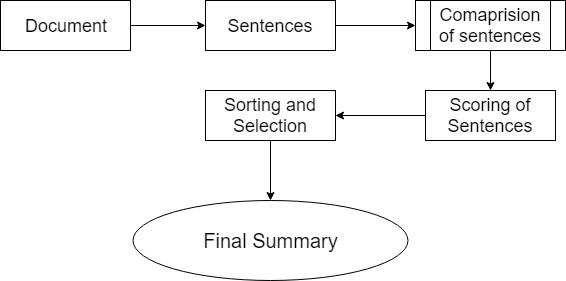
**Case Study: Text Summarization**

**Problem statement:** Build a text summarization system to highlight a summary of a given document (news article). The training and testing data set is provided for the same. **Generate both extractive as well as abstractive summary separately.** (If not able to implement the abstractive text summarization, please do send the one-page document explaining the approach)

**Solution:**

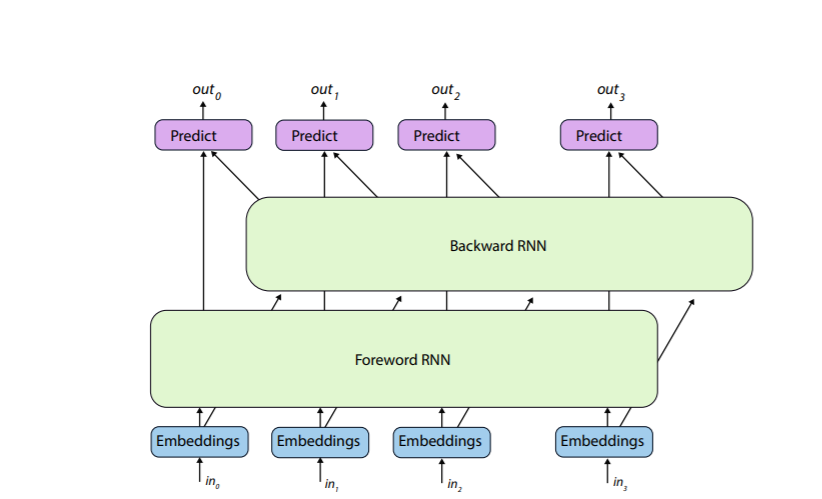
There are two prominent types of summarization algorithms

* **Extractive summarization** systems form summaries by copying parts of the source text through some measure of importance and then combine those part/sentences together to render a summary. Importance of sentence is based on linguistic and statistical features.
* **Abstractive summarization** systems generate new phrases, possibly rephrasing or using words that were not in the original text. Naturally abstractive approaches are harder. For perfect abstractive summary, the model must first truly understand the document and then try to express that understanding in short possibly using new words and phrases. Much harder than extractive. Has complex capabilities like generalization, paraphrasing and incorporating real-world knowledge.



Have built tf-seq2seq with the following goals in mind:

Baseline Neural Attention Model the Neural Attention Model] consists of an encoder-decoder RNN with attention Mechanism. The inputs that feed into the RNN are word/morpheme/phrases embeddings. Have use word embeddings for summarization.



• wi - input tokens of source article

• hi - Encoder hidden states

• P(vocab) = softmax(V hi + b) is the distribution over vocabulary from which we sample out There are many variants of RNN’s like LSTM’s and GRU’s and NAS. LSTM captures long-term dependency well and the information in the starting of the sequence is able to traverse down the line. This is done by selectively selecting and restricting the flow of information in the LSTM unit. There are three gates in an LSTM. Forget Gate Layer

• ft = σ(Wf [ht−1, xt] + bf ) • Ct = Ct ⊗ ft

In the vanilla encoder-decoder model, the encoder RNN first reads in the source string word by word, encoding the information in a hidden state and passing the context forward. On a complete pass, the encoder i.e. the Bi-Directional RNN produces an encoding of the string or a though vector so to speak which captures all the information and context of the input string. The decoder is another RNN which learns to decode the thought vector into output sequence

Attention Mechanism The basic encoder-decoder model performs okay on very short sentences, but it fails to scale up.

• The main bottleneck is the fixed sized encoding of the input string, which is the LSTM output of the last time-step. Due to its fixed size it is not able to capture all the relevant information of the input sequence as the model sizes up.

• At each generation step, only a part of the input is relevant. So how does the model decide which part of the input encoding to focus on at each generation step?

• At each step, the decoder outputs hidden state h(i) , from which we generate the output. The solution to this problem is using the attention model. This model calculates the importance of each input encoding for the current step by doing a similarity check between decoder output at this step and input encodings. Doing this for all of the input encodings and normalizing, we get an importance Vector. We then convert it to probabilities by passing through SoftMax. Then form a context vector by multiplying with the encodings.

• importance(it) = V ∗ tanh (eiW1 + htW2 + b(attn)).

• Attention Distribution a^ t = SoftMax(importance(it))

• Context Vector h(t) = Summation of (e(i) ∗ a ^ t ) .

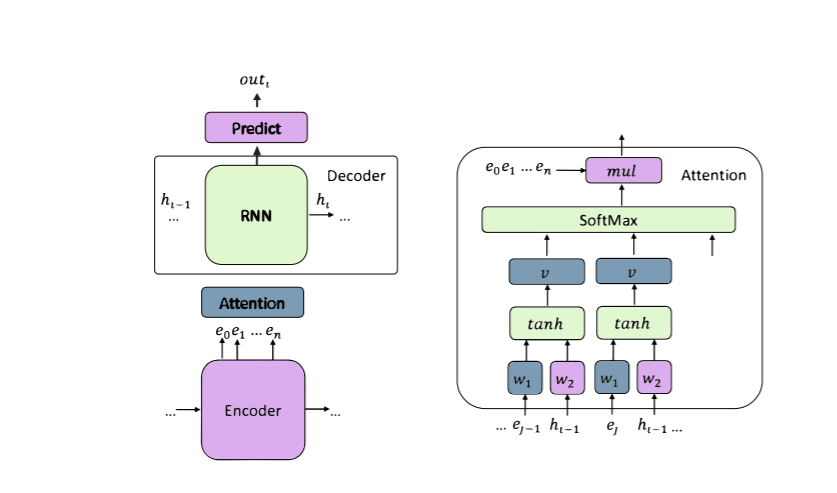
Context Vector is then fed into two layers to generate distribution over the vocabulary from which we sample.

• P(vocab(w)) = SoftMax (V’(V [h(t), h∗ t ] + b) + b’ )

• For the loss at time step t, loss(t)= − log P(w\*(t) ), where w ∗ t is the target summary word.

• LOSS = 1/ T Summation of(loss(t)) is the total loss across all timesteps.

• Have use the Backpropagation through time Algorithm to get the gradients and then apply any of the popular Gradient Descent algorithms to minimize the loss and learn good parameters.



**Benefits of the above mechanism:**

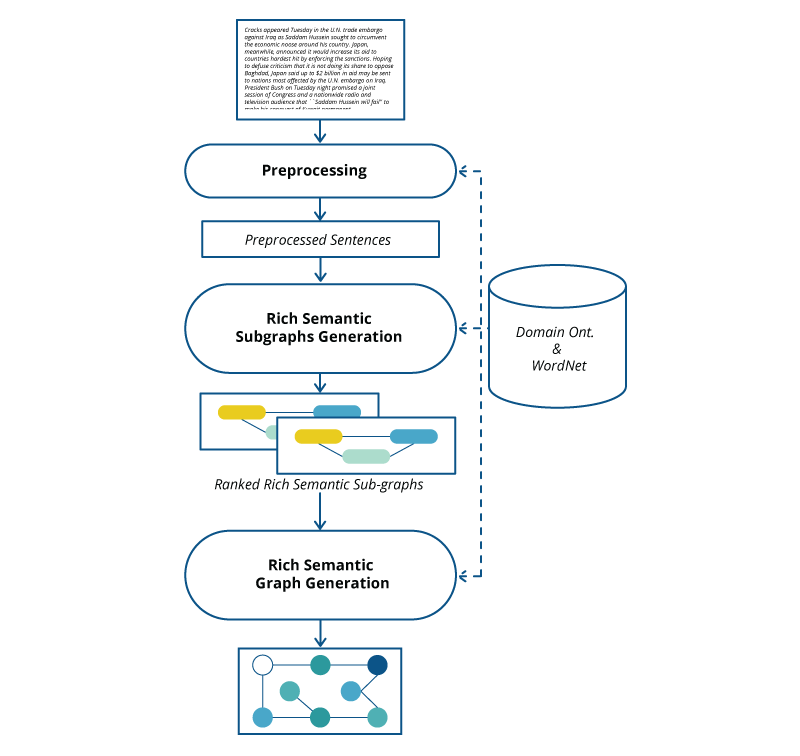
* **General Purpose**: We initially built this framework for Machine Translation but have since used it for a variety of other tasks, including Summarization, Conversational Modelling, and Image Captioning. If your problem can be phrased as encoding input data in one format and decoding it into another format, you should be able to use or extend this framework.
* **Usability**: You can train a model with a single command. Several types of input data are supported, including standard raw text.
* **Reproducibility**: Training pipelines and models are configured using YAML files. This allows other to run your exact same model configurations.
* **Extensibility**: Code is structured in a modular way and that easy to build upon. For example, adding a new type of attention mechanism or encoder architecture requires only minimal code changes.
* **Good Performance**: For the sake of code simplicity, we did not try to squeeze out every bit of performance, but the implementation is fast enough to cover almost all production and research use cases. tf-seq2seq also supports distributed training to trade off computational power and training time.

**Other Approach for abstractive summarization**

## [**PEGASUS**: A State-of-the-Art Model for Abstractive Text Summarization](http://ai.googleblog.com/2020/06/pegasus-state-of-art-model-for.html)

The dominant paradigm for training machine learning models to do this is [sequence-to-sequence](https://arxiv.org/abs/1409.3215) (seq2seq) learning, where a neural network learns to map input sequences to output sequences. While these seq2seq models were initially developed using [recurrent neural networks](https://en.wikipedia.org/wiki/Recurrent_neural_network), [Transformer](https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html) encoder-decoder models have recently become favored as they are more effective at modeling the dependencies present in the long sequences encountered in summarization.  
  
Transformer models combined with self-supervised pre-training (e.g., [BERT](https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html), [GPT-2](https://openai.com/blog/better-language-models/), [RoBERTa](https://arxiv.org/abs/1907.11692), [XLNet](https://arxiv.org/abs/1906.08237), [ALBERT](https://ai.googleblog.com/2019/12/albert-lite-bert-for-self-supervised.html), [T5](https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html), [ELECTRA](https://ai.googleblog.com/2020/03/more-efficient-nlp-model-pre-training.html)) have shown to be a powerful framework for producing general language learning, achieving state-of-the-art performance when fine-tuned on a wide array of language tasks. In prior work, the self-supervised objectives used in pre-training have been somewhat agnostic to the down-stream application in favor of generality; we wondered whether better performance could be achieved if the self-supervised objective more closely mirrored the final task.  
  
In “[PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization](https://arxiv.org/abs/1912.08777)” have designed a pre-training self-supervised objective (called *gap-sentence generation*) for Transformer encoder-decoder models to improve fine-tuning performance on abstractive summarization.

**Extractive Summarization: Graph Based Text Summarization**



This model generates an abstractive summary by repeatedly searching the graph for sub-graphs encoding a valid sentence and high redundancy scores to find meaningful paths which in turn becomes candidate summary phrases. All the paths are afterwards ranked in the descending order of the scores and duplicated paths are eliminated with the help of the Jaccard measure to create a short summary. The summarizer is considered a **“shallow” abstractive summarizer** as it uses the original text itself to generate summaries (this makes it shallow) but it can generate phrases that were previously not seen in the original text because of the way paths are explored.

**Other Suggested Methods:**

# Ontology-based methods

# Multimodal semantic model

# Information item-based methods

# Semantic Graph Model