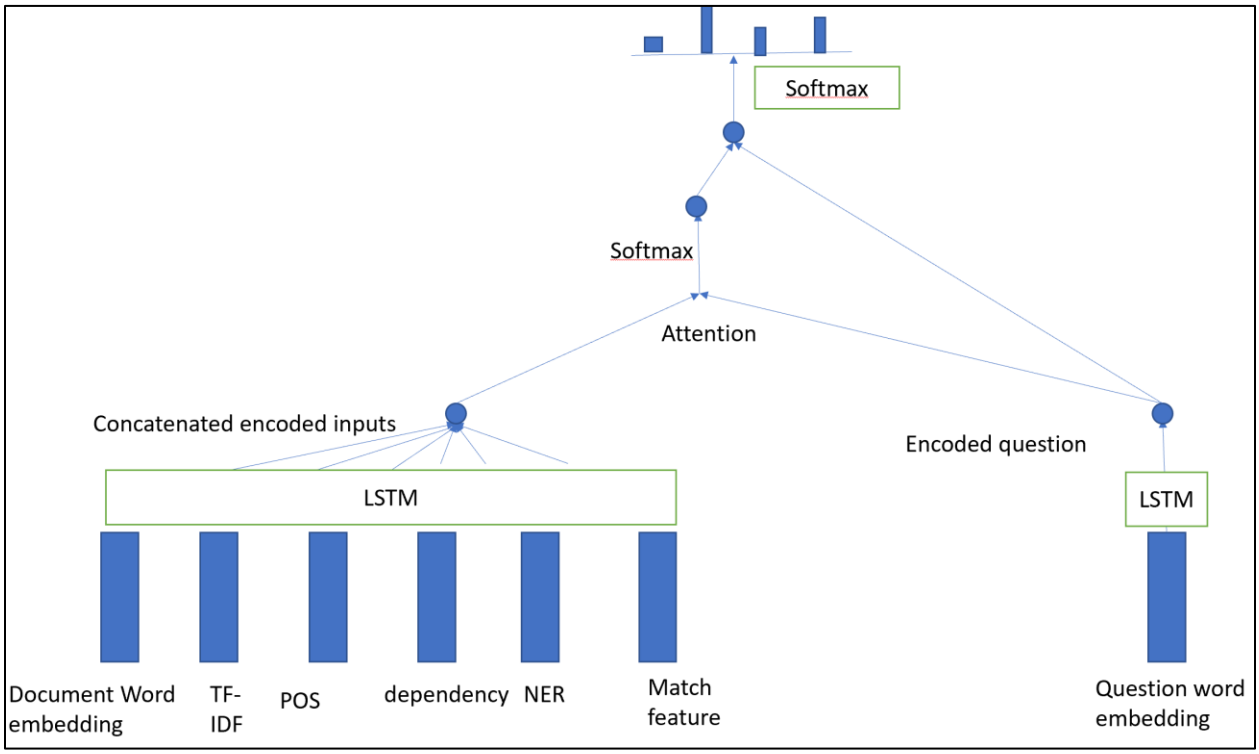
Assignment 2 report

Section A: Final Submission QA Framework

Figure 1: Overview of the architecture of the final model



1) Data Wrangling

In this section, we have pre-processed the input train and test data. First, we have converted the datasets to a list of dictionaries, where each dictionary has three keys 'document', 'question' and 'answer'.

Values of these dictionaries are a tokenized list.

e.g. first row of train dataset has been converted to below-

```
{'document': [['A', 'partly', 'submerged', 'glacier', 'cave', 'on', 'Perito', 'Moreno', 'Glacier', '.'], ['The', 'ice', 'fac
ade', 'is', 'approximately', '60', 'm', 'high'], ['Ice', 'formations', 'in', 'the', 'Titlis', 'glacier', 'cave'], ['A', 'gla
cier', 'cave', 'is', 'a', 'cave', 'formed', 'within', 'the', 'ice', 'of', 'a', 'glacier', '.'], ['Glacier', 'caves', 'are',
'often', 'called', 'ice', 'caves', ',', 'but', 'this', 'term', 'is', 'properly', 'used', 'to', 'describe', 'bedrock', 'cave
s', 'that', 'contain', 'year-round', 'ice', '.']], 'question': ['how', 'are', 'glacier', 'caves', 'formed', '?'], 'answer':
['A', 'glacier', 'cave', 'is', 'a', 'cave', 'formed', 'within', 'the', 'ice', 'of', 'a', 'glacier', '.']}
```

We have used TreebankWordTokenizer to tokenize the sentences.

2) Word Embeddings and Feature Extraction

1. **Word embedding** – We have used pre-trained gloVe model (glove.6B.100d.txt) to represent each word of the documents and questions as a vector of shape 100. In case of out of vocabulary error, word has been represented as an array of 100 zeros.

2. **TF-IDF** – We have used sklearn **TfidfVectorizer** to extract tf-idf value for each word in a document. In order to make TfidfVectorizer work on a tokenized list, we have created a customized tokenization function(dummy_fun).

3. **Match Feature**- We have compared each word of a document (after lemmatization and de-capitalization) with the words of the questions. All the words appearing in the questions have been marked as 1, whereas others have been 0.

4. **POS-Tagging** – We have used nltk.pos_tag() function on each tokenized sentence to extract the part-of-speech of the word.

5. **NER &**
6. **Dependency parsing**–We have used pre-trained model en_core_web_sm to extract named entities and dependencies from documents. This model expects a list of sentences and cannot work with already tokenized lists. So, first, we have re-constructed the sentences by joining the tokenized lists. Then, we created a customized tokenization function, so that the function uses the same tokenization method that we used in data wrangling part. After feature extraction I padded each document to make them up to the maximum document length present in train and test dataset combined.

   After all the feature extraction, we could represent the word 'Glacier' of first document sentence in train dataset with tf-idf value 0.2917469711981098, POS tagging 'NNP', named entity tagging 'ORG', dependency parsing 'pobj', word match feature 1.
   Then each value of feature was given an index and train and test datasets were built with the indexes of the feature values.
   e.g. token 'ice' in first document of train dataset, has been represented as [23,4,1,0,17,11]. Here, 23 is the index of 'ice' in dependency parsing vocabulary, 4 is the index of pos tagging vocabulary etc.

2) Sequence Model (LSTM with Attention)

I have done word-embedding of all the words in document and question.
I have also extracted 5 features from document to make our models stronger and have performed below steps sequentially to build the sequence model.

- Created inputs for word -embedding for document and questions.
- Created 5 inputs for 5 features of document words.
- Created sequential models with embedding layer of each feature with below properties-

i) Input dimension- feature vocabulary size
ii) Weights – list of values of the feature
iii) Output dimension – 100 for word embedding and 1 for other features

- Concatenated document features to create one composite input
- Put attention between encoded input and encoded question
- Created answer by concatenating permuted attention value and encoded question
- Added LSTM, regularization and dense layer on answer
- Created Model layer with all the inputs (document features + question features) and answer (word index of first and last token of each answer)
- Finally, I applied fit and predict method on the final model to get the predicted results on test dataset.

3) Output/Prediction

I achieved very less accuracy on test dataset mainly due to imbalance class problem. Here, in train dataset most of the answers are starting with 'the' and most of them are ending with ')'. That is the main reason our model performs bad to predict other answers not starting/ending with above mentioned tokens.

Section B: Results and Discussions

	Precision	Recall	F1-Score	Support
micro avg	0.47	0.47	0.47	1039
macro avg	0.04	0.06	0.04	1039
weighted avg	0.31	0.47	0.36	1039

Classification report for first token on train dataset

	Precision	Recall	F1-Score	Support
micro avg	0.04	0.04	0.04	680
macro avg	0.00	0.01	0.00	680
weighted avg	0.02	0.04	0.03	680

Classification report for first token on test dataset

	Precision	Recall	F1-Score	Support
micro avg	0.13	0.13	0.13	1039
macro avg	0.01	0.02	0.01	1039
weighted avg	0.03	0.13	0.05	1039

Classification report for last token on train dataset

	Precision	Recall	F1-Score	Support
micro avg	0.02	0.02	0.02	680
macro avg	0.00	0.00	0.00	680
weighted avg	0.00	0.02	0.00	680

Classification report for last token on test dataset

Table 1a: Classification report after using word embedding along with all the 5 extracted features from documents

	Precision	Recall	F1-Score	Support
micro avg	0.23	0.23	0.23	1039
macro avg	0.00	0.00	0.00	1039
weighted avg	0.05	0.23	0.08	1039

Classification report for first token on train dataset

	Precision	Recall	F1-Score	Support
micro avg	0.10	0.10	0.10	680
macro avg	0.00	0.01	0.00	680
weighted avg	0.01	0.10	0.02	680

Classification report for first token on test dataset

	Precision	Recall	F1-Score	Support
micro avg	0.07	0.07	0.07	1039
macro avg	0.00	0.00	0.00	1039
weighted avg	0.00	0.07	0.01	1039

Classification report for last token on train dataset

	Precision	Recall	F1-Score	Support
micro avg	0.02	0.02	0.02	680
macro avg	0.00	0.00	0.00	680
weighted avg	0.00	0.02	0.00	680

Classification report for last token on test dataset

Table 1b: Classification report after not using tf-idf feature

	Precision	Recall	F1-Score	Support
micro avg	0.23	0.23	0.23	1039
macro avg	0.00	0.00	0.00	1039
weighted avg	0.05	0.23	0.08	1039

Classification report for first token on train dataset

	Precision	Recall	F1-Score	Support
micro avg	0.10	0.10	0.10	680
macro avg	0.00	0.01	0.00	680
weighted avg	0.01	0.10	0.02	680

Classification report for first token on test dataset

	Precision	Recall	F1-Score	Support
micro avg	0.07	0.07	0.07	1039
macro avg	0.00	0.00	0.00	1039
weighted avg	0.00	0.07	0.01	1039

Classification report for last token on train dataset

	Precision	Recall	F1-Score	Support
micro avg	0.02	0.02	0.02	680
macro avg	0.00	0.00	0.00	680
weighted avg	0.00	0.02	0.00	680

Classification report for last token on test dataset

Table 1c: Classification report after not using dependency parsing

I tried other combinations as well, but they didn't provide better prediction.

Discussion - Explain your result in different aspect.

1. There must be some fault in model construction which results in poor prediction. Basically, this model does not learn anything from the extracted features. This model could give reasonable result if I could feed the whole sentence as the target labels.
2. In this architecture, I am feeding a whole document as an instance of train data. Model could perform better if I could educate the model to distinguish each sentence from the document.
3. 60% of the training document had no answers. I removed these instances from training dataset based on the assumption that model is not going to learn answer patterns from these instances. If I had included those instances in model, model would give 60% accuracy just by predicting no answer for every instance.
4. Few documents of test dataset were considerably longer than others. This made the maximum document length quite big (length = 1728). Removing these lengthy documents as outliers could make the model less complex.
5. If I could extract contextual word embedding instead of glove word embedding, this model could sense the context of the document better.
6. I didn't apply Bi-LSTM on document features and question features. Applying Bi-LSTM could provide better prediction.