Word Embeddings Comparison in Quora Insincere Questions Classification

Gaoyuan Huang

Abstract

In this paper, I compare six popular word emdeddings in Natural Language Processing using two neural network structures on Quora Insincere Question Classification dataset in Kaggle. The experiments show that the latest word embedding BERT performs best with a simple network. Adding new structures used in latest research offers a great improvement for old word embeddings. Task specific embeddings was trained as a control group which also emphasize the importance of fine-tuning.

1.Introduction

1.1Goal of this paper

Text classification is a major task in Natural Language Processing. How to represent texts has always been considered as one of the most important question in classifying texts. Word embedding as a good way to represent the content has been acting a major role recently. Utilizing huge improvement of neural network to better represent texts becomes a very effect method for NLP researchers. With so many word embeddings coming out, data science students like me and all other data science professionals have to keep up with these advancement.

Here my goal of this paper is to test whether the recent word embedding BERT, the current best of universal word embeddings and sentence embeddings, is as good as it shows in other data challenges. Another interesting to see is how user friendly and time effective this model is for common users like me. The Kaggle competition which only allows users to use Word2Vec, FastText, Glove and Pragram Embeddings gives me a nice place to show if BERT can beat them. My first usage of all word embeddings here is trying not to fine-tune them so that I can tell how well they learned the word representations before. Then I would try new techniques involved in latest research on the embeddings the competition provided to see if they can improve. For all these experiments, I set up a model as a control group which would have no pre-trained word embeddings. I hope through these experiments I can get familiar with each word embeddings but also understand the process of improving word representations.

1.2Task description

Quora Insincere Question Classification is aim to detect toxic content to improve online conversations. Internet is a place where almost anyone can say anything at anytime, which is a good thing but also a bad thing. The bad side of this is that many content of the internet contain

violent, sexaul or disparaging content which appears in some inappropriate places. People come to quora to ask for help, to share knowledge and learn from each other. Thus, these insincere content should be removed to keep Quora and other internet space nice and respectful.

The dataset contains 1306122 questions labeled with 1 and 0 to represent insincere questions and sincere questions respectively. 122513 questions are labeled as sincere questions while only 80810 questions are labeled as insincere questions. The metric used in this competition is F1 score which calculates a harmonic mean of precision and recall. This is fair due to the imbalanced target variables. Here are several examples of the data:
In sincere questions:
Has the United States become the largest dictatorship in the world?
How do I marry an American woman for a Green Card? How much do they charge?
Sincere questions:
Do you have an adopted dog, how would you encourage people to adopt and not shop?
Is it crazy if I wash or wipe my groceries off? Germs are everywhere.

1.3Related to previous work

Word embeddings, word2vec, are first trained by neural network by Mikolov (1). In the paper, they brought up CBOW and Skip-gram to predict a word by its neighbors or vise versa. Then Glove is invented based word2vec but the authors use matrix factorization skills to deal with some matrix in the networks rather than using sparse matrix which is a long time problem for NLP(2). FastText is later created to take advantages of n-grams so the word can have much more meanings in it(3). In order to improve the text classification result, CNN and bidirectional RNN was later applied(4,5). In 2017, transformer was invented by Google to make the model learn a longer context around it to fix the problem of RNN not learning well if the sequence is long(6). BERT uses transformer and bidirectional layers to enhance the network's ability to learn from its neighbor content(7).

2.Project Process

2.1Preprocess texts

I first take a look at the distribution of texts length. Most questions are 50-60 words long. Due to memory problem, I choose to set my max length of a question at 50. There are 195,000 unique words in the texts. I decide to use top 50,000 frequent unique words because they can represent 92% of all texts. Then using the max number of words I use tokenizer in Keras to process the texts. The tokenizer can remove punctuation and lower down tokens. Then I converted the training data (train_X) to indices for all the tokens. So now each token was

replaced by its index. Next, I pad the questions so each question will have the same shape of 50 as a fixed length. Now all questions are ready to be passed through word embeddings.

2.2Load word embeddings

For Word2Vec, FastText, Glove and Paragram embeddings, I use the one provided from this competition. I first create a dictionary ('embeddings_index') containing words and their vectors from those files. Then I create a 'embedding_matrix' with the shape of total number of words(min of max features(50,000) and total number of words this embedding file have) and the size of embeddings (300). I filled the embedding matrix with a random generated number from the normal distribution using the mean and standard deviation of all embedding vectors in embedding file. At the end, I check each word in our corpus (all tokens we get in last step) to see if they appear in 'embeddings_index', if yes then I can fill the our embedding matrix with the vector we get, if not we do nothing. After going through all words we tokenized, each row of the embedding matrix represent a word using 300 numbers. The index of this embedding matrix is consistent with the index of words we store in training data (train_X), so we can map training data with its corresponding vectors to create a representation of each question.

For ELMo and BERT, things are a little different. For BERT, each question should be passed through their model to get embeddings. BERT model will take care of details like tokenizations, padding and adding [EOS] tags. So for BERT, I can feed the model a list of raw string type questions. I choose to use a package bert-as-services which can do this in a simple way rather than directly getting into tensorflow codes. Due to limited computational power, I can only encode roughly 25,000 question per hour which took me a lot of time preparing the data for downstream networks. As for ELMo, I split the questions to choose first 50 words, and then concatenate them back to string storing in a list. Same as BERT, my input for ELMo is a list of strings. For next step of ELMo, it's not easy to do because I can't install AllenNLP, who created this model, or TensorFlow Hub, where the model also stores, in Kaggle, so I have to use google colab (AWS EC2 approved my request to change my limit of using GPU too late) which keeps breaking down if it runs too long. In the end, I just manage to complete three epochs of ELMo. I also feed a whole question of string type to ELMo. Both BERT and ELMo give me a vector of shape 1024.

2.3Build Neural Network

My experiments have two stage. In the first stage, I just use embedding layers, a dropout layer and dense layers. In the second stage, I add a bidirectional GRU layer, a max pooling layer for no-pretrained, Word2Vec, FastText, Glove and Paragram.

In the first stage, for Word2Vec, FastText, Glove and Paragram, first layer of my neural network is embedding layer (Fig 1). Each question in training data has a length of 50. After mapping with

embedding matrix, it then becomes 50*300 since each one maps a 300 length vector. Then for each question, I simply average the weights (the total number is 50) for each dimension (300 dimensions) to get a representation of a question. Now each question is represented by a 300 dimension vector which is then connected with three dense layers and one dropout layer to fix overfitting. The firs layer has a shape of 1024 because I want them to have same shape as ELMo and BERT when getting into the second dense layer which is the first layer for ELMo and BERT (see Fig 2). I set up a neural network without any pre-trained embeddings to better compare my results. The structure of it is same as Word2Vec etc, except the parameters in the embedding layer are trainable (See Fig 3). The activation function for the first two dense layers are “relu”. The last activation function is “sigmoid” so we can get result from 0-1 as the probability for the question to be a insincere question.

I set number of epoch for Word2Vec, FastText, Glove, Paragram and BERT as 20 (ELMo is 3 due to computational power, but the result is OK). I choose adam as the optimizer, accuracy as the metric and binary_crossentropy as loss function. The batch size for Word2Vec etc and BERT is 512. For ELMo, the size is 256 because the embedding is larger I have to avoid memory problem.

In the second stage, a bidirectional GRU layer and a max pooling layer is added. The dense layer which is used for making shape to be the same as BERT is removed. The purpose of this stage is to see whether the performance for Word2Vec, etc embeddings will improve and how much is the improvement if we add the techniques that is similar to what BERT uses.

2.4Evaluation

The evaluation is done by 5-fold cross validations. Due to the imbalance of target variable, the final metric to compare their performance is F1 score. For each prediction, I search from 0.1 to 0.501 with 0.01 as interval to find the best threshold value for the classification. After 5-fold cross validations, I calculate the mean and standard deviation of each model. For running time, since the way I load embeddings is quite different, the time varies much. For Word2Vec etc and BERT, it takes 8 seconds for one epoch of training. But ELMo takes more than 10,000s. BERT is quick because I extract embeddings in advance which takes me 4 days. I also saved history of all training to check how the loss and accuracy change for each epoch.

3.Results and discussion

The result of my experiments is presented below. In the first stage, BERT is much better than the rest of models even close to the second winner(0.71) in the public leaderboard in Kaggle with such a simple network without any other tweaking. We can definitely imagine how good the score will be if we can add more layers and fine-tune BERT. Followed by BERT is Np-pretrianed model. I’m surprised to see it’s better than ELMo. This shows how important it is to have

networks to learn as much as they can using the current corpus, even though ELMo was trained with hundreds of millions word corpus it's performing almost the same as no-pretrained model. The rest embedding is working well with such small network.

In the second stage, we can find out the huge improvement for Word2Vec etc. The result of Glove and FastText both reach 0.67 which is not I was expecting. This really shows how RNN layers like bidirectional GRU can capture the meaning of the context and boost the performance. It's also interesting to see that No-pretrained model doesn't improve too much (0.02). I think this in a way shows the advantages of pre-trained word embeddings. They have already 'read' a lot of contexts before, so when they are feeded with valuable information capture by RNN the model then can learn very effectively.

Word Embeddings	**F1-Score No RNN**	**STD No RNN**	**F1-Score with biGRU**	**STD with RNN**
No-pretrained	0.63	0.002	0.65	0.002
Word2vec	0.60	0.002	0.66	0.002
Glove	0.59	0.002	**0.67**	0.002
Paragram	0.59	0.003	0.66	0.003
FastText	0.60	0.002	**0.67**	0.002
ELMo	0.62	N/A	0.62	N/A
BERT	**0.67**	0.003	**0.67**	0.003

4.Conclusion and future work

Simple neural networks are built to compare how BERT, the best NLP model for now, performs in the same NLP task with other embeddings created prior to it. The result for first stage experiment is very obvious, with many fancy techniques (bidirectional LSTM, transformer, deep network) BERT beat all the embeddings. In the second stage, by comparing the result before and after adding RNN, I find the magic power of RNN. With just a little change of the layers, the result can be boosted as much as 11.8%. Word in texts is always related to the content before and after it, so a bidirectional RNN layer can help the model to get this information so the model can learn better. Although BERT beats all other models in 11 NLP tasks, it is created based on what people have found before. The structure of it is not mysterious but really effective.

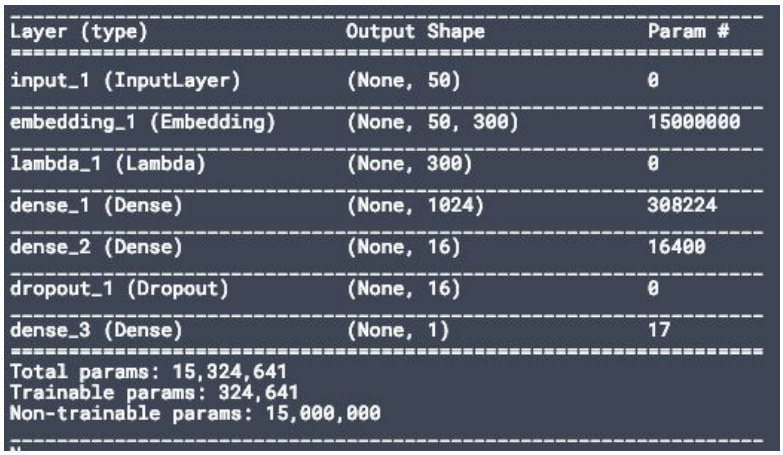
For future work, I hope I can first do more text cleaning to have more words/characters to be used. In terms of imbalance, downsampling techniques I used is not improving the performance so I'm not mentioning more details here. The most exciting thing I want to do is to fine-tune BERT model on this corpus to see how much improvement it can get. I would expect it to be similar to the winner on the public leaderboard(0.78). I also saw many kagglers implementing attention layers which BERT uses in their model to improve the performance of Word2Vec etc. I

wish there is a much more user-friendly package to use BERT in the future, otherwise the fancy model is not likely to be used by normal users.

Reference

1. Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.
2. Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
3. Joulin, Armand, et al. "Bag of tricks for efficient text classification." arXiv preprint arXiv:1607.01759 (2016).
4. Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).
5. Peters, Matthew E., et al. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018).
6. Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.
7. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

Appendix



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 50)	0
embedding_1 (Embedding)	(None, 50, 300)	15000000
lambda_1 (Lambda)	(None, 300)	0
dense_1 (Dense)	(None, 1024)	308224
dense_2 (Dense)	(None, 16)	16400
dropout_1 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 1)	17

Total params: 15,324,641
Trainable params: 324,641
Non-trainable params: 15,000,000

Fig1.1st stage structure of Neural Network for Word2Vec, FastText, Glove and Paragram

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                16400
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 16,417
Trainable params: 16,417
Non-trainable params: 0
```

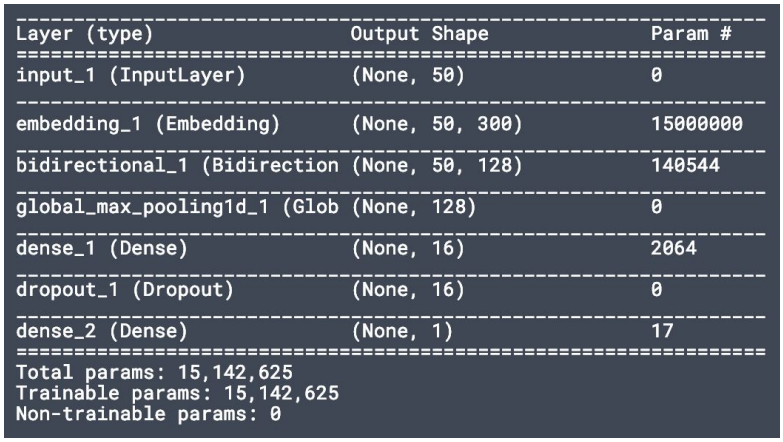
Fig 2: structure of Neural Network for ELMo and BERT

```
 _________________________________________________________________
 Layer (type)                 Output Shape              Param #
 =================================================================
 input_1 (InputLayer)         (None, 50)                0
 _________________________________________________________________
 embedding_1 (Embedding)      (None, 50, 300)           15000000
 _________________________________________________________________
 bidirectional_1 (Bidirection (None, 50, 128)           140544
 _________________________________________________________________
 global_max_pooling1d_1 (Glob (None, 128)               0
 _________________________________________________________________
 dense_1 (Dense)              (None, 16)                2064
 _________________________________________________________________
 dropout_1 (Dropout)          (None, 16)                0
 _________________________________________________________________
 dense_2 (Dense)              (None, 1)                 17
 =================================================================
 Total params: 15,142,625
 Trainable params: 15,142,625
 Non-trainable params: 0
```

Fig 3: 1st stage structure of no-pretrained embedding model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 50)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 50, 300)           15000000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 50, 128)           140544
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                2064
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 15,142,625
Trainable params: 142,625
Non-trainable params: 15,000,000
```

Fig 4: 2rd stage structure of Neural Network for Word2Vec, FastText, Glove and Paragram



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 50)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 50, 300)           15000000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 50, 128)           140544
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                2064
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 15,142,625
Trainable params: 15,142,625
Non-trainable params: 0
```

Fig 5: 2rd stage structure of no-pretrained embedding model