NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Assignment for SC4002 / CE4045 / CZ4045

AY2023-2024

Group ID: 38

Group members:

Name	Matric No.	Contributions
Gucon Nailah Ginylle Pabilonia	U2021643H	Part 1
Nigel Tan Yong	U2021165A	Part 1
Nur Dilah Binte Zaini	U2022478K	Part 1
Yap Xuan Ying	U2021297G	Part 2
Chong Yue Hong	U2120181D	Part 2

Table of Content

Part 1. Sequence Tagging: NER

This section shows our findings and answers to Question 1.

1.1 Word Embedding

Question 1.1

Based on word2vec embeddings, the most similar words to each of these words: (a) “student”; (b) “Apple”; (c) “apple” using the cosine similarity are shown in TABLE 1.

TABLE 1: BIOES Tagging Scheme

Part	Word	Most Similar Word	Cosine Similarity
a	student	students	0.73
b	Apple	Apple_AAPL	0.75
c	apple	apples	0.72

1.2 Data

Question 1.2

Part (a)

The size of the dataset is as shown in TABLE 2.

Chosen Tagging Scheme: **BIOES**

TABLE 2: Size of Datasets

Set Name	Number of Sentences (Set Size)
Train	14041
Development	3250
Test	3453

Complete set of all possible word labels based on the BIOES tagging scheme:

`S-ORG, O, S-MISC, B-PER, E-PER, S-LOC, B-ORG, E-ORG, I-PER, S-PER, B-MISC, I-MISC, E-MISC, I-ORG, B-LOC, E-LOC, I-LOC`

Part (b)

The sentence we chose is as shown in TABLE 3.

TABLE 3: Chosen Sentence

Sentence	*He has spent his adult life in politics, coming to Congress as an aide in 0000 after three years in the Air Force and then being elected to the House of Representatives himself in 0000.*

We made use of the BIOES tagging scheme as it gives a more accurate representation when differentiating multi-word and single-word entities as compared to BIO. With the End tag “E”, BIOES tagging scheme could help with identifying the last word of a name entity. The BIOES tagging scheme is as shown in TABLE 4.

TABLE 4: BIOES Tagging Scheme

O	O	O	O	O	O	O	O
He	has	spent	his	adult	life	in	politics
O	O	O	S-ORG	O	O	O	O
,	coming	to	Congress	as	an	aide	in
O	O	O	O	O	O	B-ORG	E-ORG
0000	after	three	years	in	the	Air	Force
O	O	O	O	O	O	B-ORG	I-ORG
and	then	being	elected	to	the	House	of
E-ORG	O	O	O	O			
Representatives	himself	in	0000	.			

Using the BIOES labels, we identified the named entities with more than one word as shown in TABLE 5.

TABLE 5: Named Entities

Air Force	B-ORG is assigned to “Air” as it represents the beginning of an organisation phrase while E-ORG is assigned to “Force” as it represents the end of the organisation phrase, therefore, a complete named entity is formed.
House of Representatives	B-ORG is assigned to “House” as it represents the beginning of an organisation phrase, I-ORG is assigned to and E-ORG is assigned to “Force” as it represents the end of the organisation phrase, therefore, a complete named entity is formed.

1.3 Model

Question 1.3

Part (a)

Conversion of Dataset Format

With reference to Question 1.2, the training, development and test files used in Question 1 are from CoNLL2003 which are converted into an array of sentences. The array of sentences generated are train_sentences, dev_sentences and test_sentences. This allows us to perform data cleaning in a more convenient manner.

Dealing with unknown words in training dataset

The code snippet for the creation of the pretrained dictionary is shown in Figure 1.

```
def mappingOfWord(sentences, lower):
    # Standardise the word format by converting words to lower case
    words = [[x[0].lower() if lower else x[0] for x in s] for s in sentences]

    # Define pretrained dictionary
    dictionary = creationOfDictionary(words)

    # Assign UNK tag for unknown words
    dictionary['<UNK>'] = 10000000

    # word_to_id -> a unique id which is the value starting from 0 is given to each unique word which is the key
    # id_to_word -> a unique id which is the key starting from 0 is given to each unique word which is the value
    word_to_id, id_to_word = mappingCreation(dictionary)
    print("Total unique words (%i in total): %i" % (sum(len(x) for x in words), len(dictionary)))

    return dictionary, word_to_id, id_to_word
```

Figure 1: Code snippet of creation of pretrained dictionary

Each sentence in the train_sentences is converted to the lowercase to ensure consistency and standardisation. Once all the word forms are standardised, the occurrence of each unique word in train_sentence is counted and stored in a dictionary as the value and the unique word as the key. A <UNK> key is created in the dictionary which has the value of 10000000. The dictionary generated is used as the pretrained dictionary. The words in the train_sentences are then compared to the pretrained dictionary. The words which are not found in the pretrained dictionary will be considered to be under the <UNK> key.

Dealing with unknown words in test dataset

To deal with unknown words, we used the code snippet in Figure 2.

```
for sentence in sentences:
    # Get all the words in the sentence
    str_words = [word[0] for word in sentence]

    # Convert all the words to lower case
    # Unknown words will be labelled as <UNK>
    words = [word_to_id[lower_case(word,lower) if lower_case(word,lower) in word_to_id else '<UNK>'] for word in str_words]
```

Figure 2: Code snippet of determining the known and unknown words

The train_sentences are retrieved from the training file while test_sentences are retrieved from the test file. The pretrained dictionary as mentioned earlier is used to compare the words in test_sentences. The words that are not found in either the train_sentences which is the training dataset or pretrained dictionary are considered as <UNK>.

Part (b)

The Bidirectional Long Short-Term Memory (BiLSTM) recurrent neural network was used to produce the final vector representation of each word. In general, the word embedding obtained for the individual words are concatenated with character-level representations. They then go through the BiLSTM layer to output the final vector representation for each word.

According to what we have learnt in lecture T3 "Sequence", the mathematical functions used for forward computation from the pre-trained word vectors to the final label of each word are shown in TABLE 6.

TABLE 6: Mathematical functions for bidirectional LSTM

No.	Formula	Description
1	$f^t = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)$	Forget gate
2	$i^t = \sigma(W_i h^{(t-1)} + U_i x^{(t)} + b_i)$	Input gate
3	$o^t = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o)$	Output gate
4	$\sim c^{(t)} = tanh(W_c h^{(t-1)} + U_c x^{(t)} + b_c)$	New cell content
5	$c^{(t)} = f^t * c^{(t-1)} + i^{(t)} * \sim c^{(t)}$	Cell state
6	$h^{(t)} = o^t * tanhc^{(t)} + i^{(t)}$	Hidden state

TABLE 7 illustrates the neural network used (BiLSTM) to produce the final vector representation of each word, the parameters and the length of the final vector representation of each word to be fed to the softmax classifier.

TABLE 7: Detailed settings of the bidirectional LSTM

Settings	Description
Network architecture	Bidirectional Long Short-Term Memory (BiLSTM)
Parameters with updated size	Word embedding weights: 300
	LSTM weights: 300 + 25
	Hidden dimension: 128
	LSTM biases: 2 * 128
	Linear layer weights: 2 * 128
	Linear layer biases: 19
	CRF transition matrix: [19, 19]
	Final vector representation: 19

Part (c)

The number of epochs used for training: 10 epochs

The running time to execute all the epochs is 29933.176 seconds. TABLE 8 shows the running time for each epoch.

TABLE 8: Running time (in seconds) for each epoch

Epoch Number	Running Time (seconds)
1	3289.468
2	2918.741
3	2921.706
4	2917.424
5	3036.318
6	3154.044
7	2912.550
8	2909.196
9	2908.208
10	2965.520

Given the model's training duration exceeding 8 hours, we've included a link here to the trained model for your convenience.

Part (d)

The F1 scores on the test set and development set for epoch during training is shown in TABLE 9.

TABLE 9: F1-score of Test Set and Dev Set for each epoch

Epoch Number	F1 Score of Test Set	F1 Score of Dev Set
1	0.00324	0.00471
2	0.00212	0.00353
3	0.00257	0.00406
4	0.00242	0.00342
5	0.00362	0.00472
6	0.00243	0.00427
7	0.00230	0.00319
8	0.00285	0.00338
9	0.00229	0.00317
10	0.00219	0.00393

Part 2. Sentence-Level Categorization: Question Classification

This section shows our findings and answers to Question 2.

Question 2

Part (a)

The 5 classes used after converting from the original label set to the new setting:

Label 0, Label 1, Label 2, Label 3, Label 4

We merge label-coarse 2 & 5 into 4 (OTHERS)

Part (b)

1. "Last Time Step" aggregation

Since we used Long Short Term Memory (LSTM) model to perform the sentence classification task, we tried aggregating by selecting the hidden state at the last time step.

The [:, -1, :] notation is used to slice the tensor. It means:

- :: Select all instances in the batch.
- -1: Select the hidden state at the last time step.
- :: Select all elements in the hidden dimension.

Thus, x[:, -1, :] is used to select the last time step of LSTM layer output and passed to the next dropout layer.

2. "Mean Pooling" aggregation

We also tried aggregating by computing the average hidden state across all time steps in the input sequence using torch.mean(x, dim=1).

TABLE 10: Cross-validation accuracy results

Cross-Validation Accuracy	Last Time Step	Mean Pooling
Fold 1	59.70%	87.47%
Fold 2	34.85%	85.59%
Fold 3	83.92%	87.17%
Fold 4	86.25%	87.64%
Fold 5	76.67%	89.31%
Mean	68.28%	87.44%

We tested the aggregation methods with 5 fold cross-validation to ensure a more robust and unbiased assessment of each method's effectiveness in improving model accuracy and generalizability. As shown in TABLE 10, “Mean Pooling” has a higher mean cross-validation accuracy (87.44%) across 5 folds as compared to 68.28% from “Last Time Step”. Hence, we adopt “Mean Pooling” as our aggregation method.

Part (c)

TABLE 11 illustrates the neural network used (BiLSTM) to produce the final vector representation of each word, the parameters and the length of the final vector representation of each word to be fed to the softmax classifier.

TABLE 11: Detailed settings of the bidirectional LSTM

Settings	Description
Network architecture	Bidirectional Long Short-Term Memory (BiLSTM)
Parameters with updated size	Word embedding weights: 300
	LSTM weights: 300
	Hidden dimension: 64
	LSTM biases: 2 * 64
	Linear layer weights: 2 * 64 * 5
	Linear layer biases: 5
	Final vector representation: 5

Parameter Update:

- Since we are using the pre-trained word embeddings, the parameters of the Embedding Layer are frozen (freeze=True) to keep them fixed.
- The parameters of the LSTM, Fully Connected, Softmax, and Dropout layers are learned during training and are updated through backpropagation.

TABLE 12 shows the mathematical functions used for the forward computation.

TABLE 12: Mathematical functions for bidirectional LSTM

No.	Formula	Description
1	$f^t = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)$	Forget gate
2	$i^t = \sigma(W_i h^{(t-1)} + U_i x^{(t)} + b_i)$	Input gate
3	$o^t = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o)$	Output gate
4	$\sim c^{(t)} = tanh(W_c h^{(t-1)} + U_c x^{(t)} + b_c)$	New cell content
5	$c^{(t)} = f^t * c^{(t-1)} + i^{(t)} * \sim c^{(t)}$	Cell state
6	$h^{(t)} = o^t * tanhc^{(t)} + i^{(t)}$	Hidden state

Part (d)

We set 100 epochs for training and early stopping is implemented with a patience of 3 (If the validation accuracy isn't increasing for three epochs). The training stops at **Epoch 19**.

The running time is **38 seconds**.

Part (e)

Test Accuracy = 92.20%

TABLE 13: Validation accuracies for each epoch

Epoch	Validation accuracy (Development Set)
1	42.40%
2	51.00%
3	67.20%
4	69.00%
5	83.80%
6	86.40%
7	85.00%
8	86.80%
9	88.40%
10	88.80%
11	90.40%
12	90.20%
13	90.40%
14	90.40%
15	90.00%
16	91.20%
17	89.80%
18	90.20%
19	90.20%