|  |
| --- |
|  |
| Abstract Summarization |
|  |
|  |
| **Hameed, Sameer** |
| **9/25/2018** |

|  |
| --- |
| Document to cover terminologies and technical stack used to generate Abstract summarization |

Table of Contents

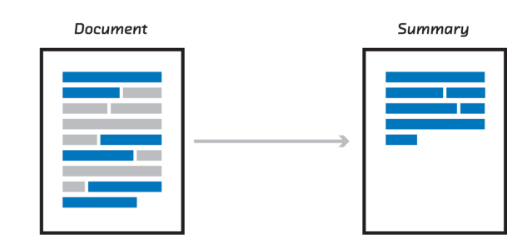
1. What is Summarization1
2. Abstractive Summarization – Scope 3
3. Application Flow 4
4. Architecture Design 4

* Data Processing (or Pre-Processing) 5
* Word Embedding5
* Deep Learning & Neural Network5
* Seq2Seq (Encoder – Decoder) 6
* Recurrent Neural Network (RNN) 6
* Bi-directional Long Short Term Memory (Bi-LSTM) 7
* Attention 8
* Beam Search 9
* ROUGE 10
* Overview Tensorflow12

**1. What is Summarization?**

With the dramatic growth of the Internet, people are overwhelmed by the tremendous amount of online information and documents. This expanding availability of documents has demanded exhaustive research in automatic text summarization. According to [1] a summary is defined as “a text that is produced from one or more texts, that conveys valuable information in the original text(s), and that is no longer than half of the original text(s) and usually, significantly less than that”.

From many years, summarization is done by humans manually. Text summarization relates to the process of obtaining a textual document, obtaining content from it and providing the necessary content to the user in a shortened form and in a receptive way to the requirement of user or application. In the present time, the amount of information is increasing gradually by the mean of internet and by other sources. To overcome this problem, Automatic text summarization could play a vital role. Researchers in text summarization have approached this problem from many aspects such as natural language processing (Zhang et al., 2011), statistical (Darling and Song, 2011) and machine learning and text analysis is the fundamental issue to identify the focus of the texts.



Text Summarization plays an important role in the area of text mining and natural language processing. Text summarization aims to compress the source text into a shorter and concise form with preserving its information content and overall meaning

Text summarization can be classified in two ways, as abstractive summarization and extractive summarization.. Below is the comparison between two -

**Extractive summarization**

* Copying parts/sentences of the source text and then combines those part/sentences together to render a summary.
* Importance of sentence is based on linguistic and statistical features.
* Extractive is easy to implement but not quite accurate.

**Abstractive summarization**

* These methods try to first understand the text and then rephrase it in a shorter manner, using possibly different words
* For perfect abstractive summary, the model has to first truly understand the document and then try to express that understanding in short possibly using new words and phrases.
* Much harder than extractive.
* Have complex capabilities like generalization, paraphrasing and incorporating real-world knowledge.
* Abstractive is complex but gives more human like summary.

Scope of this document is restricted to single document Abstractive summarization.

**2. Abstractive Summarization - Scope**

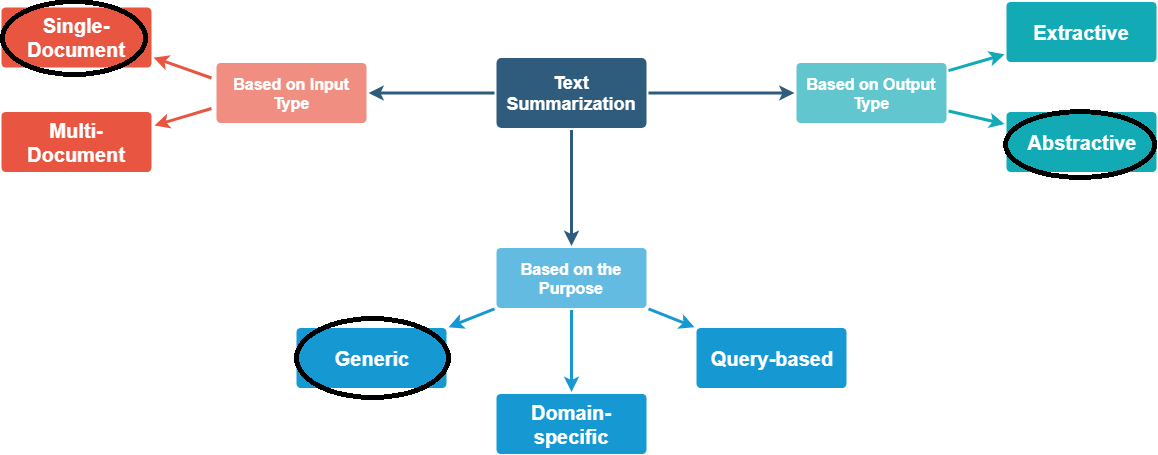
In August 2016, Peter Liu and Xin Pan, software engineers on Google Brain Team, published a blog post [2]. Their algorithm is extracting interesting parts of the text and creates a summary by using these parts of the text and allows for rephrasings to make summary more grammatically correct. This approach is called abstractive summarization. This approach is more like human summarization. For example –

**Original Text:** Alice and Bob took the train to visit the zoo. They saw a baby giraffe, a lion, and a flock of colourful tropical birds.

**Abstractive summary:** Alice and Bob visited the zoo and saw animals and birds.

There are many reasons why Automatic Text Summarization is useful [14]:

* Summaries reduce reading time.
* When researching documents, summaries make the selection process easier.
* Automatic summarization improves the effectiveness of indexing.
* Automatic summarization algorithms are less biased than human summarizers.
* Personalized summaries are useful in question-answering systems as they provide personalized information.
* Using automatic or semi-automatic summarization systems enables commercial abstract services to increase the number of text documents they are able to process.
* **Types of Text Summarization approaches**

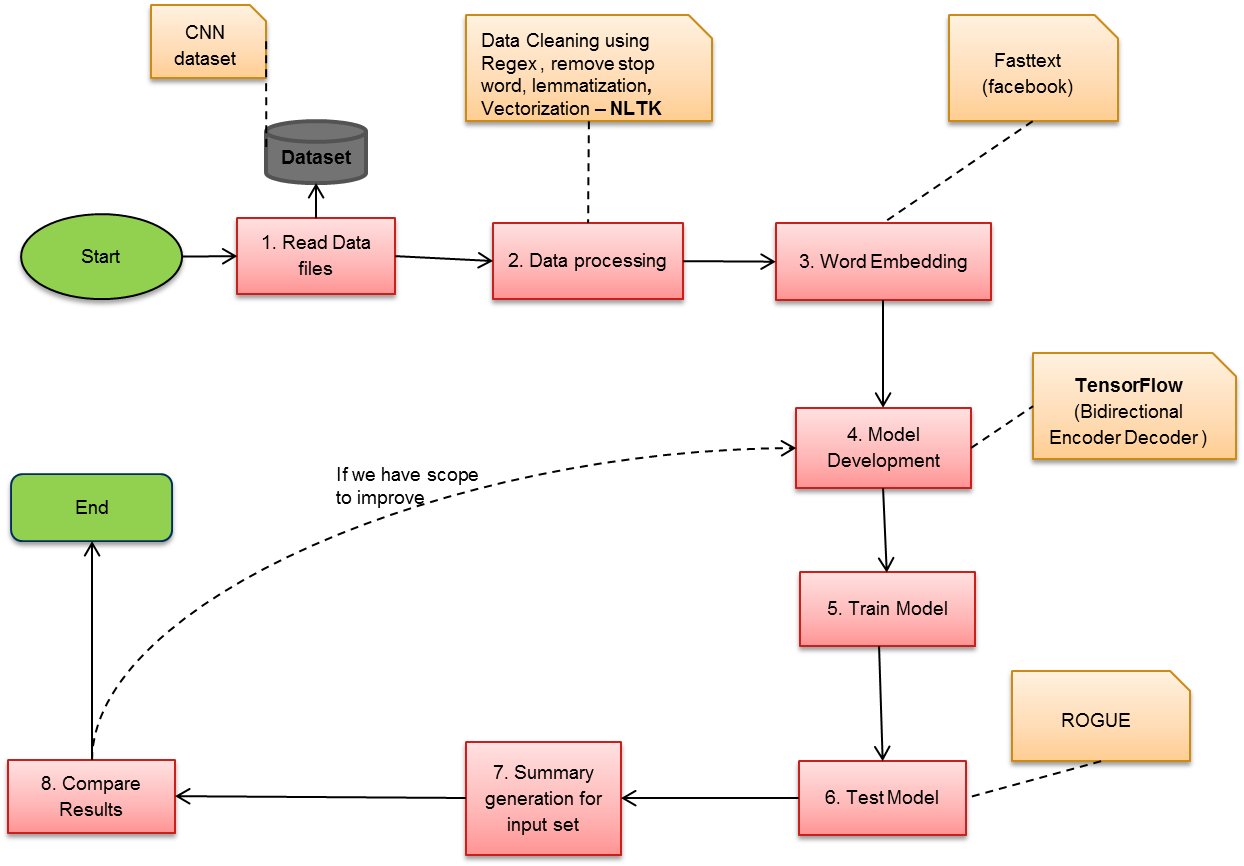


In this paper, we are here trying make a system that would – Take ***Single Document*** and provides ***Abstractive Generic*** summary.

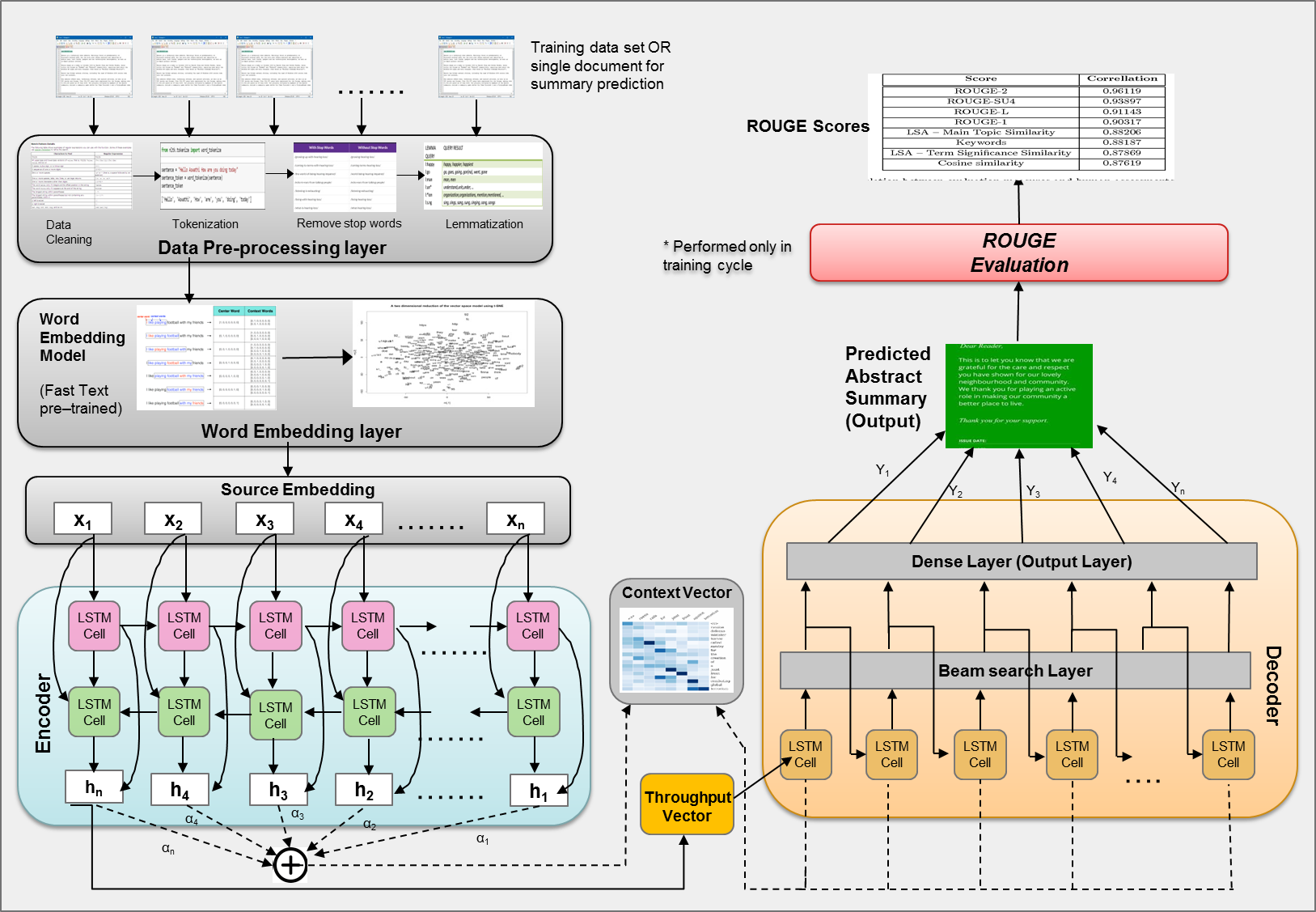
The proposed model consists of two parts, i.e., the document encoder and the sentence extractor. The document encoder has a hierarchical architecture, which suits the compositionality of documents. The sentence extractor is built with recurrent neural networks (RNN), which provides two main functionalities. On one hand, the RNN is used to remember the partial output summary by feeding the selected sentence into it. On the other hand, it is used to provide a sentence extraction state that can be used to score sentences with their representations. At each step during extraction, the sentence extractor reads the representation of the last extracted sentence. It then produces a new sentence extraction state and uses it to score the relative importance of the rest sentences. We conduct experiments on the CNN/Daily Mail dataset.

**3. Application Flow**

Below are the steps we would be covering while coming up with tool that could give us Abstractive Summary of any article.

****

**4. Architecture** **Design**



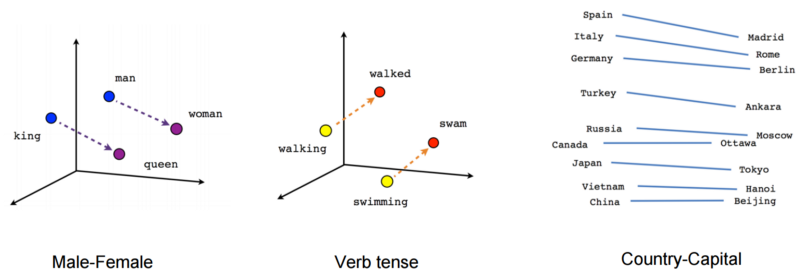
Before we move on to further things lets discuss about the basic concepts that we have put in our application from and Architecture Design, so that it will be easy to image what is happening inside.

* **Data Processing (or Pre-Processing)**

Pre-processing is an important task and critical step in Text mining, Natural Language Processing (NLP) and information retrieval (IR). In the area of Text Mining, data preprocessing used for extracting interesting and non-trivial and knowledge from unstructured text data. I idea is to create a dataset which could later be used to create valid vocabulary.

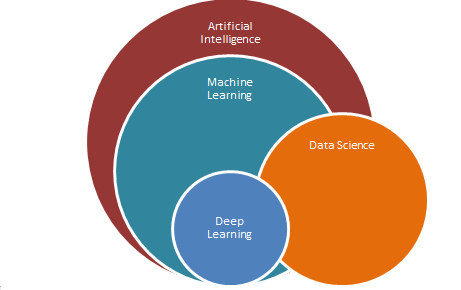
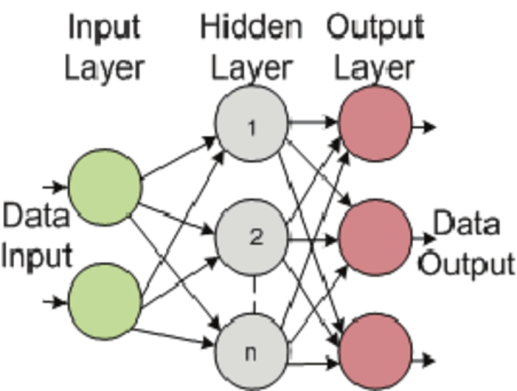
* **Cleaning** consist of getting rid of the less useful parts of text through stopword removal, *Punctuation Removal* and dealing with capitalization and characters and other details.
* **Annotation** consists of the application of a scheme to texts. Annotations may include structural markup and [part-of-speech](https://en.wikipedia.org/wiki/Lexical_category) tagging.
* **Normalization** consists of the translation (mapping) of terms in the scheme or linguistic reductions through Lemmazation.
* **Word Embedding [13] -**

Word embedding is the modern way of representing words as vectors. The aim of word embedding is to redefine the high dimensional word features into low dimensional feature vectors. In other words it represents words at an X and Y vector coordinate where related words, based on a corpus of relationships, are placed closer together. [Word2Vec](https://code.google.com/archive/p/word2vec/), [GloVe](http://nlp.stanford.edu/projects/glove/) and [Conceptnet Numberbatch](http://blog.conceptnet.io/posts/2016/conceptnet-numberbatch-a-new-name-for-the-best-word-embeddings-you-can-download/) are the most common models to convert text to vectors. In our case we are using [FastText](https://fasttext.cc/docs/en/english-vectors.html).



* **Deep Learning & Neural Network**

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans. We would be using this technique to make sure we get human like summary from this tool. While traditional [machine learning](https://searchenterpriseai.techtarget.com/definition/machine-learning-ML) algorithms are linear, deep learning [algorithms](https://whatis.techtarget.com/definition/algorithm) are stacked in a [hierarchy](https://whatis.techtarget.com/definition/hierarchy) of increasing complexity and abstraction. It utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web.

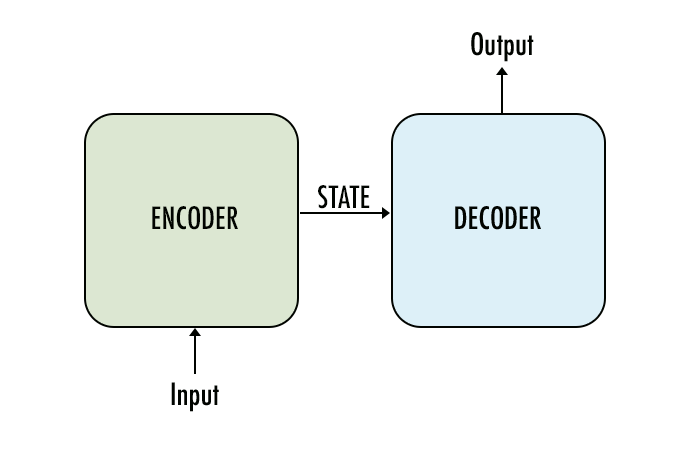
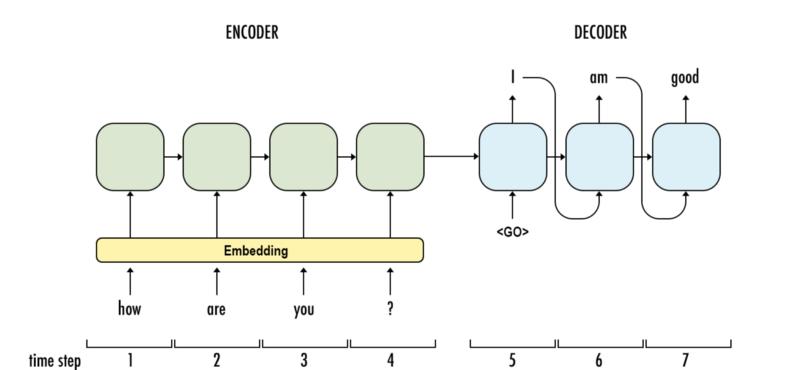
 

Neural Networks (NN) are also used for Natural Language Processing (NLP), including Summarizers. Neural networks are effective in solving almost any machine learning classification problem. Important parameters required in defining the architecture of neural network (NN) are number of hidden layers to be used, number of hidden units to be present in each layer, activation function for each node, error threshold for the data, the type of interconnections, etc. neural networks can capture very complex characteristics of data without any significant involvement of manual labour as opposed to the machine learning systems. Deep learning uses deep neural networks to learn good representations of the input data, which can then be used to perform specific tasks.

* **Seq2Seq (Encoder – Decoder)**

The Encoder-Decoder architecture is a way of organizing recurrent neural networks for sequence prediction problems that have a variable number of inputs, outputs, or both inputs and outputs. The architecture involves two components: an encoder and a decoder.

* **Encoder**: The encoder reads the entire input sequence and encodes it into an internal representation, often a fixed-length vector called the context vector. In our case, encoder is responsible for reading the source document and encoding it to an internal representation.
* **Decoder**: The decoder reads the encoded input sequence from the encoder and generates the output sequence. Here, decoder is a language model responsible for generating each word in the output summary using the encoded representation of the source document.

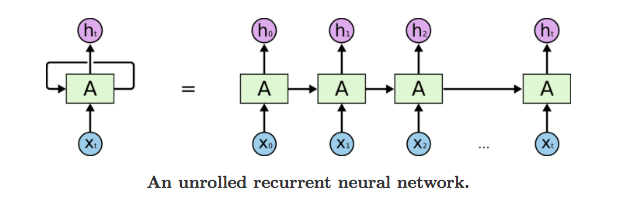
 

* **Recurrent Neural Network (RNN)**

When we human read or heard any sentence, we understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence. Traditional neural networks can’t do this, and it seems like a major shortcoming. If you want neural network to analyse anything you need to give them things scratch.

Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input. Sometimes the context is the single most important thing for the model to predict the most appropriate output. Suppose you are watching a movie, you keep watching the movie as at any point in time, you have the context because you have seen the movie until that point, then only you are able to relate everything correctly. It means that you remember everything that you have watched.

Similarly, RNN remembers everything. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other

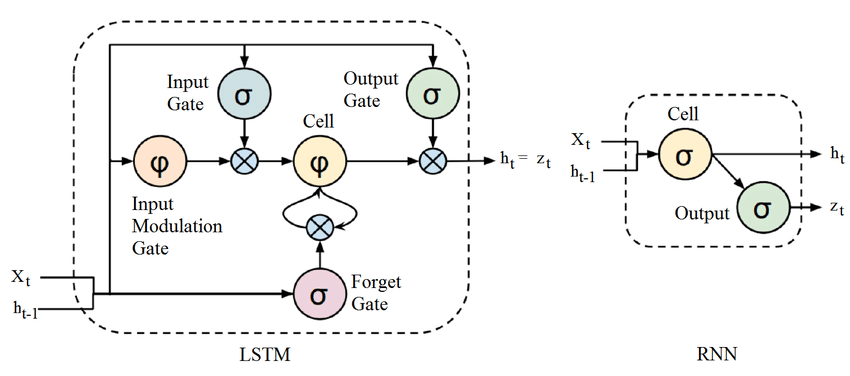


In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They’re the natural architecture of neural network to use for such data. And they certainly are used! In the last few years, there have been incredible successes applying RNNs to a variety of problems: speech recognition, language modelling, translation, image captioning… The list goes on.

Theoretically RNNs can handle context from the beginning of the sentence which will allow more accurate predictions of a word at the end of a sentence. In practice this isn’t necessarily true for vanilla RNNs. This is a major reason why RNNs faded out from practice for a while until some great results were achieved with using a Long Short Term Memory (LSTM) unit inside the Neural Network.

* **Bi-directional Long Short Term Memory (Bi-LSTM) [6]**

The LSTM is RNN architecture which can remember past contextual values. These stored values do not change over time while training the model. A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.



Intuitively, the ***input gate*** controls the extent to which a new value flows into the cell, the ***forget gate*** controls the extent to which a value remains in the cell and the ***output gate*** controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

To understand the short and long term memories consider this two sequences below:

Sequence (1): x x x A B x x

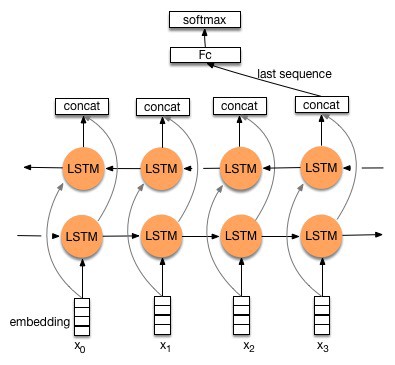
Sequence (2): x A x x x B x

In sequence 1, the character "B" appears immediately after "A" (x elements could be any character). "AB" is a pattern in our sequence. By feeding more examples of data samples that include this pattern, the RNN memorize that after "A" there must be a "B". It is a short-term memory. So if the model receives the "AB" pattern again it will remember its order and can predict the corresponding output.

In sequence 2, the character "B" comes 4 time-steps after the character "A". A simple RNN directly copies the internal state of the previous state and combines it with the input. If the model receives new data during these 4 steps, they will destroy the memory of the "A" character. To learn a long pattern like "A x x x B" we need a new mechanism.

Bi-LSTM [6] is a combination of Long Short-Term Memory (LSTM) and Bi-directional Recurrent Networks (Bi- RNN) 12. Recurrent Neural Network (RNN) which is a special development of Artificial Neural Networks (ANN) to process sequences and time series data. Unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past.

Using bidirectional will run your inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backwards you preserve information from the future and using the two hidden states combined you are able in any point in time to preserve information from both past and future. This structure allows the networks to have both backward and forward information about the sequence at every time step



**[Note –** This architecture is used in Google Translation ☺ ]

* **Attention [7]**

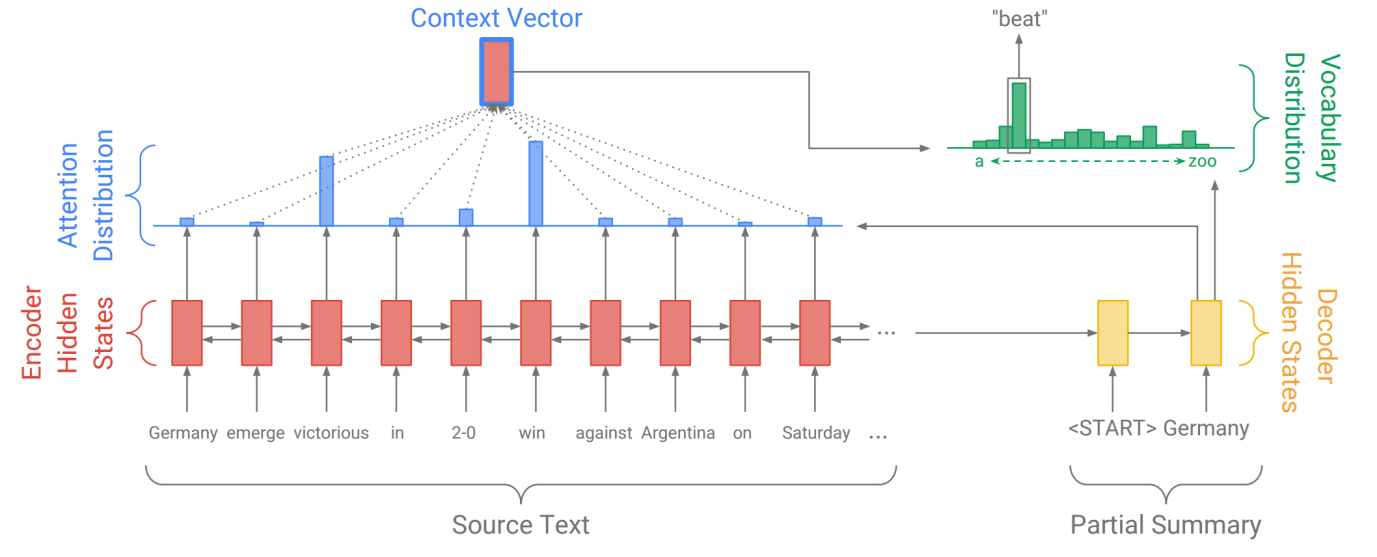
Before Attention mechanism, summarization tools relies on reading a complete sentence and compress all information into a fixed-length vector, as you can image, a sentence with hundreds of words represented by several words will surely lead to information loss, inadequate translation, etc. However, attention partially fixes this problem. It allows tools to look over all the information the original sentence holds, then generate the proper word according to current word it works on and the context. It can even allow translator to zoom in or out (focus on local or global features).

Let suppose we have -

**Source Text** - *Germany emerge victorious in 2-0 win against Argentina on Saturday*

**Expected output** - *Germany* beat Argentina 2-0

First, the encoder RNN reads in the source text word-by-word, producing a sequence of encoder hidden states. (There are arrows in both directions because our encoder is bidirectional). Once the encoder has read the entire source text, the decoder RNN begins to output a sequence of words that should form a summary. On each step, the decoder receives as input the previous word of the summary (on the first step, this is a special <START> token which is the signal to begin writing) and uses it to update the decoder hidden state. This is used to calculate the attention distribution, which is a probability distribution over the words in the source text. Intuitively, the attention distribution tells the network where to look to help it produce the next word. In the diagram above, the decoder has so far produced the first word Germany, and is concentrating on the source words win and victorious in order to generate the next word beat.



Next, the attention distribution is used to produce a weighted sum of the encoder hidden states, known as the *context vector*. The context vector can be regarded as “what has been read from the source text” on this step of the decoder. Finally, the context vector and the decoder hidden state are used to calculate the *vocabulary distribution*, which is a probability distribution over all the words in a large fixed vocabulary (typically tens or hundreds of thousands of words). The word with the largest probability (on this step, *beat*) is chosen as output, and the decoder moves on to the next step.

The decoder’s ability to freely generate words in any order – including words such as *beat* that do not appear in the source text – makes the sequence-to-sequence model a potentially powerful solution to abstractive summarization.

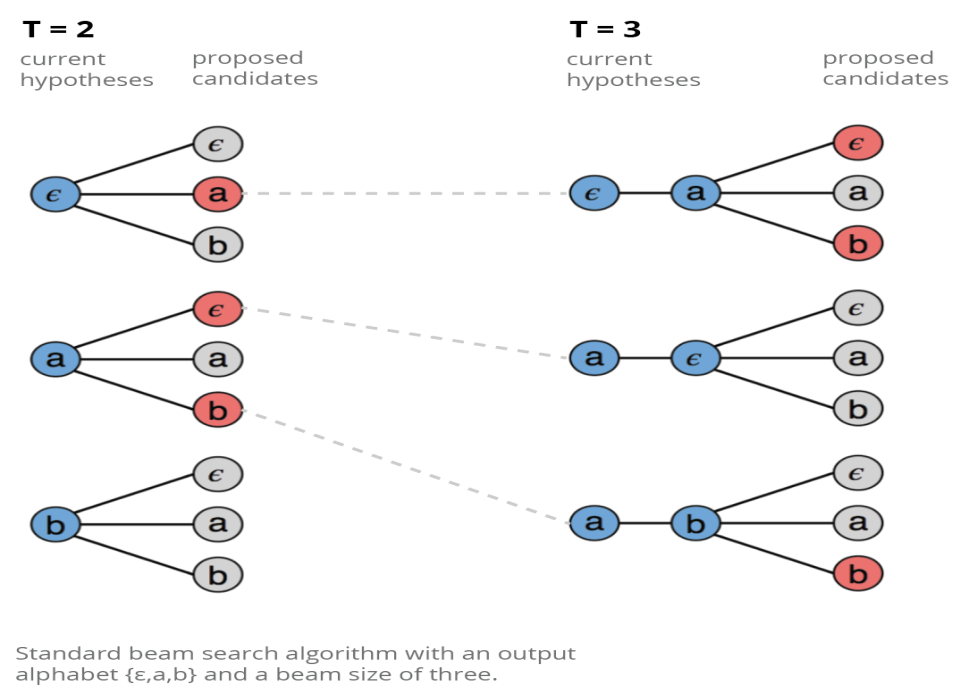
* **Beam Search [8]**

Beam search is a restricted, or modified, version of either a [breadth-first search](https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Exhaustive_search/Breadth-first_search). It is restricted in the sense that the amount of memory available for storing the set of alternative search nodes is limited, and in the sense that non-promising nodes can be pruned at any step in the search.

Other algorithms such as Greedy search only predicts the token with the best score, whereas Beam search keep track of k hypotheses (for example k=3, we refer to k as the beam size). At each new time step, for these 3 hypotheses we have V new possible tokens. It makes a total of 3V new hypotheses. Then, only keep the 3 best ones, and so on… Formally, define Ht the set of hypotheses decoded at time step t.

***Ht:={(w11,…,w1t),…,(wk1,…,wkt)}***

As "k refers to the size of the beam for generation; k = 1 implies greedy generation."

So, beam width, is a parameter in the beam search algorithm which determines how many of the best partial solutions to evaluate. In an LSTM model of melody generation, for example, beam size limits the number of candidates to take as input for the decoder. A beam size of 1 is a best-first search - only the most probable candidate is chosen as input for the decoder. A beam size of k will decode and evaluate the top *k* candidates. A large beam size means a **more extensive search** - not only the single best candidate is evaluated.  


**For example** if my sentence is - **You know nothing john snow**

For (**know**) H3 = {(know nothing), (know John), (know snow)}

* **ROUGE [9]**

***ROUGE***, or ***Recall-Oriented Understudy for Gisting Evaluation***, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation. It includes measures to automatically determine the quality of a summary by comparing it to other (human-produced) summaries created by humans. The measures count the number of overlapping units such as n-gram, word sequences, and word pairs be-tween the computer-generated summary to be evaluated and the ideal summaries created by humans.

Rogue offers many kinds of measures like

* ROUGE-N
* ROUGE-L
* ROUGE-W
* ROUGE-S

These methods use combines Precision, Recall and overlap in the measures it produces. To understand them – Let’s say we have two summaries –

#### Human Generated Summary -

You know nothing john snow

#### Summary generated by your algorithm

You know nothing little john snow

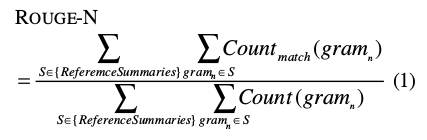
**Recall** is how much of the generated summary captures the original / human generated summary.

Recall = (no. of common words / no. of words in ground truth) = 5/5 = 1

**Precision** measures how much of the generated summary is needed or relevant.

Precision = (no. of common words / no. of words in generated summary) = 5/6

* **ROUGE-N :** It calculates the n-gram recall for the generated summary and the ground truth summary.  
  For example ROUGE-2 will calculate recall for bigrams in the summary. For the previous example



**Bigrams in Gold Standard Bigrams in generated summary**

*you know oh you*

*know nothing you know*

*nothing john know nothing*

*john snow nothing little*

*little john*

*john snow*

**In this case-**

ROUGE2 Recall = 3 / 4

ROUGE2 Precision = 3 / 6

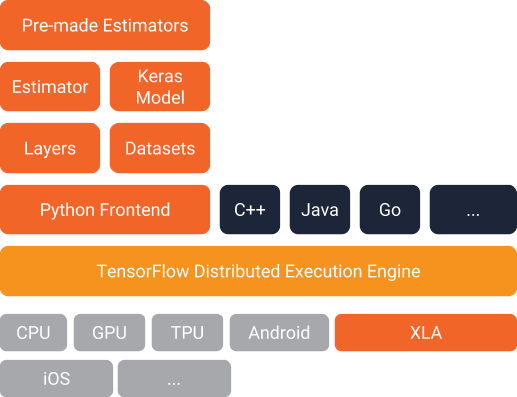
* **ROUGE-L** measures the overlap between summaries based on the longest common sub-sequences (LCS) encountered in the summaries. It uses a LCS-based F-measure to evaluate two summaries X of length m and Y of length n, where X is a gold standard summary and Y is a generated summary sentence.
* **ROUGE-W**: Rouge-L measures takes only sub-sequences into consideration, it does not check if the sub-sequences are consecutive or not. To improve the LCS method we can use a weighted scheme which will prefer consecutive sub-sequences over non-consecutive ones. One can simply store the length of consecutive sub-sequence found so far. So this way consecutive common sub- sequences are awarded more score than the ones which are common but not consecutive.
* **ROUGE-S** This is a co-occurrence statistics based on Skip-bi-grams. A Skip-bi-gram is a bi-gram which can have arbitrary gap between them. For the sentence.

Skip bi-grams are:

* + you know
  + you nothing
  + you john
  + you snow
  + know nothing
  + know john
  + know snow
  + nothing john
  + nothing snow
  + john snow
* **Overview Tensorflow [10]**

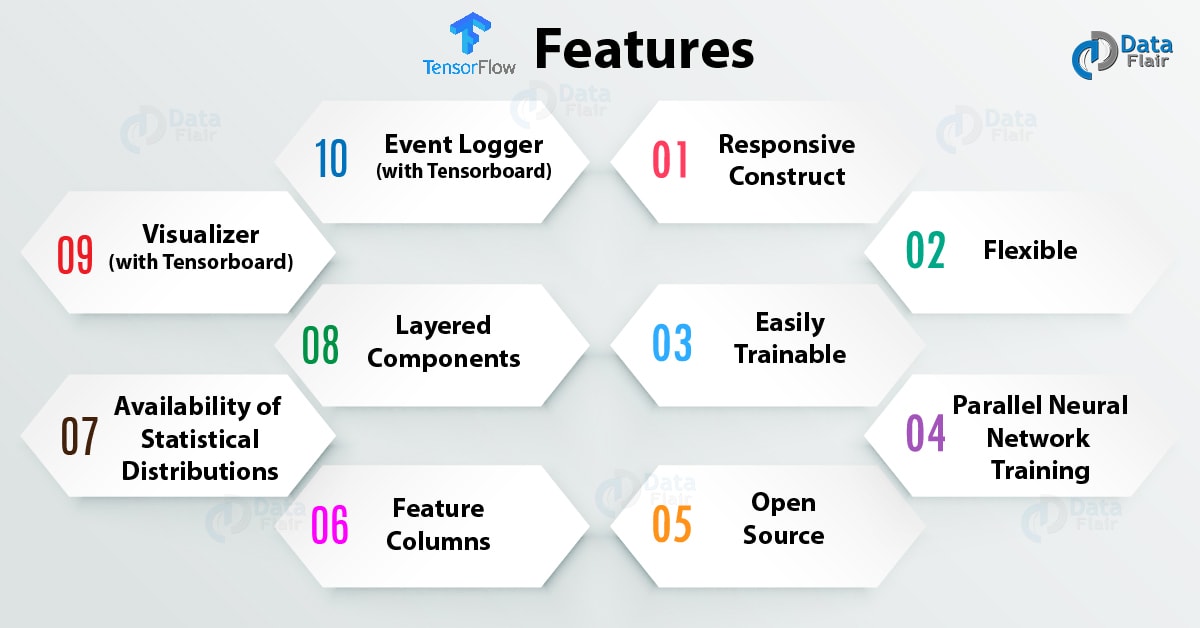
Tensorflow is a computational framework for building machine learning models. TensorFlow provides a variety of different toolkits that allow you to construct models at your preferred level of abstraction. You can use lower-level APIs to build models by defining a series of mathematical operations. Alternatively, you can use higher-level APIs (like tf.estimator) to specify predefined architectures, such as linear regressors or neural networks.

The following figure shows the current hierarchy of TensorFlow toolkits:



TensorFlow is cross-platform. It runs on nearly everything: GPUs and CPUs—including mobile and embedded platforms—and even tensor processing units (TPUs), which are specialized hardware to do tensor math on.

The single biggest benefit TensorFlow provides for machine learning development is abstraction. Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application. TensorFlow takes care of the details behind the scenes.



## TensorFlow Advantages

* TensorFlow has a responsive construct as you can easily visualize each and every part of the graph.
* It has platform flexibility, meaning it is modular and some parts of it can be standalone while the others coalesced.
* It is easily trainable on CPU as well as GPU for distributed computing.
* It has auto differentiation capabilities, which benefit gradient based machine learning algorithms, meaning you can compute derivatives of values with respect to other values, which results in a graph extension.
* It has advanced support for threads, asynchronous computation, and queues.
* It is customizable and open source.

## TensorFlow Limitations

* Have GPU memory conflicts with Theano if imported in the same scope.
* No support for OpenCL
* Requires prior knowledge of advanced calculus and linear algebra along with a pretty good understanding of machine learning.

Other Deep learning frameworks are mentioned in reference [11] section-

**5. References**

[1] Dragomir R Radev, Eduard Hovy, and Kathleen McKeown. 2002. Introduction to the special issue summarization. Computational linguistics 28, 4 (2002), 399–408.

[2] [Text summarization with TensorFlow](https://research.googleblog.com/2016/08/text-summarization-with-tensorflow.html)

[3] [Pre Processing methods](https://towardsdatascience.com/pre-processing-in-natural-language-machine-learning-898a84b8bd47)

[4] [Logic of Pre processing](https://www.kaggle.com/saxinou/nlp-01-preprocessing-data)

[5] LSTM -

<https://upcommons.upc.edu/bitstream/handle/2117/119051/131670.pdf>

<https://en.wikipedia.org/wiki/Long_short-term_memory>

[6] Bidirectional recurrent neural networks (RNN)

<https://towardsdatascience.com/seq2seq-model-in-tensorflow-ec0c557e560f>

[7] Attention

<http://www.abigailsee.com/2017/04/16/taming-rnns-for-better-summarization.html>

[8] Beam search

<https://www.coursera.org/lecture/nlp-sequence-models/beam-search-4EtHZ>

<https://www.youtube.com/watch?v=RLWuzLLSIgw>

<https://hackernoon.com/beam-search-a-search-strategy-5d92fb7817f>

[9] ROGUE

<http://rxnlp.com/how-rouge-works-for-evaluation-of-summarization-tasks/>

<http://iiitd-ir-melanage.blogspot.com/2018/03/text-summarization-evaluating-summaries.html>

[10] Tensorflow

<https://opensource.com/article/17/11/intro-tensorflow>

<https://www.tensorflow.org/tutorials/>

<https://www.datacamp.com/community/tutorials/tensorflow-tutorial>

https://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html

[11] Deep Leaning frameworks

<https://cv-tricks.com/deep-learning-2/tensorflow-or-pytorch/>

<https://www.exastax.com/deep-learning/a-comparison-of-deep-learning-frameworks/>

# [12] Coding with Seq2Seq model in Tensorflow –

# <https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66>

# https://developers.google.com/machine-learning/crash-course/

[13] Word embedding

<https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285>

[14] [Automatic Text Summarization](http://amzn.to/2fhUPNt),