vec is a "predictive" model that predicts context given word, GLOVE learns by constructing a co-occurrence matrix (words X context) that basically count how frequently a word appears in a context. Since it's going to be a gigantic matrix, we factorize this matrix to achieve a lower-dimension representation. We also explored ConceptNet word embeddings. They are used when we have a multilingual NLP problem or machine translation problem. They were adding unnecessary complexity to our code.

Next we will see the encoder block. Here we define the multilayer bidirectional lstm for the encoder part of our seq2seq, we define our variables here in a name scope that we call "encoder". We also use the concept of Dropout. We use it after each cell in our architecture , it is used to randomly activate a subset of our net, and is used during training for regularization. Now after defining the forward and backward cells , we actually connect them together to form the bidirectional structure. We use the stack_bidirectional_dyn-amic_rnn which gives output to fed into the decoder.

About the decoder, the decoder is divided into 2 parts training part (to train attention model), testing/running part (for attention & beam search). We first define out (name scope) & (variable scope) for both parts , we also define a multilayer cell structure that would be also used for both parts. We use Bahadanu attention. Now we would need to define the inputs to the decoder cell , this input actually comes from 2 sources. The encoder output (used within initial step) and the decoder input (summary sentence in the training phase). Finally we define the outputs , that would actually directly reflect to the real output from the whole seq2seq architecture , as this phase is where prediction is actually computed. Then in the loss block is where training actually occurs , here training actually occurs through multiple steps. By calculating loss, calculating gradients and applying clipping on gradients(to prevent exploding gradients) and then applying optimizer (here we use Adam optimizer).

Then we calculate the rouge and bleu scores.

## Observations and Results

**Summaries generated from Transcripts :**

***Article****:* In our lecture we looked at the corporate social responsibility and also the ethics.

***Summary****:* corporate groups defend ethics.

***Article****:* Let us now turn our attention to the all-pairs shortest paths problems where we try to find the shortest paths.

***Summary****:* a past road <unk> us job problems.

***Article****:* We allow negative edge weights but not negative cycles because, with negative cycles, our shortest path is not well defined.

***Summary*** : <unk> is not life among provided.

***Article****:* Deep learning started showing a lot of promise in a lot of fields like NLP, vision, speech.

***Summary****:* tiny penalized for a change of era descent.

**Summaries generated from News Dataset**

***Article****:* us business leaders lashed out Wednesday at legislation that would penalize companies for employing illegal immigrants.

***Summary****:* us business leaders lash out at illegal immigrants

*For recent news*

***Article:*** Voting is under way for Lok Sabha seats spread across 11 states and union territories.

***Summary:***<unk> underway for comparison

*Recent News from CNN site( As the model was trained on CNN dataset)*

***Article:*** Russian authorities are growing increasingly concerned about a polar bear that has been wandering in a village hundreds of miles from its usual habitat in search of food, according to local reports.

***Summary:*** search widens in of bear from home area

## Conclusion

According to me the training dataset is news dataset is very different from the dataset that I am testing on and hence whatever I tried, the results did not seem to be improving. Moreover, another issue is that news datasets are somewhat similar or relating to certain common topics, so it probable for the model to predict. In transcripts, the challenges are more. All transcripts are very different from each other, they have no relation. One is about Social responsibility, Other about Deep learning etc and hence the model gets confused. The transcripts are also very messy, at places "and" is written as "ah" etc. A fix to this is to manually clean up the dataset and use it to train the model.

## Future Work

In future I hope to work on reinforcement learning and further train the model. I also want to explore GANs and BERT model.

## Bibliograhy

*Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond( Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, Bing Xiang)*

*A Video Summarization Approach Based on Machine Learning(W. Ren and Y. Zhu)*

*VideoKen: Automatic Video Summarization and Course Curation to Support Learning*

*Generative Adversarial Network for Abstractive Text Summarization( Linqing Liu Yao Lu Min Yang1 Qiang Qu, Jia Zhu Hongyan Li )*

https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f

*https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python*