Darmstadt's Deep Learning for NLP 2020 Shared Task

Cui Yi
TU Darmstadt
yi.cui@stud.tu-darmstadt.de

Chen Zixuan
TU Darmstadt
chenzixuan0613@gmail.com

Abstract

This article introduces the similarity of semantic texts under the adversarial attack in the shared task submitted to DL4NLP 2020 by the University of Darmstadt. Our system first recovers certain characters under attack, then converts the words into word vectors and calculates the average word vector of the sentence as input, and then uses a simple MLP network model.

1 Introduction

The DL4NLP 2020 shared task deals with semantic textual similarity under adversarial attacks. In the shared task, we define semantic textual similarity (STS) as a supervised regression task in which the semantic similarity of two pieces of text (typically sentences) should be determined. However, the test datasets have been perturbed with so-called adversarial attacks, which are modifications to the input of a model that do not change the label or score. Usually, their goal is to fool a deep learning model but not humans. Thus, attacks are often chosen in such a way that humans are robust to them. (1)
Examples for adversarial attacks are:
• disemvoweling: Some of the vowels have been removed.
• visual attacks: Similar-looking characters have been replaced by each other.

2 Related work

In this task, in order to get the correct evaluation of the attacked test set. We first deal with the attacked characters, and then try to train a simple multi-layer perceptron (MLP) networks and long short-term memory (LSTM) networks to score the similarity of the two sentences.

2.1 Reverse VIPER

According to the visual perturber VIPER (2), that some adversarial attacks randomly replace characters in the input with their visual nearest neighbors in a visual embedding space. In our approach we only attempted to choose one of them to reverse the adversarial attacks. We choose to use the description-based character embedding space (DCES) to recover attacked characters. DCES is based on the textual descriptions of Unicode characters. We first obtain descriptions of each character from the Unicode 11.0.0 final names list (e.g., LATIN SMALL LETTER A for the character 'a'). Then we determine a set of nearest neighbors by choosing all characters whose descriptions refer to the same letter in the same case, e.g., an alternative to "LATIN SMALL LETTER A WITH GRAVE" is "LATIN SMALL LETTER A" as it contains the keywords SMALL and A. (2)

This preprocessing can only recover part of the attacked characters. But there are still many other types of attacks that are not considered. Some attacked characters are still in the word, which causes "out of vocabulary". That means can not be found in the word2vector.

2.2 Word Embeddings

We use "wiki-news-300d-1M.vec.zip" embeddings, load the first 80000 lines of the file and map every token to the corresponding embedding vectors. If a token does not exist in the vocabulary, embed this token as a 0-vector with the same dimension as the word embeddings.

3 Model

3.1 MLP

We use a simple multi-layer perceptron to score the similarity of the two sentences. Based on home exercise 06 model, we build a simple MLP with

random hyperparameter (number of hidden dense layers, dropout rate, activation function). We define the averages of the words embeddings to embed both sentences as the input of the fully connected layer. After the dropout and the regularization of BatchNormalization, the final output is trained.

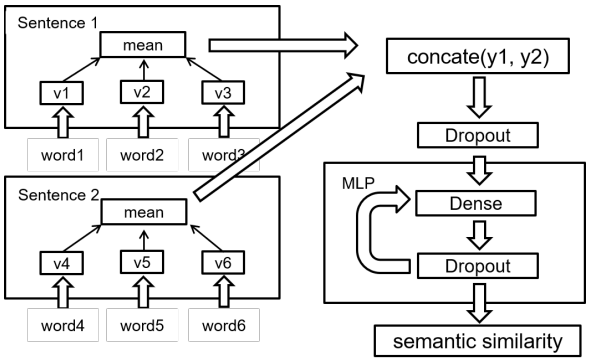

Figure 1: MLP

3.2 LSTM

Long short-term memory (LSTM) networks, are a special kind of RNN. This approach takes into account the semantic relationship between the context and the sentence, and can more accurately determine the semantic similarity.

Based on LSTM, we built (LSTM+MLP) network. In this approach, LSTM actually plays the role of encoder, inputting each word of the whole sentence in sequence, and the labeling corresponding to each word, so as to obtain the whole sentence word vector based on the word. Firstly we define embedding layer as the mapping layer of the input layer and the LSTM layer, and map the input sentence code into a list of word vectors as the input of the LSTM layer. The output of the two LSTMs is spliced and used as the input of the fully connected layer. After the Dropout and the regularization of BatchNormalization, the final output is trained.

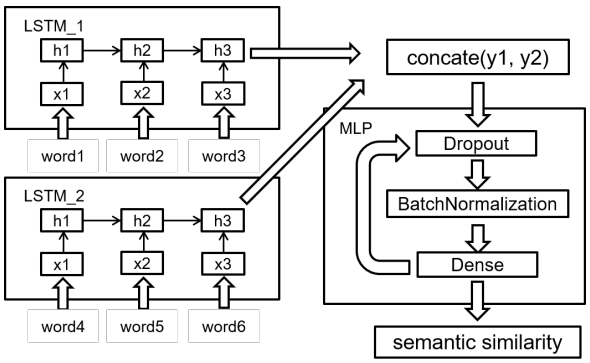

Figure 2: LSTM

4 Experiments & Results

4.1 Experiments Setup

• Preprocessing: load dataset, word vector convert and inverse attack

• Processing: deep learning network model based on keras API, MLP and pipeline (LSTM + MLP)

• Postprocessing: prediction the results of the trained model and save the result.

We artificially expand the training data, such as swapping the positions of two sentences, while retaining their relevance. On this basis, the data set in which the sentence has been changed can be regarded as a new set of data set. In addition, we merge training data and development data together as a new data set. In the internal verification phase of training, we use the test data of hex 06 as the verification set to detect the training result.

In the development phase, because the project was too late to start, many ideas were too late to be verified, so we only obtained a partial comparison of the results of the approaches: (see Table 1)

Method	Score
MLP	39.82%
LSTM	30.92%
Bi-LSTM	28.44%

Table 1: Scores of different models

After comparison, we found that MLP has better expressiveness in the estimation of semantic similarity, so in the subsequent test phase, we will only optimize the results to a certain extent for the MLP network model.

We set 5 activation functions, range of the hidden layer size and dropout rate. Each random search is implemented based on training data, and internal validation is done through development data, which ensures that the optimal parameters of random search are always applicable to the subsequent training. In this experiment, we conducted 10 sets of random search, each set of random search tested 40 different random combinations of hyperparameters, after completing all random search, compare the results and find the group with the smallest metric (mse), and determine it as the optimal parameter of the model.

In this process, we also tried to adjust the loss function of the model, namely mean absolute percentage error, mean squared logarithmic error, mean squared error. The results show that logistics

mse can quickly converge in the initial epoch of the model compared to ordinary mse. When the epoch is high, the convergence effect of mse is better than that of logistics mse. Therefore, our models will use mse as the loss function and metric.

4.2 Result

Through random search, we finally determined the following parameters of MLP (table 2):

Type	Value
Dense size	337, 171
Dropout rate	0.5, 0.1, 0.2
Activation	'relu', 'selu', 'selu'
Loss	MSE
Metric	MSE

Table 2: MLP hyperparameters

In the final test ranking, the MLP model based on random search to obtain hyperparameters has a final score of 29.60%. Compared with the baseline, the difference is not obvious.

5 Discussion

In the development phase, through comparison, the results of LSTM are better than Bi-LSTM. This is because our task this time is to compare the semantic similarity between sentences. All sentences are relatively simple English main sentences, and the reading order is from left to right. LSTM is more suitable for this kind of reading It is customary that the entire word vector decoded by the encoder is more meaningful than Bi-LSTM.

However, compared to searching the word vector of each word of the entire sentence and directly averaging it to obtain the entire sentence word vector, the effect of LSTM is unreasonable. As an encoder, you should directly perform a series of pre-training to make it have a certain labeling accuracy, and then connect it to the subsequent MLP, the result will be better in theory.

MLP shows a strong ability to estimate semantic similarity here. By observing each group of optimal parameters, we find that most of the optimal parameters only contain 2 layers of Dense. If artificially increasing the number of Dense layers, it will lead to model complexity Increase, the dimension of the model mapping increases (in other words, the nonlinearity of the mapping will increase), thus causing the problem of overfitting.

The data set obtained through the reverse attack can reduce the generation of OOV to a certain extent, thereby optimizing the estimation of semantic similarity. In an ideal situation, a suitable reverse transformation can completely recover the impact of adversarial attacks. In this case, the estimation of semantic similarity will become extremely simple

6 Conclusion

Through a one-week project, we demonstrated the potential of MLP in handling semantic similarity issues. For adversarial attacks, MLP has a certain robustness, but the effect is not ideal. We found that the reverse attack can help to improve the estimation of semantic similarity, and can further improve the stability of the model based on the existing model.

References

[1] Deep Learning for NLP 2020 Shared Task, May 29, 2020

[2] Steffen Eger, Gözde Gül ¸Sahin, Andreas Rücklé , Ji-Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, Iryna Gurevych Text Processing Like Humans Do: Visually Attacking and Shielding NLP Systems, 2019, Department of Computer Science, Technische Universität Darmstadt