Relevant Question Detection using BERT and Deep Neural Network Classifier

First Author yezhang @umass.edu

Second Author yuanjarwang @umass.edu

Third Author kaiyuandeng @umass.edu

Fourth Author yuxinhuang @umass.edu

1 Problem statement

Our project will attempt to apply BERT and Deep Neural Network Classifier on the task of Community Question Answering (cQA): given an input question, we want to find the most relevant answers from corpora of existing questions. This task can be further divided into evaluating question-question similarity and question-answer similarity. In our project, we will focus on evaluating question-question similarity which involves detecting existing relevant questions given a new input question. Our implementation will be : given an input question pair, classify it as "Relevant" or "Irrelevant". We believe that this task has practical significance on not only Q&A fora but also Q&A web pages of companies. When receiving new questions from users, we can immediately provide corresponding answers extracted from existing relevant questions, instead of inefficiently posting similar questions again and wasting both respondents' time and the site's storage.

2 What you proposed vs. what you accomplished

- ~~Collect and preprocess dataset~~
- ~~Data exploration~~
- ~~Data balancing~~
- ~~Implement baseline algorithm~~
- ~~Implement similarity measurements~~
- ~~Apply BERT word embeddings~~
- ~~Build and train Neural Network Classifier~~
- ~~Tune hyperparameters of Neural Network Classifier~~
- Tune BERT word embeddings: Since we spent plenty of time on tuning hyperparameters of Neural Network Classifier, we decided to use pre-trained BERT word embeddings for saving time.
- ~~Implement evaluation metrics and evaluate baseline algorithm & proposed approach~~

3 Related work

The researches of community Question Answering(cQA) has emerged since this problem was proposed as a task in SemEval 2016(Nakov et al., 2016) and 2017(Nakov et al., 2017). Also, in SemEval 2017, the stackexchange dataset was provided, which becomes a very popular cQA datatset.

Before some good pre-trained language model(like BERT) were proposed, the most of prior works on computing semantic similarity involved complex pipelines and manual feature engineering. For instance, KeLp (Filice et al., 2016) uses similarity features like cosine similarity, Jaccard coefficient, parse tree of sentences, etc., then uses Kernel based SVM to do binary classification. In addition to traditional NLP features, (Feng et al., 2017) introduces several neural network based matching features which enable system to measure text similarity beyond lexicons. In (Koreeda et al., 2017), they combine neural similarity features and hand-crafted comment plausibility features, and model inter-comments relationship using conditional random field. In the (Gonzalez et al., 2018), they present a strong baseline for question relevancy ranking by training a simple multi-task feed forward network on a bag of 14 distance measures for the input question pair, which is fast to train and uses only language-independent features. Instead of manually designed the features, some works are

tring to extract features by neural network. For instance, the DIIN(Gong et al., 2017) model uses encoders to encode both the sentences and uses an interaction layer on top of it which is fed into a feature extraction layer. Finally the output layer decodes the acquired features to give predictions.

Besides these supervised learning methods, there are also some unsupervised learning methods such as (Rücklé et al., 2019) and (Yang et al., 2018). These methods aim to minimize the cost of obtaining large amounts of labeled question pairs. In (Yang et al., 2018), they propose a novel approach to sentence-level semantic similarity based on unsupervised learning from conversational data. The assumption is that semantically similar sentences have a similar distribution of potential conversational responses, and that a model trained to predict conversational responses should implicitly learn useful semantic representations. Rather than training on conversational data, (Rücklé et al., 2019) use automatic generated duplicate questions or weak supervision using the title and body of a question to train, both of them achieve good performances even though they do not require any labeled data.

After the BERT was innovated, several works are done, based on BERT, on this Quora Question Pairs task with best performance such as MT-DNN (Liu et al., 2019) and AUTOHOME-ORCA(Lv et al., 2019). With MT-DNN's model incorporating a pre-trained bidirectional transformer language model similar to BERT (Devlin et al., 2019) while the fine-tuning part is leveraging multi-task learning. For the AUTOHOME-ORCA model, the predictions of several variants of BERT model encoding the meta information(e.g the length of questions) are combined to create an ensemble model.

Our approach is using pre-trained BERT and a multiple layer perceptron(MLP) to classify duplicate questions. The good thing is that it doesn't need any feature engineering. Since we have transferred to Quora Question Pairs dataset, which contains less meta information than the stackexchange ones, we didn't introduce any meta information into our classifier.

4 Your dataset

We collected two datasets for training our model : (1) CQADupStack (Link : shorturl.at/bmBZ1):

This dataset is provided in SemEval 2017 for subtask E of task 3 which is "Multi-Domain Duplicate Question Detection". This dataset collects threads from twelve StackExchange subforums and here is an example from it:

Figure 1: Data example of CQADupStack

Since the original dataset size is too huge (50GB), we only sample 1/5 of the original dataset to perform data exploration. First, we calculate the numbers of relevant questions & irrelevant questions and find out it is a highly unbalanced dataset which might prevents our model from actually learning difference between positive and negative cases.

Data \ Counts	Relevant	Irrelevant	Total
StackExchange	4,535	682,115	686,650

Figure 2: Counts for both labels of CQADupStack

Second, we compute distributions of length of input text sequence when using body text (OrgQuestionBodyText + RelQuestionBodyText) and topic text (OrgQuestionTopicText + RelQuestionTopicText) respectively. We find out there are about 15% of input text sequences very likely to have lengths longer than BERT's limit (512 tokens) and more than 95% of input text sequences have lengths longer than 100 tokens when using body text. In the other hand, all of the input text sequences have lengths lower than 100 tokens when using topic text.

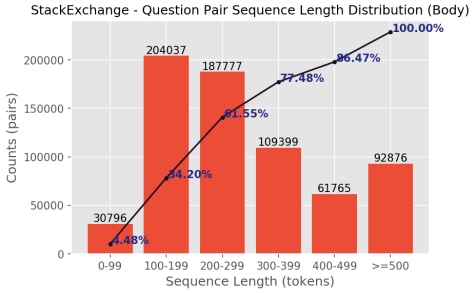

Figure 3: Distribution of input text sequence length (Body Text)

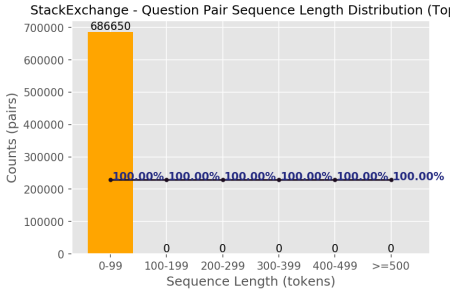

Figure 4: Distribution of input text sequence length (Topic Text)

(2) Quora Dataset (Link : shorturl.at/gilGV):

This dataset is provided on Kaggle for Quora's prediction competition in 2017. Which includes genuine examples from Quora, and here are some examples from it :

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to	What is the step by step guide to inv	0
1	3	4	What is the story of Kohinoor (K	What would happen if the Indian gov	0
2	5	6	How can I increase the speed of	How can Internet speed be increased	0
3	7	8	Why am I mentally very lonely?	Find the remainder when [math]23^{	0
4	9	10	Which one dissolve in water quil	Which fish would survive in salt wat	0
5	11	12	Astrology: I am a Capricorn Sun	I'm a triple Capricorn (Sun, Moon ar	1

Figure 5: Data examples of Quora Dataset

For data exploration, we compute numbers of relevant questions & irrelevant questions first as above and find out Quora dataset is a much more balanced dataset.

Data \ Counts	Relevant	Irrelevant	Total
Quora	149,263	255,027	404,290

Figure 6: Counts for both labels of Quora Dataset

Second, only topic text is provided in Quora dataset, we can only compute distribution of length of input text sequence when using topic text (OrgQuestionTopicText + RelQuestionTopicText). We find out more than 99.9% of input text sequences have lengths less than 100 tokens.

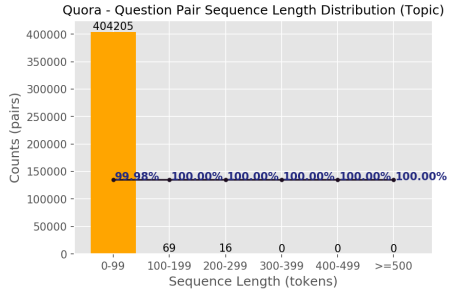

Figure 7: Distribution of input text sequence length (Topic Text)

Based on our data exploration, we decide to use topic text instead of body text in our input text sequence for BERT. Because some of input text sequences will be discarded due to token limit of BERT and longer input text sequences will increase computational burden when using body text. Then we decide to use only Quora dataset for this project because it's much more balanced than CQADupStack.

4.1 Data preprocessing

(1) Data cleaning : Because CQADupStack is provided in the format of XML, it is very large in size and hard to be reused. We use "xml" package to parse the XML tree structure for extracting useful information such as topic text, body text, and label first. Second, we store these information into dictionaries and use "pickle" package to store these dictionaries in a binary file. For Quora dataset, we perform the same preprocessing in order to keep consistency.

(2) Data splitting : We split Quora dataset into train, validation, and test datasets with the ratio 80:10:10.

5 Baselines

For the baseline algorithm, we apply straightforward word matching technique. First, we clean the text of question topic by removing punctuation, stop words from it and apply stemming on it. Second, we convert processed text into bag-of-words representation by using space split. Finally, we use this bag-of-words representation to calculate Jaccard Similarity within each question-question pair and classify each pair to be relevant or irrelevant by setting a threshold which is a hyperparameter. For the evaluation, we compute F-1 Score, Precision, Recall, and Accuracy of predicted labels which are shown in "Evaluation" section later.

We choose to use word matching technique because it's very intuitive that questions have more words in common might be more likely to describe similar or same topics.

We tune our threshold on training dataset by applying grid search in the range from 0.01 to 1. We iterate through 500 different thresholds and compare their F1-Score on training dataset. Finally, we have threshold = 0.3016 with the highest F1-Score on training dataset = 0.6439. We also visualize how F1-Score, Precision, and Recall vary according to different thresholds below :

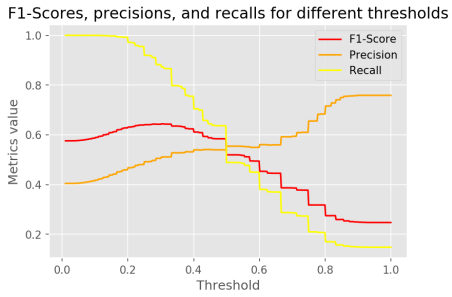

Figure 8: F1-Score, Precision, Recall for different thresholds during training

6 Your approach

Model For our proposed approach, we use BERT + Neural Network Classifier. First, we concatenate two topic texts in each question pair with special tokens to form input text sequences for BERT which look like : [CLS] question 1 topic text [SEP] question 2 topic text [SEP]. Second, we feed these input text sequences into pre-trained BERT to get embeddings. Third, we extract CLS embeddings to represent each question pair's sequence embeddings. Finally, we feed these sequence embeddings into our Neural Network Classifier to perform binary classification. For the cases in which our baseline algorithm fails, we expect our proposed approach to perform much better since our sequence embeddings include more information between two input question topics apart from common words which might helps improve classification performance.

Implementation The sequence encoding module, the pre-trained base BERT, was directly imported from *transformers* libraries. The classifier was implemented as a neural network with one fully-connected hidden layer using pytorch. We wrote our own input conversion function and batch data loader which could use as a data stream pipeline from raw data to BERT encoder. We also wrote a training process which would automatic save history and model checkpoints to google drive, and could continue training from any checkpoint even if the training was interrupted accidentally.

Optimizer We have tried three different pytorch build-in optimizers: SGD, Adam(Kingma and Ba, 2014), and AdamW(Adam with fixed weight decay). Here's what we found: SGD is much slower than Adam and AdamW, also SGD is harder to tune(i.e. hard to find a good learning rate). Between Adam and AdamW, Adam is easier to stuck at flat area while the AdamW could get over flat point within less epochs and keep improving performance. Therefore, we selected AdamW as our final optimizer.

Hyperparameters For the hyperparameters tuning, we use random search. The hyperparameters we tuned include: learning rate, batch size, hidden layer size, weight decay of AdamW, and the max length of sequence fed into BERT. The search space is defined as table 1

hyperparam	candidate set	distribution
learning rate	[0.00001, 0.0005]	uniform
batch size	{32, 64, 100}	uniform
hidden size	{100, 200}	uniform
max$_l ength$	{128, 512}	uniform
weight decay	{0.01, 0.001}	uniform

Table 1: Search space of hyperparameters

Each configuration of hyperparameters will be run 10 epochs, then we compare the validation loss among them. Finally, our best training configuration is: $learning\_rate = 0.000472$, $batch\_size = 100$, $hidden\_size = 200$, $max\_length = 128$, $weight\_decay = 0.001$. Some things we found: There's no much difference among 0.0003 to 0.0005 for learning rate. For the weight decay, 0.001 is better than 0.01. For the max length parameter, since we are using quora data set which has short text, set limit to 128 is better than 512. If we set max length to 512, the model would longer time to train, also the performance would decrease. The possible reason is that there are many '[PAD]' tokens in the sequences. The figure 9 shows the difference between hidden size.

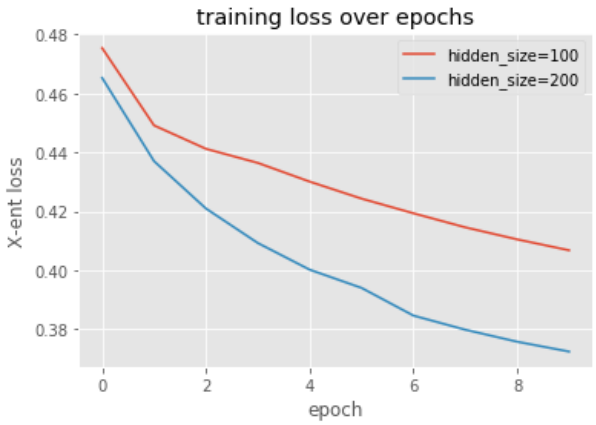

Figure 9: Training loss of models with different hidden size

Training For the training process, we firstly split data set into 80:10:10 as training, validation, and test set. Then, we train on the training set and calculate the cross-entropy loss on validation set after each epoch, once the validation loss stop improving for a few epochs, we early stop the training process to avoid overfitting. Each epoch takes around 4000 seconds to run. The learning curves is showed in figure 10 and 11. As shown in the figure 11, the model is overfitting after 15 epochs. We use test set to evaluate the model at 15, 20, 25, 30 epochs to see whether the assumption of overfitting is true or not. The evaluation results shows in table 2. It seems like the model didn't overfit according to the tabel.

epochs	f1	precision	recall	accuracy
15	0.7450	0.7694	0.7222	0.8188
20	0.7295	**0.7973**	0.6723	0.8172
25	0.7681	0.7651	0.7712	**0.8293**
30	**0.7756**	0.7391	**0.8159**	0.8269

Table 2: Evaluation(on test set) at different epochs

Tools and Libraries We wrote all out code on Colab. We imported *transformers*[1] to get pretrained BERT tokenizer and BERT model(*Note that we used version 2.1.1, since there was a bug for the bert tokenizer in version 2.2.1 when we used it*). We used pickle as out file I/O tool. Pytorch was used to define and train our classifier. We saved all our training history and model checkpoints into our google drive. For the results analysis, matplotlib and numpy were used to plot figures and compute measure metrics.

Some tricks For the colab, we wrote a main notebook which contains all basic functions such as data loader, input conversion, training process and so on. Then every member make a copy of main notebook, so that we could do different things at the same time without affecting others work. For example, we could run training process with different configs at the same time. At first, our training was extremely slow, then we found out the reason is that we were running it on CPU. After changed the notebook settings and moved all our model into GPU, the speed was increased by over 100 times.

Also, we use google drive to share data so that we could directly download data into our colab notebook's pydrive via share link, which helps us save our local disk memory.

[1]https://github.com/huggingface/transformers

Figure 10: Training cross-entropy loss over each 100 batches

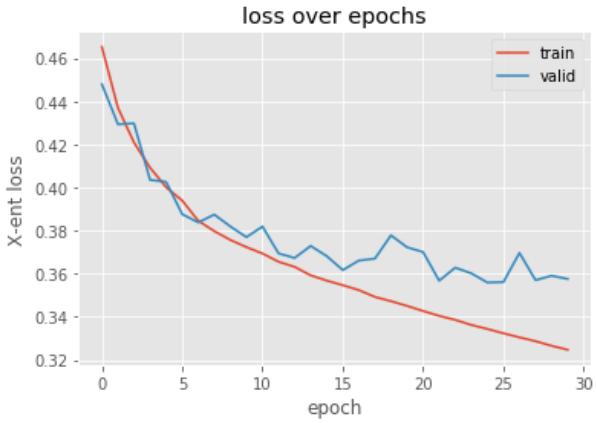

Figure 11: Training cross-entropy loss over epochs

Issues Although we have used GPU provided on google colab to speed up our training process, we still take around 6 hours to train with one combination of hyperparameters which makes us hard to tune hyperparameters well. Also, we have encountered memory issues that we are not able to load the whole training dataset in the same time and are not able to convert too many input text sequences into BERT embeddings in the same time (The maximal batch size is about 32). Which makes us have restricted batch size and might not be able to train very stably.

7 Evaluation

For the evaluation, we calculate 4 measure metrics: F1 score, precision, recall, and accuracy. The table 3 shows the final results. According to the table, our approach outperformed the baseline model except at recall rate. We'll analysis the rea-

son why this happened at the next section.

model	F1	precision	recall	accuracy
baseline	0.6369	0.5017	0.8719	0.6356
NN	0.7756	0.7391	0.8159	0.8269

Table 3: The performance of models

8 Error analysis

8.1 Case Study

We took 200 samples from test set and analyzed the error cases of both model. The statistical results is listed in table 4. There are 55 cases out of 200 where NN corrects and base model fails, 12 cases in the opposite way, and 24 cases both of them fails.

model	FP	FN
base	70	9
NN	25	11

Table 4: Error cases(FP: false positive, FN: false negative)

The figure 12, 13 and 14 show some sample cases. For every kind of cases, we selected five samples. According to statistical analysis, most of cases where Bert+NN model gives correct prediction while the baseline doesn't, are "not duplicated" samples. This means baseline is lean to predict more positive("duplicated") labels than our models. The intuitive explanation is that the threshold of Jaccard similarity was set too low. This conclusion also was reflected in the fact that baseline has higher recall. As we can see in the first 3 examples in the figure 12, the word matching methods would give out positive prediction while there is a big part of question pairs is overlapping. However, two questions might have totally different meaning with a few word changed or added. For example, in the first case, there's only one word different between Q1 and Q2, but they are asking different companies. For the duplicated cases, the word matching fails because two questions are using different words to express the same meaning. We think the BERT+NN model can survive from these cases is attributed to BERT could encode semantic features rather than only stay at lexical layer.

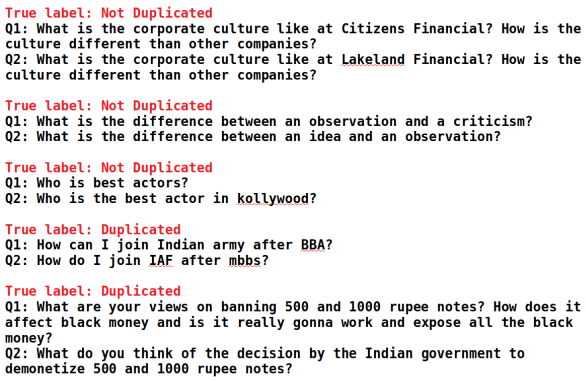

Figure 12: Sample cases where baseline model fails and our model successes

The figure 13 shows five sample cases where BERT+NN performs worse than the word matching. There are more "duplicated" samples in this case. Those question pairs have a lot of overlapped words. However, we are not sure why our model would fail on them.

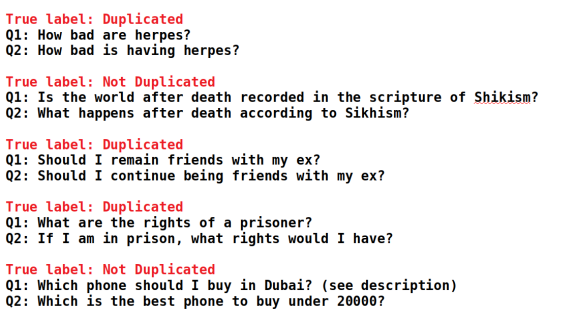

Figure 13: Sample cases where baseline model successes and our model fails

Figure 14 shows the cases where both models fail. We think some of them were mislabelled by human, for example, the first 2 question pairs should be labelled as "duplicated" in our opinion. For the rest of them, we think these might be difficult cases or some "strange" cases. Some of them are difficult to classify even for human being.

9 Contributions of group members

- Ye Zhang: Apply BERT embeddings / Write data pipeline/ Build and train Neural Network Classifier / Tune hyperparameters of Neural Network Classifier / Implement evaluation metrics and evaluate baseline algorithm and proposed approach / Poster presentation / Write final report

```
label: Not Duplicated
Q1: Do you have any creative and innovative ideas for a mobile app?
Q2: Do you have any innovative ideas for a mobile app?

label: Not Duplicated
Q1: How can I develop a sense of humour?
Q2: How can I get a good sense of humour?

label: Not Duplicated
Q1: What is Kerbal Space Program?
Q2: What is your review of Kerbal Space Program?

label: Duplicated
Q1: What is it like to be in a mental hospital?
Q2: What was your experience in a psychiatric hospital?

label: Duplicated
Q1: What are movies that everyone should see?
Q2: Are there any good movies to watch?
```

Figure 14: Sample cases where both models fail

- YuanJar Wang: Data collecting / Data preprocessing / Data exploration / Implement baseline algorithm / Tune hyperparameters of Neural Network Classifier / Implement evaluation metrics and evaluate baseline algorithm and proposed approach / Make poster / Write final report
- KaiYuan Deng: Data preprocessing / Implement similarity measurements
- YuXin Huang: Data collecting / Data exploration

10 Conclusion

Takeaway We learn how to build everything from scratch in this project and get familiar with lots of packages used frequently in NLP field. Also, we learn how to deal with problems that might be encountered in reality such as computational resource limit, storage limit, messy dataset ... etc.

Thoughts When we were doing homework, we were given starter code and just needed to fill codes in specified lines. However, we have to implement everything from scratch in this project. A surprisingly hard part is to design a great pipeline of our whole project to connect everything perfectly especially when we have to collaborate with others. We are not that surprised by our results but we think there are still lots of future works for us to improve our proposed model.

Future Work

- Train our model on unbalanced data and compare the results.
- Use different ways to generate sequence embeddings of question pairs.
- Make use of question body text and other metadata to improve performance.

References

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Feng, W., Wu, Y., Wu, W., Li, Z., and Zhou, M. (2017). Beihang-MSRA at SemEval-2017 task 3: A ranking system with neural matching features for community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 280–286, Vancouver, Canada. Association for Computational Linguistics.

Filice, S., Croce, D., Moschitti, A., and Basili, R. (2016). KeLP at SemEval-2016 task 3: Learning semantic relations between questions and answers. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1116–1123, San Diego, California. Association for Computational Linguistics.

Gong, Y., Luo, H., and Zhang, J. (2017). Natural language inference over interaction space. *CoRR*, abs/1709.04348.

Gonzalez, A., Augenstein, I., and Søgaard, A. (2018). A strong baseline for question relevancy ranking. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4810–4815, Brussels, Belgium. Association for Computational Linguistics.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Koreeda, Y., Hashito, T., Niwa, Y., Sato, M., Yanase, T., Kurotsuchi, K., and Yanai, K. (2017). bunji at SemEval-2017 task 3: Combination of neural similarity features and comment plausibility features. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 353–359, Vancouver, Canada. Association for Computational Linguistics.

Liu, X., He, P., Chen, W., and Gao, J. (2019). Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy. Association for Computational Linguistics.

Lv, Z., Liu, D., Sun, H., Liang, X., Lei, T., Shi, Z., Zhu, F., and Yang, L. (2019). Autohome-orca at semeval-2019 task 8: Application of bert for fact-checking in community forums. In *SemEval@NAACL-HLT*.

Nakov, P., Hoogeveen, D., Màrquez, L., Moschitti, A., Mubarak, H., Baldwin, T., and Verspoor, K. (2017). SemEval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 27–48, Vancouver, Canada. Association for Computational Linguistics.

Nakov, P., Màrquez, L., Moschitti, A., Magdy, W., Mubarak, H., Freihat, A. A., Glass, J., and Randeree, B. (2016). SemEval-2016 task 3: Community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 525–545, San Diego, California. Association for Computational Linguistics.

Rücklé, A., Moosavi, N. S., and Gurevych, I. (2019). Neural duplicate question detection without labeled training data. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1607–1617, Hong Kong, China. Association for Computational Linguistics.

Yang, Y., Yuan, S., Cer, D., Kong, S.-y., Constant, N., Pilar, P., Ge, H., Sung, Y.-H., Strope, B., and Kurzweil, R. (2018). Learning semantic textual similarity from conversations. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 164–174, Melbourne, Australia. Association for Computational Linguistics.