Report of Coursework 1

Boxin Huang, Guowen Song, Huanjie Guo, Qi Dai, Xiaohan Yu, Yi Miao

1. Introduction

Question-answering has become one of the most popular applications in Natural Language Processing industry. In order to answer a question correctly, the search engine needs to understand what the question asks for rather than simply returning lots of documents containing the keywords of the question. Question Classification, aiming at the identification of the semantic categories of questions, can not only reduce the search space of plausible answers but also suggest different processing strategies towards different types. The accuracy of question classification plays an important role in enhancing the classification process of question-answering system. In this paper, two question classifiers were built based on different sentence representation approaches. And our goal was to find out which was the most efficient way to solve this question classification problem.

2. Sentence representation models

The features of machine learning are basically numerical attributes, which allows anyone to perform mathematical operations on them. But in natural language processing, it is usually the case that the dataset contains more string value and character value than numerical value. Hence, to classify questions, first, we need to convert questions into vectors. This can be achieved in 3 steps, namely pre-processing, word embeddings and sentence representation. The first two steps are explained in detail in section 3. Since sentence representation is the most critical phrase to convert sentences to vectors, a variety of different techniques can be used to represent sentences as vectors. In this coursework, we chose to study the bag-of-words and the BiLSTM approaches to process sentence representation.

2.1Bag-of-Words

Bag-of-words (BOW) is a very simple and flexible approach to extract features from documents because it only measures the occurrence of words and discards the order and structure of words. For example, *"Tom gives Ben an apple"* and *"Ben gives Tom an apple"* have the same sentence representations because they have the same vocabulary.
According to the vector representation of bag-of-words:

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

where *s* is a sentence/question and $vec_{bow}(s)$ is the vector representation for *s*. *vec(w)* is the vector representation for word *w*.
To obtain a sentence's vector representation, we just need to perform an addition operation on all

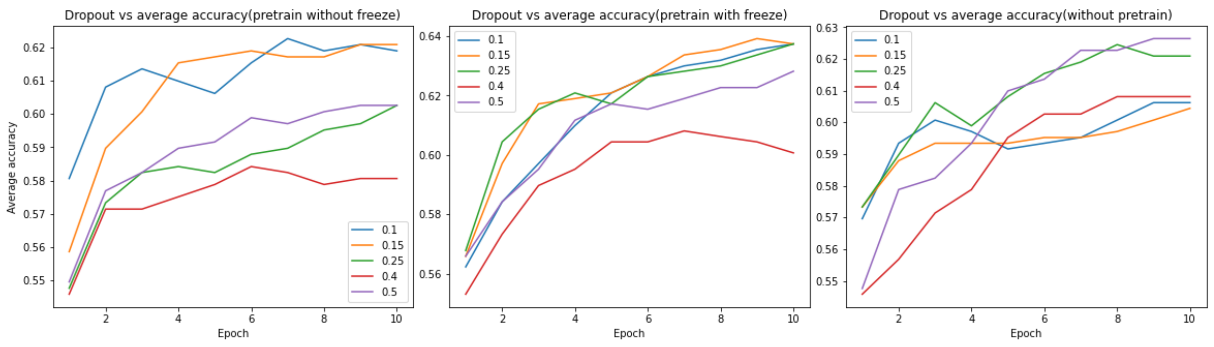

Figure 1Training accuracy with 3 embedding options

the word vectors achieved in word embeddings stage. Then divide the result vector by the sentence length. For a sentence set, in this coursework, we simply use a loop to process each sentence's vector representation and append them to the sentence representations list.

2.2BiLSTM

An RNN is more accurate than a strictly feedforward neural network in nature language processing problems because we can include sequence information of words in it. LSTM network is a special kind of RNN, which is capable of learning long-term dependencies. A BiLSTM, or Bidirectional LSTMs is a sequence processing model that consists of two LSTMs. One takes the input in a forward direction, and the other in a backwards direction.
This BiLSTM module includes two outputs, out and cell. Out is used to obtain the sentence classification of the model, which needs to be used for training and testing, and the cell is used to obtain the sentence representation. In detail, after word embeddings, those word vectors are passed to LSTM cells. Then, the LSTM cells add recurrent connections to the network and give us the ability to include information about the sequence of words in the question.

Choice of hyperparameters:

- Vocabulary dimension: set to 200 according to project requirements;
- Model layers: set to 1;
- Target_size: According to the number of labels, set to 50;
- Hidden layer dimension: In order to facilitate the combination of generated sentence representation and SoftMax, set it to 100.
- Dropout: Dropout can prevent the model from overfitting and improve the stability and robustness of the model.

Given five scattered dropout values in the range of 0.1～0.5, applying dev.txt (verification data) with other parameters fixed for testing via the final SoftMax classifier, obtaining the training accuracy rate in the three embedding cases. The results can be seen in figure 1.
In the first two cases, when dropout is set to 0.15, the model works best; although in one case, dropout of 0.15 performs poorly, the overall accuracy rate exceeds 60%, so dropout = 0.15 is selected as the hyperparameter of the BiLSTM model. Hyperparameters in Dataloader:

- Batch_size: Number of sentences per training, set to 10 in the project.
- Padding size: Unify the length of the sentences, determined by the input of the configuration file.

Training

Since the model is used for the sentence representation in the multi-classification problem, for the calculation of the loss of this classification, the method used by the model is to calculate the CrossEntropyLoss.

For the optimizer, the optimization algorithm uses Adam's method and uses the default parameters, that is, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False.

Testing

Function *eval()* is defined to observe the performance of the test dataset on the current model. This function compares the label calculated by the dataset under the current model with the real label to obtain the correct sentence of the classification result accounting for the total test data set Percentage.

In each epoch, after the training, using *eval()* to check the accuracy rate. After multiple training, the accuracy rate changes gradually, so stop training and select the model.

3. Experiment set-up

3.1Experimental data

In this coursework, we make use of the dataset from https://cogcomp.seas.upenn.edu/Data/QA/QC/. We used Training set 5 in the training stage. Because there was no development set, we split the dataset into 10 portions manually and allocated 9 portions for training and the other was for development purpose.

This was implemented in split_file.py.

3.2Pre-processing

As data in Training set 5 is like *"ENTY:cremat What films featured the character Popeye Doyle ?"*. Therefore, the first thing we did is to split each line from the first whitespace to obtain its label and sentence separately. Then, for each sentence, we lowercased the first letter in the first word. This was to prevent duplicates like "what" and "What" in the next step's vocabulary. After this, we removed periods or question marks at the end of each sentence. As to other extraneous punctuations that appear in sentences, we considered them as stop-words and removed them in the tokenization process. To clarify, the stopword list provided by NLTK cannot be used directly because it contains interrogative words like *"when"*, *"where"* and *"how"*, which play an important role in question classification.

The pre-processing stage was achieved in prepocessing.py.

3.3Word Embeddings

In this coursework, we implemented two kinds of word embeddings, which were randomly initialized word embeddings and pre-trained word embeddings.

Random initialized word embeddings: This is a simple way to embed words. To build a vocabulary, a threshold k was needed to filter those less frequent words besides tokens. Then, we mapped words in the vocabulary to integer indices and used *numpy.random()* method to randomly initialize every word vector.

Pre-trained word embeddings: In this coursework, we implemented the Word2Vec algorithm using the skip-gram architecture by ourselves. To define and train the skip-gram

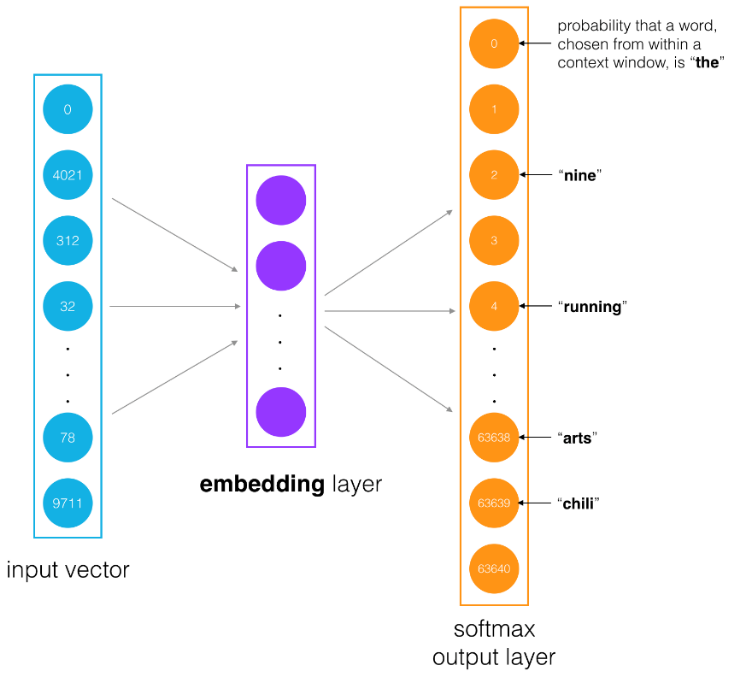

Figure 2 The general structure of skip-gram network. https://github.com/udacity/deep-learning-v2-pytorch/raw/9b0cc5718acbdb2436ce2169b8afa95716df6491/word2vec-embeddings/assets/skip_gram_arch.png

model, we first generated batches to define the surrounding context. Then, we built the skip-gram model with three layers. The input layer was batches of word tokens. The hidden layer (our embedding layer) consisted of linear units. Finally, into a *softmax* output layer.

In the model training process, we used a modified loss function where we only cared about the true example and a small subset of noise examples.

$$-\log\sigma({u_{w0}}^T v_{w1}) - \sum_{i}^{N} \mathbb{E}_{wi\sim P_n(w)} log\sigma\,(-{u_{wi}}^T v_{w1})$$

Where ${u_{w0}}^T$ is the embedding vector for our "output" target word and v_{w1} is the embedding vector for the "input" word.

3.4Evaluation metrics

In this paper, we used accuracy to evaluate the performance of our classifiers.

4. Experiments Results

Before running these 6 classifiers, we assumed that the BiLSTM models with pre-trained word embeddings with fine-tune option would perform the best accuracy. However, it turned out that the bag-of-words model with pre-train word embeddings with freeze option had the highest accuracy rate in this question classification problem (Table 1). In the meantime, we noticed that pre-trained word embeddings with fine-tune option had a higher accuracy rate than that with freeze option in both sentence representation methods. In addition, with randomly initialized word embeddings, the bag-of-words method obtained its lowest accuracy rate while the BiLSTM method reached its highest accuracy.

5. Ablation study

5.1 What happens if you freeze/fine-tune the pre-trained word embeddings?

To implement this function, we performed *nn.embedding.from_pretrained()* method on the pre-trained word embeddings. *nn.embedding.from_pretrained(weight, freeze = True)* means freeze pre-trained word embeddings are used in the following models. And *nn.embedding.from_pretrained(weight, freeze = Flase)* means fine-tune pre-trained word embeddings are used. Unlike the freeze one, pre-trained word embeddings with fine tune option can modify the weight of word embeddings to obtain a better classification performance. This was validated by the experiment results in section 4.

No.	Word Embedding		Sentence Representation	Acc 1	Acc 2	Acc 3	Average Acc
1	Random Initialized	———	Bag-of-Words	0.6160	0.6240	0.6160	0.6190
2	Random Initialized	———	BiLSTM	0.5476	0.6099	0.6227	0.6264
3	Pre-train	freeze	Bag-of-Words	0.666	0.652	0.658	0.6590
4	Pre-train	freeze	BiLSTM	0.5934	0.5952	0.6007	0.6044
5	Pre-train	fine-tune	Bag-of-Words	0.6680	0.6460	0.6740	0.6630
6	Pre-train	fine-tune	BiLSTM	0.5458	0.6136	0.6227	0.6190

Table 1 Experiment results of 6 combinations

5.2 What happens if you use randomly initialized word embeddings instead of pre-trained word embeddings?

Compared to pre-trained word embeddings, randomly initialized word embeddings provide the weight of word vectors by randomly initializing a *num×embedding_dim* shape tensor, which cannot reveal the connection between two semantically similar words. Therefore, in the bag-of-words methods, the performance of randomly initialized word embeddings in this question classification problem was inferior to pre-trained word embeddings. This was validated by the experiment results in section 4.

6. In-depth analyses

6.1 What happens if you use only part of the training set?

This question was tested based on bag-of-words sentence representation with randomly initialized word embeddings. And we choose to train the classifier with 70% of the total training dataset. It turned out that, if we deleted 30% of the training set randomly, then the accuracy of question classification was reduced by approximately 28.9%, which is unacceptable and out of our expectations. Then we counted the frequency of the same label in the deleted dataset and that remained in the training dataset. We found that our random training data dropping algorithm caused the quantitative imbalanced distribution on different kinds of labels left in the training set.

Our improvement towards this problem was that we monitored the number of questions with different kinds of labels and dropped questions from groups that had more instances than others randomly to keep the quantity balanced among all groups. By implementing this strategy, we successfully improved the classifier’s accuracy, with an average improvement of about 3% compared to training with the whole dataset.

The approach we used in this problem enlightened us that to acquire a better classification performance based on a given dataset, we could prune the dataset and make the number of instances with different labels maintain roughly the same in quantity.

6.2 Which classes are more difficult to classify?

We collected sentences with wrong prediction in each method and summarized the top 5 labels which had high possibility to be predicted wrongly in table 2.

From table 2, we could conclude that different classifiers learnt quite different features, which made them operate different kinds of wrong predictions. For example, in these two bag-of-words methods with pre-trained word

Method \Most-common	**BoW, random initialized**	**BoW, pre-trained, freeze**	**BoW, pre-trained, fine-tune**	**BiLSTM, random initialized**	**BiLSTM, pre-trained, freeze**	**BiLSTM, pre-trained, fine-tune**
1	NUM:dist	DESC:def	DESC:def	HUM:desc	DESC:def	LOC:other
2	ENTY:other	LOC:other	LOC:other	ENTY:termeq	ENTY:techmeth	DESC:def
3	ENTY:substance	ENTY:substance	ENTY:substance	HUM:ind	DESC:def	NUM:speed
4	NUM:other	NUM:dist	NUM:dist	DESC:def	ENTY:techmeth	ENTY:animal
5	ENTY:animal	ENTY:other	NUM:date	ENTY:techmeth	ENTY:plant	ENTY:termeq

Table 2 Top 5 wrongly classified labels with each method

embedding approaches, the classifiers often made wrong prediction between DESC:def and sub-classed in the ENTITY class, this was mainly because almost all questions in classed mentioned above started with the word “what” and the classifiers failed to learn the slight difference between those labels.

7. Conclusion

In this paper, we implemented different word embedding methods and studied the theory of the Word2Vec algorithm that used the skip-gram architecture. We built classifiers based on different sentence representation methods and determined that the bag-of-words model with pre-trained word embeddings with freeze option performed the best accuracy. From the training testing experience of this classification problem, we found that classifiers with different models tended to perform wrong classification result on different but fixed labels. Therefore, our future work will be trying to analyze the internal reason for our finding and give physical interpretations.

8. References

Zhang, D., Lee, W.S., 2003. *Question classification using support vector machines*, in: .. doi:10.1145/860435.860443

Marinčič, D., Kompara, T., Gams, M., 2012. *Question Classification with Active Learning*, in: Transactions on Petri Nets and Other Models of Concurrency XV. Transactions on Petri Nets and Other Models of Concurrency XV, pp. 673–680.. doi:10.1007/978-3-642-32790-2_82

Li, X., Huang, X., & Wu, L. (2005). *Question Classification using Multiple Classifiers*. ALR/ALRN@IJCNLP.

Alaa Mohasseb, Mohamed Bader-El-Den, Mihaela Cocea, *Question categorization and classification using grammar based approach*, Information Processing & Management, Volume 54, Issue 6, 2018, Pages 1228-1243, ISSN 0306-4573,

Paritosh, Pantola. *“Natural Language Processing: Text Data Vectorization”* Jun 14 2018, https://medium.com/@paritosh_30025/natural-language-processing-text-data-vectorization-af2520529cf7

Prabhu. *“Understanding NLP Word Embeddings — Text Vectorization”* Nov 11 2019, https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223

Jason, Brownlee. *“A Gentle Introduction to the Bag-of-Words Model”* Oct 9 2017, https://machinelearningmastery.com/gentle-introduction-bag-words-model/

Jason, Brownlee. *“A Gentle Introduction to Long Short-Term Memory Networks by the Experts”* May 24 2017, https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

Christopher, Olah. *“Understanding LSTM Networks”* Aug 27 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Cezanne, Camacho. *“SkipGram Word2Vec implementation”* Oct 16 2018, https://github.com/udacity/deep-learning-v2-pytorch/blob/master/word2vec-embeddings/Negative_Sampling_Solution.ipynb

N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov (2014). *“Dropout: A Simple Way to Prevent Neural Networks from Overfitting.”* Journal of Machine Learning Research 15(1):1929-1958

Kingma, D and Ba, J. (2015) Adam: *A method for Stochastic Optimization.*

Ruder, S. (2017) *An overview of gradient descent optimization algorithms.*