YNU_AI1799 at SemEval-2018 Task 11: Machine Comprehension using Commonsense Knowledge of Different model ensemble

QingXun Liu, HongDou Yao, Xiaobing Zhou*, Ge Xie
School of Information Science and Engineering
Yunnan University, Yunnan, P.R. China
*Corresponding author, zhouxb.cn@gmail.com

Abstract

In this paper, we describe a machine reading comprehension system that participated in SemEval-2018 Task 11: Machine Comprehension using commonsense knowledge. In this work, we train a series of neural network models such as multi-LSTM, BiLSTM, multi-BiLSTM-CNN and attention-based BiLSTM, etc. On top of some sub models, there are two kinds of word embedding: (a) general word embedding generated from unsupervised neural language model; and (b) position embedding generated from general word embedding. Finally, we make a hard vote on the predictions of these models and achieve relatively good result. The proposed approach achieves 8th place in Task 11 with the accuracy of 0.7213.

1 Introduction

Machine Comprehension using Commonsense Knowledge is a well-researched problem in NLP. In order to simplify the task of the process, we turn this task into text classification work and use a deep learning neural network to fulfill it. The method of deep learning models used in text analysis has achieved numerous notable advances in recently years (e.g., (Breck et al., 2007), (Yessenalina and Cardie, 2011) and (Socher et al., 2011)). However, in most previous works, the tasks are to apply a single model to a particular data set task.

The single model is a vertical stack of multiple hidden layers, which is not good for text analysis and processing. The first drawback is the need to consume more hardware resources, followed by over-fitting and loss of feature information. So the task here is to apply different structure sub models to the same train-set. We train many classic sub models with one layer on top of word embedding, like LSTM, CNN, Attention,Attention+BiLSTM, multi-BiLSTM+CNN and some other models which are slightly different from the above models with different activation functions and different layers inside the model. In each single model, we use a flag to determine which embedding tool is used or not.

Most of the deep learning involve word vectors represented by neural language models ((Morin and Bengio, 2005) , (Yih et al., 2011) and (Mikolov et al., 2013)). Using the learned word vectors for classification task will naturally increase the effect. Word vectors are expressed as a hidden-layer word vector of the specified dimension (1-of-V, here V is the vocabulary size), the training methods can be found here[1]. In our system, we introduce different word vectors: 100 billion words of Google News, Glove vectors of 100 dimensions and word vectors self-trained on the basis of official task data. Finally, a hard vote(the majority voting from result document) is made on the results of those different models. Many tasks often suffer from insufficient training data. In this work, we parse external data from CodaLab introduction data, including DeScript(**?**) data, RKP(Regneri et al., 2010) data and OMCS (Singh et al., 2002) data to trained embedding.

2 System Description

We treat this task as a classification process. First we repeat the question and answer text, making it match instance texts. The final number of samples are same as the number of answers in the data-set. After statistical analysis of the data-set, we treat Instance texts, Question texts and Answers texts as to a rational text length of words $\{w_1,w_2,\ldots,w_n\}$ in which n is the max length of a text. Here, the length of Instance is controlled as 350, and the length of Question is controlled as 14 as well as Answer. Before that, we count the frequency of

[1]https://radimrehurek.com/gensim/models/word2vec.html

Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), pages 1038–1042
New Orleans, Louisiana, June 5–6, 2018.

occurrence of each word in the data-set, and use this word frequency to create a dictionary, then express each word in terms of frequency order of corresponding word (Kim, 2014). Next, we train word embeddding according to (Chiu et al., 2016), and at the same time download the trained embedding sets that have been trained[2].

The ensemble model architecture, shown in figure 1, is an ensemble of many single models(We call them sub models). Because each sub model is independent of each other, their weights are not shared and just use the same word embedding when training each sub model. The process of the whole ensemble model is carried out model by model. First, each model is run independently, and then the result file is saved. After running all the independent models, the result files are taken out and the final result is determined by the majority vote. In model training, we use the early stop mechanism (Sarle, 1995) to terminate the training of subsequent epoch when the overfitting is appeared. At the same time, the data is shuffled on every epoch. The core of our paper is based on classic models, adding other network layers, so that the independent models have their own structure. Next, are introduced the input structures of the sub models.

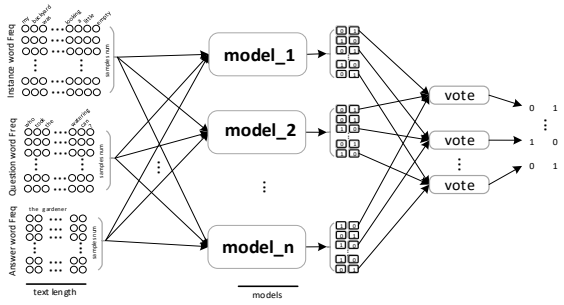

Figure 1: The architecture of the models ensemble.

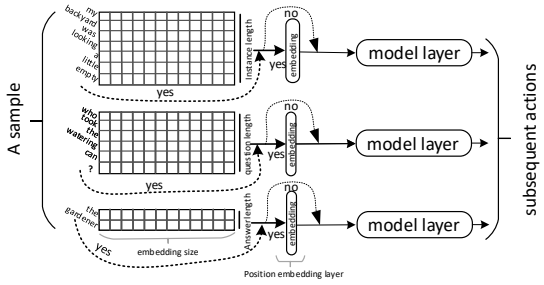

Figure 2: The input structure of the submodel

[2]https://github.com/xgli/word2vec-api

2.1 Similar input structures

Figure 2 shows the architecture of the sub models input. The input part of all sub models uses this structure. Within the structure, two flag variables are used to control the use of word embedding. One is whether to use the position embedding, and the other is to control whether or not the trained word embedding is used. From this structure, data flow to the subsequent network layer such as the classic model layer CNN, LSTM, BiLSTM and Attention.

2.2 The merge layer

In order to combine three text feature information, we have a merge layer in all of our sub models. The merge layer of these sub models is almost the same. Only the attention layer merge is different from other sub models. The merge of most sub models first combines the instance matrix data with the question in sum operation, while combining the matrix data with the answer, and finally combining the two sets of matrix data in the *dot* operation. However, in the sub model with Attention, merge layer combines the matrix data of the three texts with *dot* operation. Next, we will focus on two sub models using classical models,while the rest of the other part of each sub models are similar to the two models.

2.3 Based on multi-BiLSTM+CNN model

Long short-term memory (LSTM or BiLSTM) is applied to text classification ((Liu et al., 2016) and (Lai et al., 2015)). The Convolutional Neural Networks (CNN) is for local feature extraction (LeCun et al., 1998). CNN can achieve good results in the semantic analysis (Yih et al., 2014), and other traditional NLP tasks (Collobert et al., 2011), especially in sentiment analysis and question classification (Kim et al., 2016). It is our novelty to combine CNN with multiple layers of BiLSTM with BiLSTM in front.

Here set the *multi* = 2, of course, it can be 3 or more. Each multi-BiLSTM internal implies a dropout layer to prevent over-fitting (Srivastava et al., 2014). Multi-BiLSTM is one of the core layers in this model which takes an input sequence of word embedding. Just like (Liu et al., 2016) and (Lai et al., 2015), this layer runs on the data of word embedding. After these three branches running (Just as the three model layers in Figure 2), we make batch normalization and then merge

them into CNN1D layer. Finally, we use the Softmax classifier to predict the results. Other sub models that do not use Attention are similar to the sub model structure, instead of replacing CNN1D with other structures.

2.4 Based on Attention + BiLSTM model

Attention is mostly used for document categorization (Yang et al., 2016).The model architecture is different from the multi-BiLSTM+CNN architecture of word embedding layer. The structure of this model is roughly: a word embedding input structure, followed by attention layer which include the merge layer. Next to the batch normalization layer, BiLSTM layer , Softmax layer. We do a position embedding operation before inputting the word embedding into attention layer. According to (Vaswani et al., 2017), the formula for constructing Position Embedding is as follows:

$$\begin{cases} PE_{2i(p)} = sin(p/10000^{2i/dpos}) \\ PE_{2i+1(p)} = sin(p/10000^{2i/dpos}) \end{cases} \quad (1)$$

Here is to map the position *p* to the position vector of *dpos* dimension. The value of the *i* th element of this vector is $PE_i(p)$. Word embedding first goes through the position embedding layer which is included in the architecture of the sub model input. Then the feature data enters into attention layer. In attention layer, weights and bias are randomly added to position embedding and excess numbers are masked as zero. We do batch normalization for the data coming out of attention layer, then input them into BiLSTM. Similarly, the results are obtained after Softmax layer.

3 Data Preparation

Although official data is regular, we have done a further normalization. All data set used by each model undergoes the following preprocessing steps:

1) The texts were lowercased

2) Using NLTK to tokenize each text

3) Abbreviations:
We're very careful about the abbreviation, as "'s" in "it's time for me to take her out." is not the same as "s" in "Tom's dad ordered pizza yesterday for the family." We treat the first "s" as "is," and the second, of course, is an adjective. We restore those abbreviations in Table 1 to normal forms.

examples	normalization
I'm	I am
n't	not
does't	does not
it's	it is
...	...
that's	that is
neighbor's	neighbor
wouldn't	would not
wont't	will not

Table 1: normalization patterns.

4) Removing other characters:
Removing other characters, such as "!","%","?", "#", "","@" Etc. Of course, not all other symbols that seem useless are removed. Like "$" are not removed.

4 Experiments and Results

In order to optimize our network, we use (Kingma and Ba, 2014) optimizer on training model. All our experiments have been developed using an open source software library of Tensorflow with CUDA enabled, and run on a computer with Intel Core(TM) i3 CPU 760 @2.8GHz, 8GB of RAM and GeForce GTX960 GPU. Due to the lack of hardware capacity, we do not run the entire system in one time. Instead, we run single model each time with different word embeddings. When we use the word embedding of Google News 300d on some sub models, the system gives memory exhausted, and we switch to a smaller glove_27B_100d to run successfully. Table 2 shows our results for various models. As it can be seen from the table, ensemble results from the more different models get better results when other conditions are similar. Here we ensemble these models: RNN, GRU, BiLSTM, multi-BiLSTM+CNN and Attention+BiLSTM, based on their high accuracy. The dropout probability is 0.6 in each model, and the initial learning rate is 0.01.

5 Conclusion

In this project, we ensemble a variety of structurally different models to tackle this task. The

model	self trained	glove_twitter_27B_100d	GoogleNews_300d.bin
RNN	—	0.6638	—
LSTM	0.7001	0.7042	0.6932
GRU	—	0.6732	—
CNN1D	0.5634	0.6324	—
CNN2D	—	0.5683	—
CNN2D+LSTM	—	0.5573	—
BiLSTM	0.6734	0.7135	—
Attention	—	0.6731	0.6863
Attention+BiLSTM	0.6653	0.6943	0.6934
multi-BiLSTM+CNN	0.6725	0.6834	—
Combine to all models	0.6550	0.7213	0.6923

Table 2: Result for various models on task data set.

performance of a single model is poorer than the ensemble model. And the bigger the difference between the models, the higher performance the ensemble model makes. Still our results are less satisfying than top teams on the leaderboard. We will adjust the model, improve the hardware configuration of the computer, collect more external data, and do more experiments to get a better result in the future.

Acknowledgments

This work was supported by the Natural Science Foundation of China No.61463050, No.61702443, No.61762091, the NSF of Yunnan Province No.2015FB113, the Project of Innovative Research Team of Yunnan Province.

References

Eric Breck, Yejin Choi, and Claire Cardie. 2007. Identifying expressions of opinion in context. In *International Joint Conference on Artificial Intelligence*, pages 2683–2688.

Billy Chiu, Gamal Crichton, Anna Korhonen, and Sampo Pyysalo. 2016. How to train good word embeddings for biomedical nlp. In *Proceedings of the 15th Workshop on Biomedical Natural Language Processing*, pages 166–174.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *AAAI*, pages 2741–2749.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *In International Conference on Learning Representations (ICLR)*.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In *International Joint Conference on Artificial Intelligence*, pages 2873–2879.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252.

Michaela Regneri, Alexander Koller, and Manfred Pinkal. 2010. Learning script knowledge with web experiments. In *Meeting of the Association for Computational Linguistics*, pages 979–988.

Warren S. Sarle. 1995. Stopped training and other remedies for overfitting. *In Proceedings of Symposium on the Interface*, pages 352–360.

Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. 2002. Open mind common sense: Knowledge acquisition from the general public. 2519(5):1223–1237.

Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John Mcintyre Conference Centre, Edinburgh, Uk, A Meeting of Sigdat, A Special Interest Group of the ACL*, pages 151–161.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I., editor, *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.

Ainur Yessenalina and Claire Cardie. 2011. Compositional matrix-space models for sentiment analysis. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John Mcintyre Conference Centre, Edinburgh, Uk, A Meeting of Sigdat, A Special Interest Group of the ACL*, pages 172–182.

Wen-tau Yih, Xiaodong He, and Christopher Meek. 2014. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 643–648.

Wen-tau Yih, Kristina Toutanova, John C Platt, and Christopher Meek. 2011. Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 247–256.