NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Assignment for SC4002 / CE4045 / CZ4045

AY2023-2024

Group ID: G51

Group members:

Name	Matric No.
Karanam Akshit	U2020311E
Jameerul Kader Faizan	U2023863D
Louis Wirja	U2023825E
Sam Mun Kit, Jared	U2022434K
Clement Liew Zheng Qi	U2022031E

Question 1

1.1 Word Embeddings Similarity

<Based on word2vec embeddings you have downloaded, use cosine similarity to find the most similar word to each of these words: (a) “student”; (b) “Apple”; (c) “apple”. Report the most similar word and its cosine similarity. >

```
def most_similar_word(word):
    try:
        similar_words = pretrained_model.most_similar(word, topn=1)
        for similar_word, score in similar_words:
            print(f"{similar_word}: {score}")
    except KeyError:
        print(f"'{word}' is not in the vocabulary.")

print("Most similar word to 'student':")
most_similar_word('student')

print("\nMost similar word to 'Apple':")
most_similar_word('Apple')

print("\nMost similar word to 'apple':")
most_similar_word('apple')
```

```
Most similar word to 'student':
students: 0.7294867038726807

Most similar word to 'Apple':
Apple_AAPL: 0.7456986308097839

Most similar word to 'apple':
apples: 0.720359742641449
```

1.2 CoNLL2003 Dataset

(a) Describe the size (number of sentences) of the training, development and test file for CoNLL2003. Specify the complete set of all possible word labels based on the tagging scheme (IO, BIO, etc.) you chose.

The size (number of sentences) of the training, development, and test files for CoNLL2003 is as follows:

| Training: 14,987 |
| Development: 3,466 |
| Test: 3,684 |

The complete set of all possible word labels based on the BIO tagging scheme is as follows:
1. O (Outside): Represents words that are not part of any named entity.
2. B-PER (Beginning of a Person entity): Marks the first word of a named entity that is a person.
3. I-PER (Inside of a Person entity): Marks words inside a named entity that is a person.
4. B-LOC (Beginning of a Location entity): Marks the first word of a named entity that is a location.
5. I-LOC (Inside of a Location entity): Marks words inside a named entity that is a location.
6. B-ORG (Beginning of an Organization entity): Marks the first word of a named entity that is an organization.
7. I-ORG (Inside of an Organization entity): Marks words inside a named entity that is an organization.
8. B-MISC (Beginning of a Miscellaneous entity): Marks the first word of a named entity that is miscellaneous or doesn't fit into the other categories.
9. I-MISC (Inside of a Miscellaneous entity): Marks words inside a named entity that is miscellaneous.

(b) Choose an example sentence from the training set of CoNLL2003 that has at least two named entities with more than one word. Explain how to form complete named entities from the label for each word, and list all the named entities in this sentence.

Here is an example sentence from the training set of CoNLL2003 that has at least two named entities with more than one word:

The U.S. Federal Reserve is expected to raise interest rates next week.

To form complete named entities from the label for each word, we need to group together all of the words that have the same label and are contiguous. In this example, the named entities are:

U.S. Federal Reserve (ORG)
interest rates (MISC)

Therefore, the complete set of named entities in this sentence is:

U.S. Federal Reserve, interest rates

1.3 Model Description

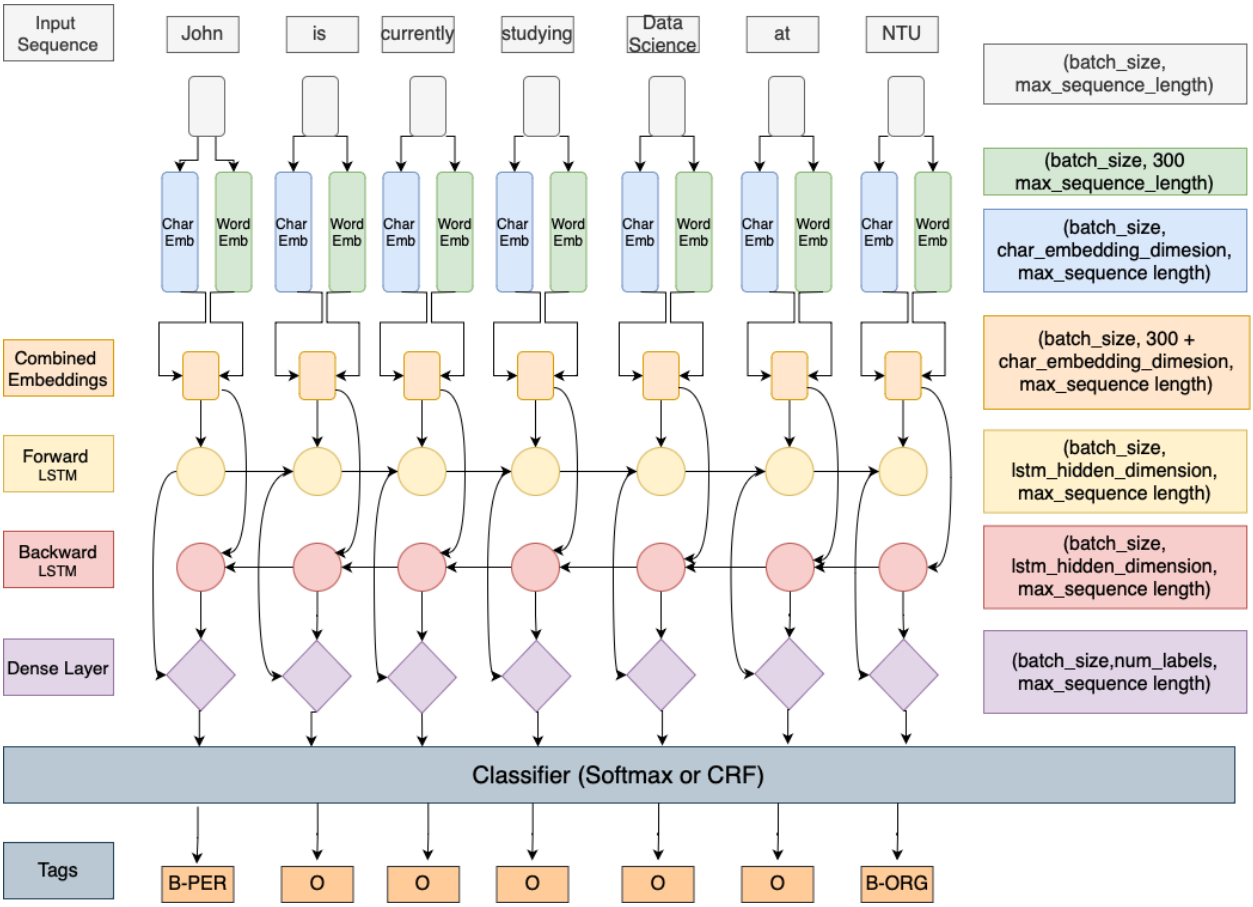


Figure 1.1 Overall Architecture for Named Entity Recognition

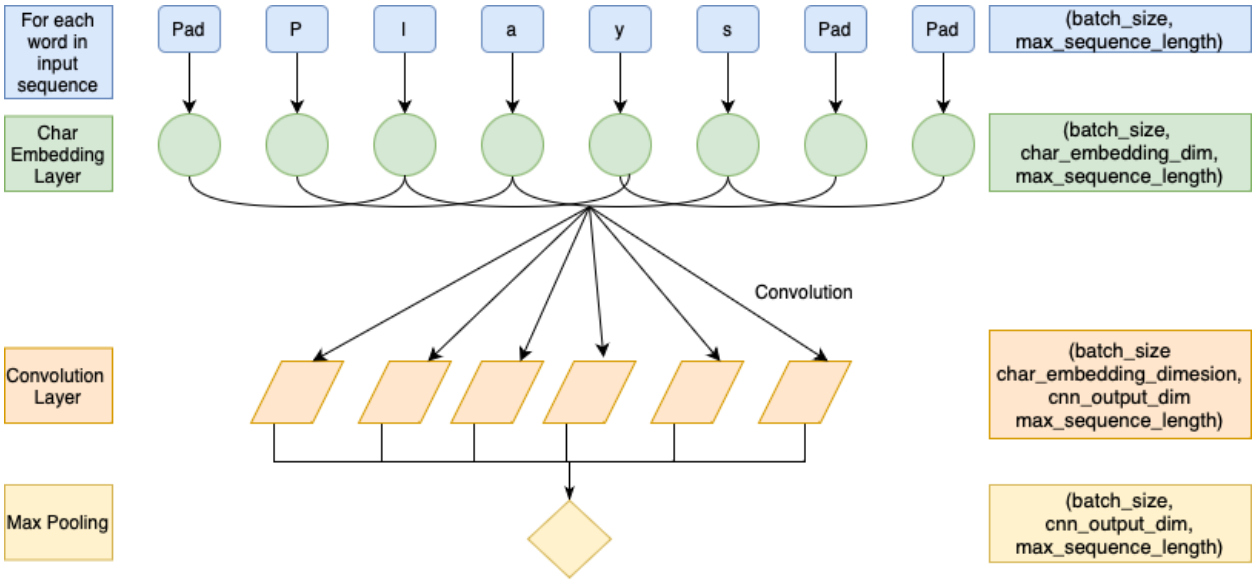


Figure 1.2 Character Embeddings using CNN

<Describe what neural network you used to produce the final vector representation of each word and what are the mathematical functions used for the forward computation (i.e., from the pretrained word vectors to the final label of each word). Give the detailed setting of the network including which parameters are being updated, what are their sizes, and what is the length of the final vector representation of each word to be fed to the softmax classifier. >

This architecture is inspired from the paper "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF" (Ma, 2013). Refer to Figures 1.1 and 1.2 for a better visual understanding.

1.3.1 Word Embeddings

Each word in the sequence is vectorised using 'word2vec-google-news-30' pre-trained Word2Vec model. If the sequence length is smaller than 'max_sequence_length', then a vector with 0s is applied.

Q1.3 (a) <Discuss how you deal with new words in the training set which are not found in the pretrained dictionary. Likewise, how do you deal with new words in the test set which are not found in either the pretrained dictionary or the training set? Show the corresponding code snippet.>

The same idea is also used for Out of Vocabulary Words. A zero vector is used as these embeddings are anyway Frozen (gradients for these are not updated), so even if a specific new embedding is created, it would not be any different from the zero vector.

This step is achieved while building the PyTorch Dataset class, and adding the [0] * 300 vector into the pretrainied_embeddings, and setting the index for OOV as the last item in the word_to_index. This is the code snippet (look at the highlighted portion). This is the same for the training/validation and test set.

```
word_to_index = pretrained_model.key_to_index
OOV_INDEX = 3000000
pretrained_embeddings =pretrained_model.vectors
pretrained_embeddings = np.append(pretrained_embeddings, [[0] *300], axis=0)
pretrained_embeddings = torch.FloatTensor(pretrained_embeddings)
class CoNLL2003Dataset(torch.utils.data.Dataset):
    def __init__(self, sentences_list, word_to_index, char_to_id, label_to_index, max_sequence_length, max_word_length):
        super(CoNLL2003Dataset, self).__init__()

        self.sentences = []  # List to store sentences
        self.labels = []     # List to store labels
        self.char_indices = []  # List to store character-level data

        for sent in sentences_list:
            words = []
            labels = []
            char_data = []  # Store character-level data
            for word in sent:
                words.append(word[0])
                labels.append(word[-1])
                # Convert word to character indices (rs)
                char_indices = [char_to_id.get(char,75) for char in word[0]]
                char_data.append(char_indices)
            words_index = [int(word_to_index.get(word, OOV_INDEX)) for word in words]
            labels_index = [int(label_to_index[label]) for label in labels]
```

1.3.2 Convolutional Neural Network (CNN) to obtain Character-Level Embeddings

Prior studies by Santos and Zadrozny (2014) and Chiu and Nichols (2015) have indicated the efficacy of employing CNN to extract morphological information, such as word prefixes or suffixes, from character representations and encode it into neural structures.

To get the character level embeddings, a dictionary is created with all the characters present in the training set including <PAD> and <OOV> tokens. The dictionary is then sorted by the frequency of appearance in the test set. The <OOV> token is added as there might be some characters in the test or validation set that are not present in the train set.

For each word in the sequence, the character's are extracted and are mapped to the index as stated to the dictionary. If the word length is smaller than the **max_word_length,** then the <PAD> tokens will be added on the left and right side. This vector representation is then inputted to the nn.Embedding Layer, with a dimension of **char_embedding_dim.** The gradients and parameters of this layer will be updated during backpropagation.

This character embedding layer is then fit into a convolutional layer with a kernal_size of **cnn_output_dimension** and with an appropriate **cnn_filter_size**. A max pool is done on the output of the convolutional layer. This now forms the character-level embedding.

1.3.3 Bi-Directional Long Term Short Term (Bi-LSTM) Network

The previously obtained Word Embeddings (from pre trained Word2Vec model) and character-level embeddings from the CNN, is concatenated and fed into a single Bi-LSTM layer with size **bilstm_hidden_size**.

For the mathematical functions that were used for Bi-LSTM, you can refer to Q2.

1.3.4 Linear Dense Layer

A linear transformation is conducted on the output of the previous Bi-LSTM layer, with a size of **num_labels.** Here the shape of the final layer before feeding to the sequence labelling layer (either Softmax or CRF) is **(batch_size, max_sequence_length, num_labels).** No activation function has been used here.

1.3.5 Conditional Random Fields (CRF) Layer

In sequence labeling tasks, particularly those involving structured prediction, it is beneficial to consider label correlations in neighboring positions. For example, in POS tagging, certain label sequences are more likely based on linguistic patterns. Similarly, in NER with BIO annotation, certain label sequences are constrained. To address this, a conditional random field (CRF) is employed to jointly model label sequences, allowing for a more context-aware decoding approach instead of decoding each label independently. Using CRF gave a better F1 Score as compared to using Softmax.

This is how CRF works:

Input: $z = \{z1, \cdots, zn\}$

Output: $y = \{y1, \cdots, yn\}$

Y(z) denotes the set of possible label sequences for z. The probabilistic model for sequence CRD defines a family of conditional $p(\boldsymbol{y}|\mathbf{z}; \mathbf{W}, \mathbf{b})$ probability over all possible label sequences y given z in the following way:

$$p(\boldsymbol{y}|\mathbf{z}; \mathbf{W}, \mathbf{b}) = \frac{\prod_{i=1}^{n} \psi_i(y_{i-1}, y_i, \mathbf{z})}{\sum_{y' \in \mathcal{Y}(\mathbf{z})} \prod_{i=1}^{n} \psi_i(y'_{i-1}, y'_i, \mathbf{z})}$$

Where $\psi_i(y', y, \mathbf{z}) = \exp(\mathbf{W}_{y',y}^T \mathbf{z}_i + \mathbf{b}_{y',y})$ are potential functions $\mathbf{W}_{y',y}^T$ and $\mathbf{b}_{y',y}$ and are the weight vector and bias corresponding to to label pair *(y',y)*

For CRF Training, a maximum likelihood estimation can be used. This loss function is used:

$$L(\mathbf{W}, \mathbf{b}) = \sum_i \log p(\boldsymbol{y}|\mathbf{z}; \mathbf{W}, \mathbf{b})$$

Maximum likelihood training chooses parameters such that the log-likelihood L(**W, b**) is maximized. Decoding is to search for the label sequence **y*** with the highest conditional probability:

$$\boldsymbol{y}^* = \underset{y \in \mathcal{Y}(\mathbf{z})}{\operatorname{argmax}}\, p(\boldsymbol{y}|\mathbf{z}; \mathbf{W}, \mathbf{b})$$

The decoding can be solved efficiently during training and evaluation using the Viterbi Algorithm.

1.3.6 Model Setup

The model was implemented on PyTorch, and the CRF layer was implemented with the help of Pytorch-CRF library. During the training, the maximum likelihood function as described above was used as the loss function and the Adam Optimizer was used for optimizing the loss function. After which (W,B) for each layer are updated as defined by the learning rate. We monitor the validation loss, and save the model that has the least validation loss.

1.3.7 Hyperparameter Tuning

(c) Report how many epochs you used for training, as well as the running time.

Hyperparameter Tuning:

(i) Changing the batch_size:

Batch_size	Val Loss	Val F1
16	15.9376	0.9164
64	54.9910	0.9056
128	106.8120	0.9018

(ii) Changing the hidden dimension for the BiLSTM layer

Dimension Size	Val Loss	Val F1
512	56.6263	0.8995
256	55.0818	0.9009
128	52.9519	0.8981

1.3.8 Final Model Hyperparameters

For Model Definition:

max_sequence_length	100
max_word_length	15
char_embedding_dim	100
cnn_output_dim	64
cnn_filter_size	3
bilstm_hidden_size	512
num_labels	9

For Training:

epochs	10
learning_rate	0.001
batch_size	16

<Report the f1 score on the test set, as well as the f1 score on the development set for each epoch during training.>

1.3.8 Results

Final Model Training

Epoch	Val F1	Val Loss
1	0.8595	18.6494
2	0.8776	14.9281
3	0.9033	13.0431
4	0.8990	13.0791
5	0.9131	12.6203
6	0.9128	12.5246
7	0.9124	12.9516
8	0.9071	15.1481
9	0.9049	15.6012
10	0.9080	17.0144

Time taken to train: 397.2787s

Results

Model	Test F1	Val F1
Single Linear Layer	0.645	0.698
Bi-LSTM	0.820	0.877
CNN-BiLSTM	0.857	0.905
CNN-BiLSTM-CRF	0.864	0.913

As seen from the results, CNN-BiLSTM-CRF gave the best performance with a test F1 score of 0.864.

Question 2

2.1 Labels

After converting to the new setting, we have arbitrarily chosen the labels ‘0’, ‘1’, ‘2’, ‘3’ from the coarse labels, and combined the subsequent labels ‘4’ and ‘5’ as ‘OTHERS’.

2.2 Aggregation methods

In our experiments, we have tried max pooling, mean pooling, min pooling, and last word representation with a BiLSTM. This section discusses the aggregation methods used, as well as a comparison for the methods to obtain the best performing method.

2.2.1 Max Pooling

A max pooling approach takes the maximum value across the LSTM outputs to create a fixed length output. This method captures the most significant outputs from the LSTM.

2.2.2 Mean Pooling

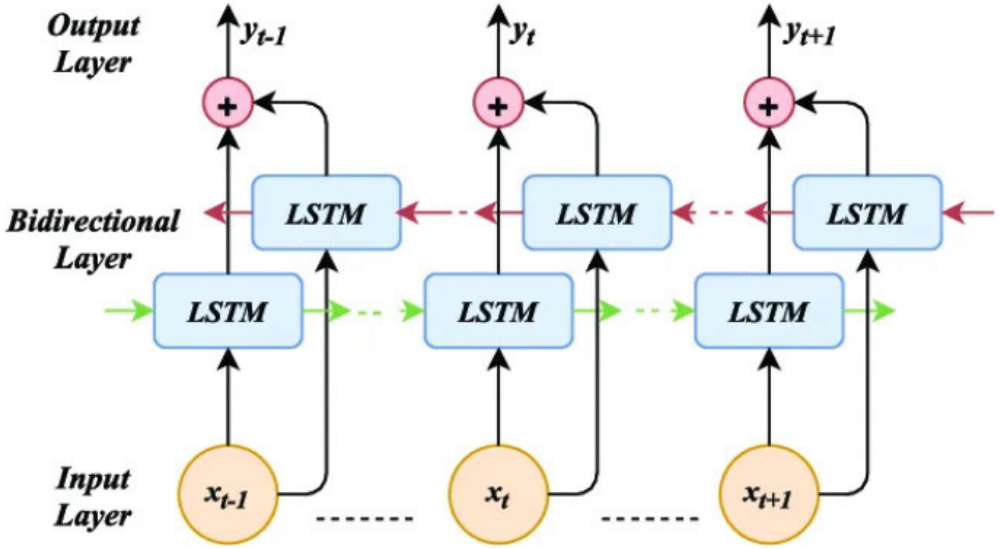
As opposed to max pooling where the maximum value of the outputs is taken, mean pooling involves taking the mean or average of the embeddings at each time step, which ensures that all parts of the sequence contribute equally to the final feature representation.

2.2.3 Min Pooling

A min pooling approach takes the minimum value across the LSTM outputs to create a fixed length output. This method reduces the information from the outputs of the LSTM.

2.2.4 Last Word Representation

Since we are using a BiLSTM model, we will apply a concatenation of the two final hidden states. One hidden state is obtained from processing the sequence from start to end, and the other is obtained from processing end to start. Concatenating these two states provides a representation that captures information from both the beginning and the end of the sequence.



2.2.4 Comparison of aggregation methods

Aggregation Method	Last Word Representation	Average Pooling	Max Pooling	Min Pooling
Test Accuracy/(%)	93.6	92.4	91.4	93.0
Runtime-run across the same cpu/gpu/ (seconds)	76.20	155.92	123.11	99.56

We obtained the following test accuracies across the different aggregation methods, with last word representation having the best test accuracy of 93.6% and also the fastest runtime as well.

2.3 Neural network

As mentioned previously, the BiLSTM model will be used to produce the final vector representation of each word.

Each word in the input text is represented by a 300-dimensional pre-trained word vector from the ‘word2vec-google-news-300’ model. The sequences are then packed for processing.

The LSTM cells within the BiLSTM model consist of three gates: forget, input and output. The forget gate decides whether to keep or delete the current information, the input gate determines how much new information will be added to the memory, and the output gate determines whether

the current value in the cell contributes to the output. The sigmoid and tanh functions are shown below for reference.

$$\sigma(x) = 1/(1 + e^{-x})$$

$$tanh(x) = \left(e^{x} - e^{-x}\right)/\left(e^{x} + e^{-x}\right)$$

Forget Gate

A sigmoid function is used for this gate to make the decision of what information needs to be removed from the model's memory. This choice is basically determined by the values of h_{t-1} and x_t. The gate's output, f_t, is a number between 0 and 1 where 1 denotes that the learned value has been fully eliminated. This output is computed by:

$$f_t = \sigma(W_{f_h}[h_{t-1}], W_{f_x}[x_t], b_f)$$

Input Gate

The decision regarding the addition of new data to the LSTM memory is made by this gate. The sigmoid and tanh layer make up this gate. Firstly, the sigmoid layer determines which values need to be changed, and the tanh layer creates a vector of new values to be added to the memory. These two layers are computed by:

$$i_t = \sigma(W_{i_h}[h_{t-1}], W_{i_x}[x_t], b_i)$$

$$c_t = tanh(W_{c_h}[h_{t-1}], W_{c_x}[x_t], b_c)$$

The combination of these two layers allows the LSTM to update where the current value from the forget gate output is multiplied by the old value and then subsequently added with the new candidate value (i.e. $i_t * c_t$). The formula is given below:

$$c_t = f_t * c_{t-1} + i_t * c_t$$

Output Gate

The output is computed at this gate by first utilizing a sigmoid layer to determine which part of the stored memory can contribute to the output. A tanh function is performed to map the values, which is followed with the multiplication to the output from the sigmoid layer to compute the output of the LSTM.

$$o_t = \sigma(W_{o_h}[h_{t-1}], W_{o_x}[x_t], b_o)$$

$$h_t = o_t * tanh(c_t)$$

Concatenated Hidden State

After passing through the BiLSTM model, the packed sequence is transformed back to the tensor form of sequences, and the final hidden state is generated by concatenating the last forward and backward states to form the final feature representation of the sequence. This final hidden state is a vector of length 2*hidden dimensions, where the value of hidden dimensions is 100, resulting in the length of each sequence to be 200.

Linear Layer and Softmax Classifier

The concatenated hidden state is then passed through a fully connected linear layer, which transforms it to the output dimension of the classification task. The size of the final output is 5, which corresponds to the number of labels being classified.

Network Parameters

The weights and biases of the BiLSTM (denoted by W and b) are the parameters being updated during training. These parameters are optimized to minimize the cross entropy loss between the predicted probabilities and the true class labels.

2.4 Epochs and running time

Training was set to run with 20 epochs. Early stopping was implemented to monitor the validation accuracy on the development set. A patience of 5 was set, where training would stop if the validation accuracy did not improve after 5 epochs. The running time reported is 17.77 seconds.

```
Running time: 17.77 seconds
```

2.5 Results

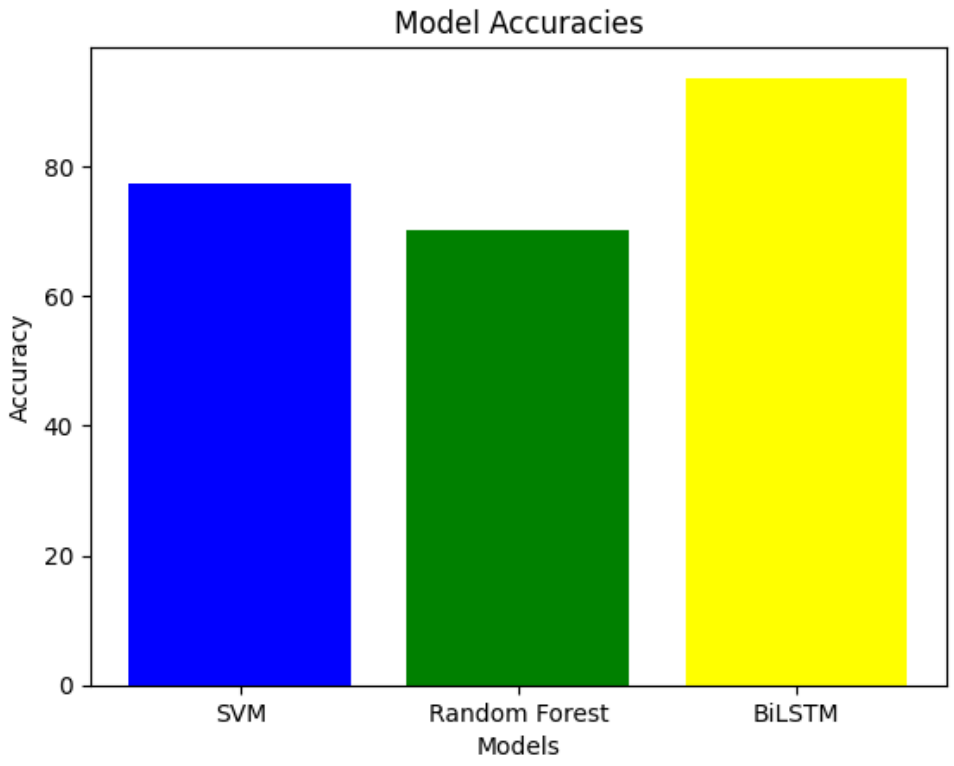
Training the BiLSTM model with last word representation took 9 epochs until accuracy stopped increasing. The accuracies on the development set for each epoch can be seen in the image below. The accuracy on the final epoch before early stopping was 91.6%.

```
Epoch 1/20 - Validation Loss: 0.3822 - Accuracy: 85.60%
Epoch 2/20 - Validation Loss: 0.3049 - Accuracy: 88.20%
Epoch 3/20 - Validation Loss: 0.3200 - Accuracy: 90.00%
Epoch 4/20 - Validation Loss: 0.2869 - Accuracy: 92.40%
Epoch 5/20 - Validation Loss: 0.3171 - Accuracy: 91.40%
Epoch 6/20 - Validation Loss: 0.3389 - Accuracy: 91.60%
Epoch 7/20 - Validation Loss: 0.3383 - Accuracy: 91.60%
Epoch 8/20 - Validation Loss: 0.3577 - Accuracy: 91.40%
Epoch 9/20 - Validation Loss: 0.3668 - Accuracy: 91.60%
Training stopped early at epoch 9. Accuracy on Development set did not improve for 5 epochs.
```

The accuracy obtained on the test set was 93.6%.

```
Test Loss: 0.3180 - Test Accuracy: 93.60%
```

The accuracies obtained from the BiLSTM model outperforms the baseline SVM and Random Forest classifiers by quite a large margin, and it proves to be able to classify sentence level questions well.



References

Q1

Xuezhe Ma and Eduard Hovy. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany. Association for Computational Linguistics.