Question Classifier Using BoW and BiLSTM

Bushui Zhang (UID: 10407579), Yazhuo Cao (UID: 10329221),
Yecheng Chu (UID: 10319044), Zhaoyu Zhang (UID:10838545),
Zhengqian Jin (UID:10839527)

1 Introduction

Question Answer System(QAS) is a principal area of Natural Language Processing(NLP) and the ability to accurately classify question types is crucial as it improves the effectiveness and efficiency of QAS such as search engines.

This is an interesting but challenging area of NLP as the information involved has a high volume, variety and a cross-domain solution is needed.

In this paper, the effect of using random and pre-trained word embedding methods on both Bag of words(BoW) and BiLSTM sentence representation models have on the performance of question type classification are investigated.

2 Methodologies

In order to feed data to the neural network for question classification, we first need to convert input sentences into network processable vectors. This can be done with three main steps, data preprocessing, word embedding and sentence representation. Among them, sentence representation plays the most significant role. In this section, the BoW model and BiLSTM networks that were used for this purpose in this paper will be described below, the remaining steps will be discussed in more detail in section 3.1.

2.1 BoW Model

The Bag of Words (BoW) model is used to extract features from a set of text and returns a vector representing the occurrence of words. As the words are in a "bag", the order and structure of the words are discarded. The vector representing a sentence in this case is simply the average of vectors corresponding to its words as shown in Figure 1 and in the equation 2.1 below where *s* is the sentence that needs to be classified, *w* is the word in sentence *s* and | *bow(s)* | is the number of words in the sentence *s*.

$$vec_{bow}(s) = \frac{\sum_{w \in bow(s)} vec(w)}{|bow(s)|} \quad (1)$$

2.2 BiLSTM Model

One of the most popular types of neural networks used in natural language processing is the recurrent neural network (RNN). It takes a statistical machine learning approach to model sequential data and allows the words to affect all that came after it. Long Short-Term Memory (LSTM) is an architecture of RNN that considers order dependency in a sentence as it is capable of storing and using vector representation from previous words in the sentence.

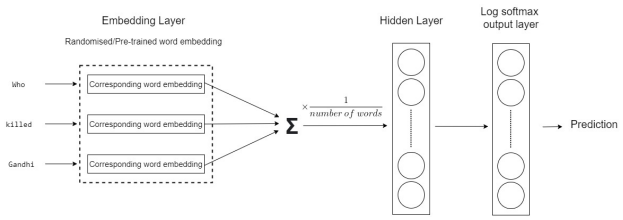

Figure 1: A diagram representing the BoW Model implemented in this paper.

The Bidirectional LSTM (BiLSTM) network is a type of BiRNN where the signal can propagate backward as well as forward in time. It is a variant of LSTM networks where LSTM networks are set up bidirectionally, one forward and one backwards to allow signals to travel in both directions. The BiLSTM layers used in this paper pass the data through a fully connected linear layer, followed by a log softmax function to obtain the probability of the prediction of each class as shown in Figure 2.

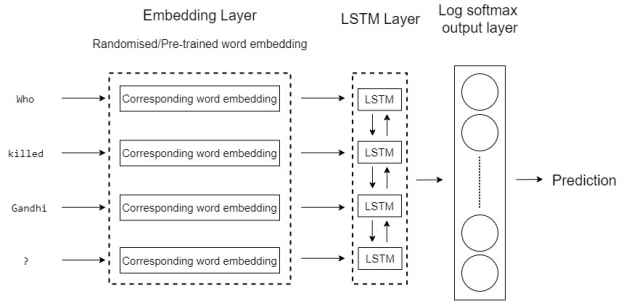

Figure 2: A diagram representing the BiLSTM Model implemented in this paper.

2.3 Ensemble Model

Based on two different models above, we also ensemble them into one model which combines the outputs from their output layers during the testing procedure, and then

the ensemble version will predict the labels from the combined scores according to the same principle described above.

3 Experiments

3.1 Pre-processing

This is an important step of classification as mentioned in section 2 as it improves both the speed and the performance of the model.

3.1.1 Data Split

The data used in this paper is the Training set 5 from existing experimental data for question classification(Li & Roth, 2002). In order to separate data used for training and parameter fine-tuning, the data are randomly divided into 10 parts where nine are used as a training set and one is used as a development set to help verify the performance of the model designed and adjust their hyper-parameters. The data used for testing is a separate dataset from the same source (Li & Roth, 2002) that has been used for testing only.

3.1.2 Stopword Removal

The data contain tuples of feature, the question string and label, the question type. The question string is tokenised and only meaningful words are kept for further processing. The words that will be removed are the stopwords which are too common, punctuation and some abbreviations where they do not contribute to the classification of problem type. This improves the training accuracy and speed as it reduces the amount of data the model processes for each epoch.

3.1.3 Data Lowercasing

The next step is to unify the letter case in the words by lowercasing. This is the final step before sentence vectorisation to ensure token consistency and avoid duplicate definitions caused by different letter casing of the same word.

3.2 Sentence Vectorisation

To achieve sentence vectorisation, we created a *Vocabulary* object for each sentence in the dataset.

Hence the sentence vectorisation implementation was divided under 2 different circumstances, which was shown in Figure 3:

3.2.1 Randomly Initialised word embeddings

1. Initialise a *Vocabulary* object, remove stopwords from the sentences in the training dataset
2. Count the number of appearances of each word and store them into *word2count*
3. Sort the dictionary and filter out rare words with an occurrence below a threshold.
4. Use the remaining word to build a *word2ind* dictionary
5. Build a *torch.nn.Embedding* object using the total number of words and embedding dimension
6. Detach the embeddings to numpy array, followed by converting into *Tensor* object and stored in *word2vec* dictionary.

Kocmi and Bojar (Kocmi & Bojar, 2017) established in 2017 that different methods of random initialisation do not present significant differences in the performance of the model if the variance is reasonably low. High variance on the other hand prevents the network from using the embedding space to perform the task but instead compels it to use other free parameters.

3.2.2 Pre-trained word embeddings

1. Read the pre-trained word embedding vectors, GloVe, into a dictionary, where each pair is of (token, vectors)
2. Use the embeddings to build a *word2ind* dictionary
3. replace *word2vec* with the pre-trained word embeddings, convert each line into a *Tensor* object, store all Tensors into a list and turn it into *FloatTensor*
4. Use pytorch's builtin function *torch.nn.Embedding.from_pretrained*, create word embeddings in a *Vocabulary* object, while has a option of *freeze* the fine-tune during training.

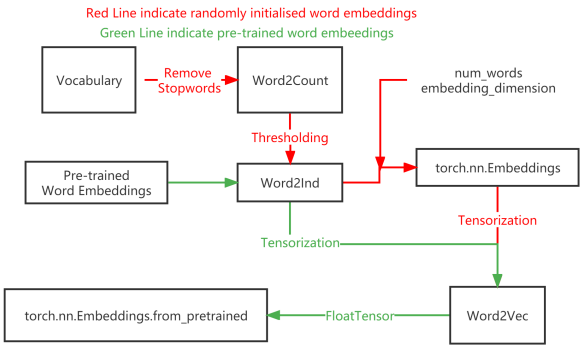

Figure 3: Flowchart of create randomly initialised and pre-trained word embeddings

3.3 Hyper-parameters

With the goal of getting the best values for the various hyper-parameters in the experiment and therefore the best performance, hyper-parameter tuning was performed using the training set and development set. After the two models were created, the development set was used instead of the test set to obtain the performance of the

model's question type prediction, which was shown in the form of visualisation in the following sections.

The number of neurons used in the hidden layer of the BoW model was set to be 2/3 of the word embedding dimension, which was generated from Dr. Heaton's experiments (Heaton, 2008).

3.3.1 Evaluation Metrics

We chose to import skicit-learn's package

$$sklearn.metrics.f1\_score \quad (2)$$

$$sklearn.metrics.accuracy\_score \quad (3)$$

$$sklearn.metrics.confusion\_matrix \quad (4)$$

to evaluate the model's performance. *F1_score* and *accuracy_score* were used to measure the model's accuracy, whereas *confusion_matrix* was used to visualise if a model performs well on predicting labels, using number of *true positive*, *true negative*, *false positive* and *false negative*.

3.3.2 Learning Rate Tuning

We used a range of different learning rates from 0.01 to 0.1 to train each model and find the local maxima (unfortunately we cannot find global maxima due to algorithm complexity and excessive time of training), collect their accuracy and F1-scores, and plotted them into a series of figures, one of them is shown below (Figure 4) and the others are shown in the Appendix A:

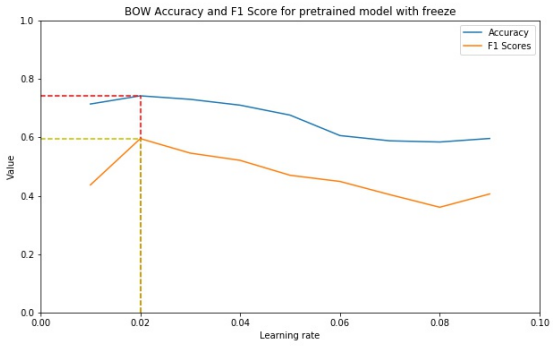

Figure 4: Performance of the BoW Model using pre-trained word embeddings and frozen

From the experimental results, we chose the best learning rate which leads to the highest accuracy which also has a satisfying F1-score. After exploring the six sets, we chose 0.02 for the learning rates for BoW models, 0.07 for BiLSTM with random initialised word embeddings and 0.08 for the BiLSTM with pre-trained word embeddings.

3.4 Experimental Results

Table 1 presents the model testing results using the tuned learning rates for both models. Using the best word embedding settings for both models, we build an ensemble model and shows the result in Table 2 with better performance.

Model	Embedding settings	Learning rate	Accuracy	F1-score
BiLSTM	pre-trained fine-tune	0.08	0.84	0.67
	pre-trained freeze	0.08	0.83	0.68
	random	0.07	0.78	0.60
BoW	pre-trained fine-tune	0.02	0.80	0.67
	pre-trained freeze		0.74	0.57
	random		0.73	0.53

Table 1: Learning rate tuning results

Model	Embedding settings	Evaluation
BiLSTM	pre-trained fine-tune	Accuracy 0.836 F1-score 0.666
BoW		Accuracy 0.800 F1-score 0.665
Ensemble	Ensemble above	Accuracy 0.852 F1-score 0.697

Table 2: Ensemble results

4 Analysis

Each row in the data set was a tuple of label, the question type and feature, the question string. The feature is used to train the model and the result prediction of question type (predicted label) is compared with the labelled question type (true label).

In this section, some in-depth analyses are discussed regarding different settings.

4.1 Reduced Training Set Result

In order to investigate the effect of using only part of the training set, we experimented with the BoW model with frozen pre-trained word embeddings.

In Table 3, we present the testing results for this experiment. We can observe that the accuracies and F1-scores decrease with fewer training samples utilised. This result is consistent with our expectations, as fewer training data can result in higher variance when a suitable model is chosen. Test set accuracy decreases gradually with the increase in variance.

Train used	100%	80%	60%	40%
Training set accuracy (epoch 9)	0.78	0.76	0.73	0.72
Testing set accuracy	0.74	0.69	0.66	0.56
F1-score	0.57	0.50	0.45	0.37

Table 3: Results with reduced size of training set

4.2 Embedding Setting Comparison

4.2.1 Fine-tuning versus Frozen

According to Table 1, for both BiLSTM and Bow models, pre-trained word embeddings with fine-tuning have better accuracy and F1-score than pre-trained word embeddings with frozen. The result is as expected since the embedding layer with a fine-tuning option allows the weights of word embeddings to be updated during the training process, making them better fitted for this specific task than the frozen one.

4.2.2 Pre-trained embeddings versus Random embeddings

Test results with pre-trained word embeddings have shown to be better than randomly initialized word embeddings, whether or not the pre-trained word embedding freezes itself or updates during training. This result is as predicted because the pre-trained word embeddings have data vectors related to the question type before being put into the models as input, so in training, pre-trained word embeddings have essentially more training epochs than randomly initialised word embedding, meaning they undoubtedly would have a higher accuracy and F1-score.

4.3 Most Misclassified Labels

Each model was trained with three different settings for word embeddings and the three most misclassified labels are shown in Table 4.

Model	Embedding settings	Misclassify #1	Misclassify #2	Misclassify #3
Bilstm	pre-trained fine-tune	('ENTY:substance', 7)	('NUM:other', 6)	('DESC:def', 6)
	pre-trained freeze	('ENTY:other', 8)	('ENTY:substance', 8)	('DESC:def', 7)
	random	('ENTY:other', 9)	('ENTY:substance', 9)	('LOC:other', 8)
Bow	pre-trained fine-tune	('DESC:def', 16),	('NUM:other', 11)	('ENTY:substance', 8)
	pre-trained freeze	('DESC:def', 21),	('NUM:dist', 9)	('ENTY:animal', 9)
	random	('LOC:other', 12)	('ENTY:animal', 12)	('NUM:dist', 11)

Table 4: Most misclassified labels

Some labels are more difficult to predict than others due to the data imbalance in the training set. For example, there are 386 data samples in the training set labelled "DESC: def" and 406 samples labelled "LOC: other" both have a significantly higher occurrence than other labels.

As a consequence, the model become overfitted to the frequent labels presented in the training set, making these labels more likely to be misclassified in testing as it has trouble generalising to new data .

4.4 Confusion Matrix

The confusion matrices are plotted below using heat map to compare the model performance on the classification of question type labels.

One of them is shown in Figure 5 and the others are shown in the Appendix B.

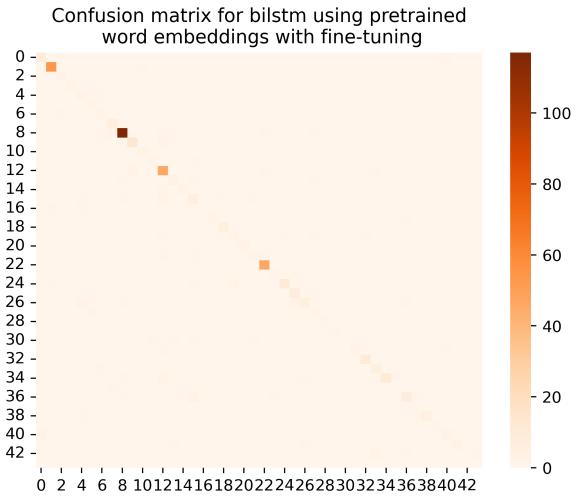

Figure 5: Confusion Matrix of BiLSTM Model with pre-trained and fine-tuning

In a confusion matrix, on each row, if the square with the deepest colour is on the main diagonal, it represents the prediction result of that label is well performed. From the confusion matrix shown above, we can see it generally follows this trend meaning that the performance of the BiLSTM model with pre-trained word embeddings and fine-tuning is satisfactory.

The cell on the main diagonal line represents the label is predicted correctly and the deeper the colour, the higher the occurrence of that label. Therefore the difference in the depth of colour in the confusion matrix above suggests data imbalance in the testing set.

5 Conclusion

In this experiment, we have implemented 2 machine learning models together with 3 different word embedding settings to classify question types. We also implemented one ensemble model based on the two models built before. Generally speaking, we found that the BiLSTM model has a better performance in terms of accuracy and F1-score than the BoW model, the pre-trained word embeddings perform better than randomly initialised word embeddings. The setting which will lead to the best performance is found to be the BiLSTM model and pre-trained word embeddings with fine-tuning. With the help of the ensemble method, the performance can be further enhanced based on the two models with the best settings.

References

Heaton, J. (2008). *Introduction to Neural Networks for Java, Second Edition*.

Kocmi, T., & Bojar, O. (2017). An exploration of word embedding initialization in deep-learning tasks. *CoRR*, *abs/1711.09160*.
URL http://arxiv.org/abs/1711.09160

Li, X., & Roth, D. (2002). Experimental data for question classification.
URL https://cogcomp.seas.upenn.edu/Data/QA/QC/

A Appendix: Learning Rate Selection

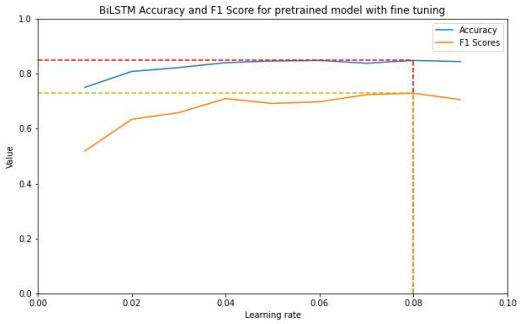

Figure 6: Evaluation of BiLSTM Model with pre-trained and fine-tuning

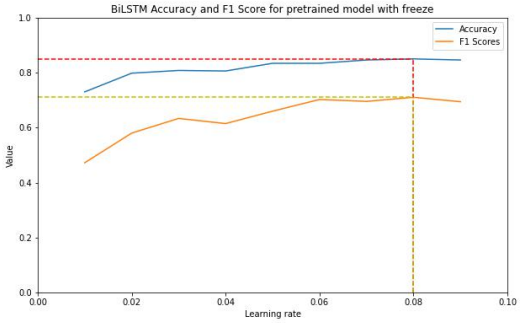

Figure 7: Evaluation of BiLSTM Model with pre-trained and freeze

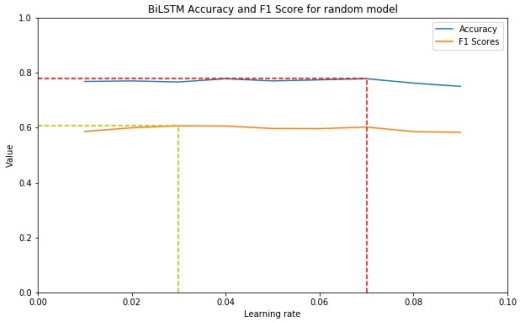

Figure 8: Evaluation of BiLSTM Model with random initialised embeddings

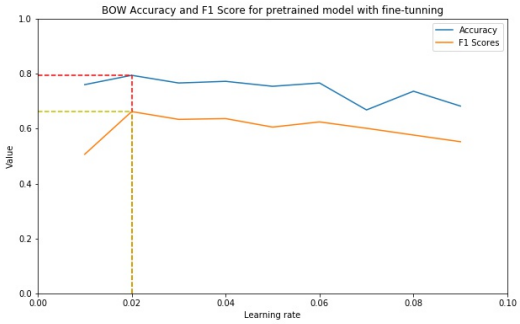

Figure 9: Evaluation of BoW Model with pre-trained and fine-tuning

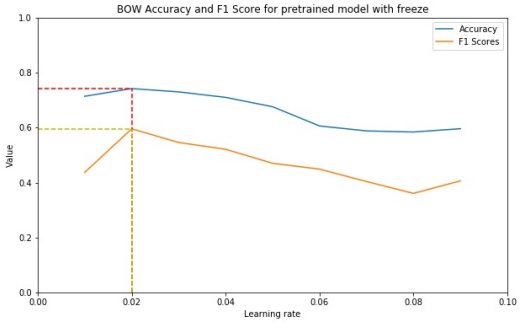

Figure 10: Evaluation of BoW Model with pre-trained and freeze

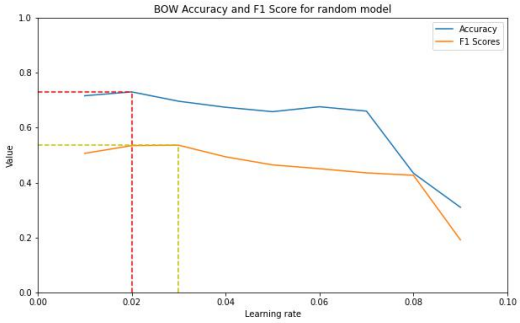

Figure 11: Evaluation of BoW Model with random initialised embeddings

B Appendix: Confusion Matrices

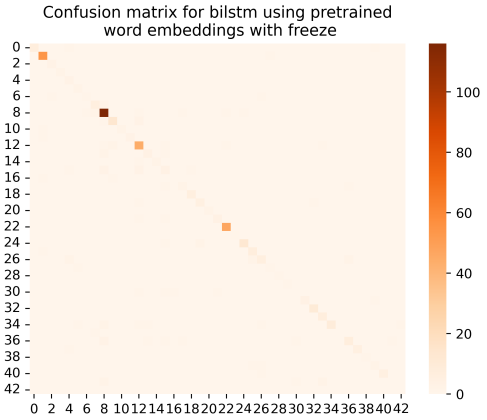

Figure 12: Confusion Matrix of BiLSTM Model with pre-trained and freeze

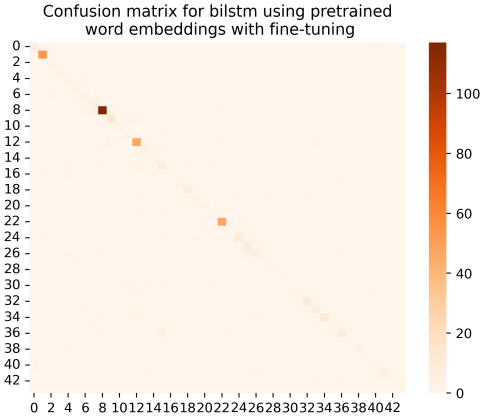

Figure 13: Confusion Matrix of BiLSTM Model with pre-trained and fine-tuning

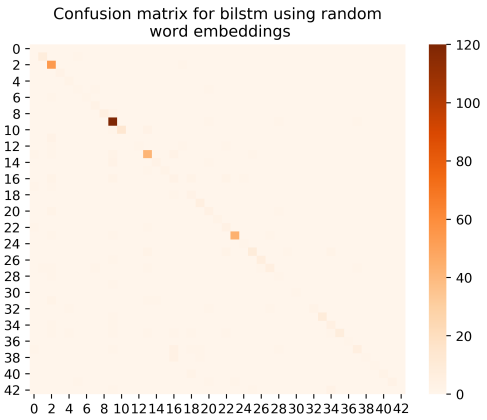

Figure 14: Confusion Matrix of BiLSTM Model with random initialised embeddings

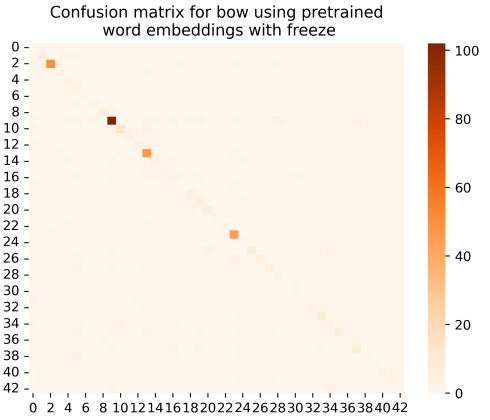

Figure 15: Confusion Matrix of BoW Model with pre-trained and freeze

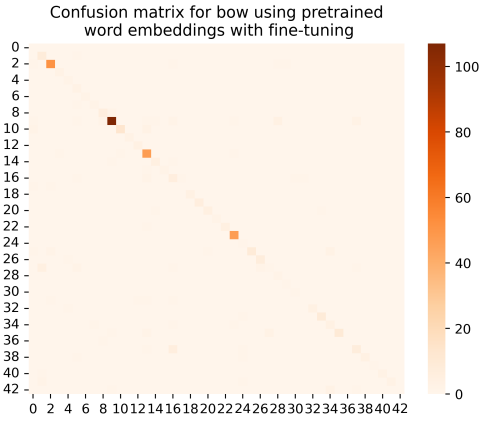

Figure 16: Confusion Matrix of BoW Model with pre-trained and fine-tuning

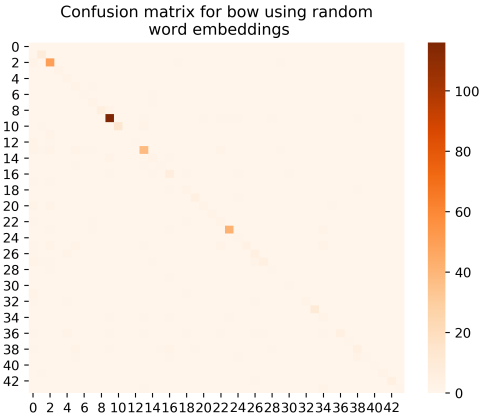

Figure 17: Confusion Matrix of BoW Model with random initialised embeddings