Neural Networks Extractive Summariser

Dataset:

Link: Uploaded on github
The **training dataset** used was BBC news summary dataset having news articles and corresponding 'extractive' summary from 5 domains - business (510 articles), entertainment (386 articles), politics (417 articles), sport (511 articles), tech (401 articles) - in total 2225 articles. The dataset was divided into training (1800 articles) and validation (425 articles).
The **testing dataset** (all_data_combined_shuffled_3Dec_7Dec_aug19_dev) consisted of 579 articles. These were news articles as well containing fields like Title, content, Paragraphs sentiment, Whole sentiment etc

Library:

Allennlp (written in pytorch) was used to create the whole seq2seq pipeline.

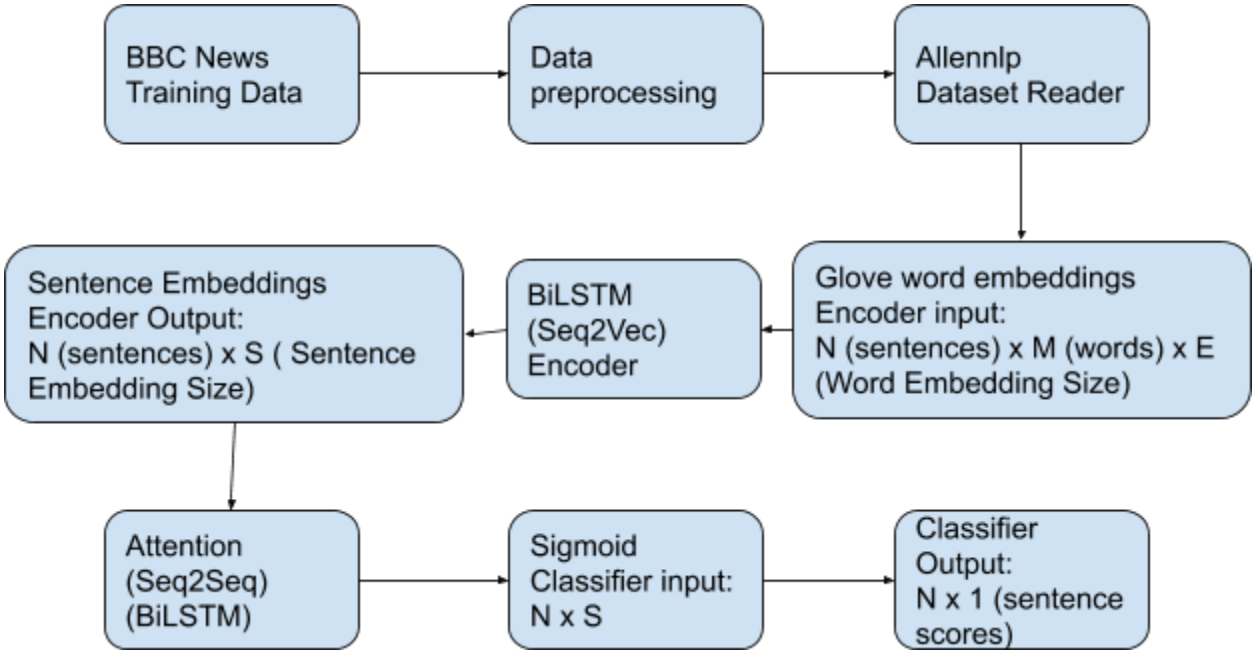
Data PreProcessing:

Each article was split up into separate sentences using nltk sentence segmenter. Then for each article a one hot vector was created with a length equal to the number of sentences in the article with 1 for the sentence index that is present in summary and 0 for others. A json dictionary is created with the list of sentences as 'content' and the one hot summary vector as 'relevant'. This dictionary is fed to the Allennlp reader. The reader converts content and relevant to corresponding listfields which are then fed into the model.

Model and Training:

Glove word embeddings were used of size 100d as the word embeddings. A bidirectional Seq2vec LSTM encoder with hidden size = 64 and number of hidden layers as 1 was used for getting the sentence embeddings. Each sentence was passed individually passed into the encoder. This encoder takes as input of size NxMx100 where N= number of sentences, M = number of words in that sentence and 100 is the word embedding size. This encoder returns a size of NxS as output where S is the sentence embedding size. This NxS is then fed into a Seq2Seq Bidirectional LSTM which acts as attention model. This attention model compares each sentence to the rest of the sentences in the article and updates the embeddings accordingly to minimise the loss. The loss is calculated against the one hot vector formed earlier where each index 1 corresponds to a sentence present in that summary. The output of this is finally fed into a 3-layer classifier with final layer as sigmoid that returns the corresponding scores of each of the sentence between 0-1 (which shows how likely it is to be included in the summary)

Here is a pictorial representation of model pipeline:



After training for 50 epochs, these were the statistics achieved:


```
 "best_epoch": 3,
  "peak_cpu_memory_MB": 2763.992,
  "peak_gpu_0_memory_MB": 801,
  "training_duration": "0:26:47.949685",
  "training_start_epoch": 0,
  "training_epochs": 49,
  "epoch": 49,
  "training_accuracy": 0.9478505762009581,
  "training_loss": 0.03650735903656025,
  "training_cpu_memory_MB": 2763.992,
  "training_gpu_0_memory_MB": 801,
  "validation_accuracy": 0.6151578546976036,
  "validation_loss": 0.36478194599310276,
  "best_validation_accuracy": 0.6506276150627615,
  "best_validation_loss": 0.21883400120668942
```


Testing and Evaluation:

As discussed earlier, the model was to be used to generate summaries for the dev data provided (all_data_combined_shuffled_3Dec_7Dec_aug19_dev) which consisted of 579 articles. These summaries were to be further used for evaluating sentiment. The summaries and the whole code and pretrained model is uploaded in the GitHub repository.