Improving Commonsense Validation Using External Knowledge

Anonymous ACL submission

Abstract

Sen-Making is a task that aims to evaluate the sense making cability of a system. In this paper, we propose a method that combines the fine-tuning based and knowledge based approaches for Sen-Making task, which effectively utilizes state-of-the-art pre-trained language models and ConceptNet that provides our model with external knowledge. The accuracy of our method on testset is 87.80%. We also present an analysis showing the differences between BERT and XLNet for this task and compare our method with KagNet.

1 Introduction

Commonsense reasoning is a field of artificial intelligence that aims to help computers understand and interact with people more naturally by finding ways to collect assumptions and teach them to computers. It is useful in several applications of Natural Language Processing, such as Machine Translation, Speech Recognition, Question Answering, and so on. However, it is still a quite challenging task. While it is easy for human beings to judge whether a statement is logically valid or not with commensense knowledge, it remains difficult for machines to tell right from wrong.

To help evaluate whether a system has a sense making capability, a testset consisting of pairs of natural language statements with similar wordings was created (Wang et al., 2019a). This dataset is made up of sentence pairs, in which one of them is more reasonable (*e.g. Sentence 1: He put a turkey into the fridge. vs. Sentence 2: He put an elephant into the fridge.*) Our task is to distinguish statements that make sense from those that do not make sense.

In this paper, we improve the accuracy of the Sen-Making task by proposing a method that combines the fine-tuning based and the knowledge based approaches. First, we use fine-tuning-BERT that learns the bidirectional contexts. Second, we add ConceptNet as an external knowledge source by adopting Graph Convolutional Network to encode the schema graphs and using bi-LSTM module to capture the information between different entities for our knowledge based part. Then, we combine the score of these two parts, weighted by λ and (1-λ), respectively. Our method achieves an accuracy of 87.80%.

2 Related Works

Our work involves commonsense reasoning, NLP tasks that use neural networks to introduce external knowledge, and pre-trained language models. We briefly describe each of these fields in this section.

2.1 Commonsense Reasoning

There exist a few large-scale datasets that aim to evaluate machine's commonsense reasoning ability. SWAG is a dataset for studying physically grounded commonsense inference, specifically, the task is to predict which event is most likely to occur next in a video (Zellers et al., 2018). ATOMIC, organized in if-then relation types,focuses on sequences of events and the social commonsense relating to them (Sap et al., 2019). Among all, CommonsenseQA (Talmor et al., 2018), a dataset of multiple-choice questions based on knowledge encoded in CONCEPTNET (Speer et al., 2017), is a similar dataset to the dataset of Sen-Making task. The best baseline of CommonsenseQA achieves an accuracy of 56% using BERT-large (Devlin et al., 2018). Commonsense Auto-Generated Explanation (CAGE) framework, with a dataset named Common Sense Explanations (CoS-E) that consists of human explanations for commonsense reasoning, increases the accuracy of CommonsenseQA task to 72% (Rajani et al., 2019). Taking this as an example, the introduction of external knowledge plays a decisive role in similar tasks and data sets.

2.2 External Knowledge

Knowledge Graph, defined as a set of concepts connected by relationships where the concepts form the nodes of the graph and the relationships are the labeled edges, has been widely used in fields like question answering and recommendation systems (Wang et al., 2019b). Some of the well known KGs include Freebase, DBpedia, Yago, and ConceptNet. ConceptNet is a freely-available semantic network that includes commonsense knowledge acquired through crowd sourcing. Utilizing a large knowledge base like conceptnet ensures that the entity or concept being queried is closely related to commonsense relations and that the common sense domain associated with it is comprehensive (Stocks et al., 2019). It is reported that ConceptNet may be more appropriate if commonsense reasoning is required in a task. In this work, we use ConceptNet as our external knowledge database.

2.3 Pre-trained Contextual Representations

Recent work including Embeddings from Language Models (ELMO) and Bidirectional Encoder Representations from Transformers (BERT) give words different embedding vectors based on the context, being able of modling different word senses in different contexts, which can be used as features or fine-tuned for downstream tasks including question answering, sentiment classification, etc (Devlin et al., 2018). For example, the Generative Pre-trained Transformer (GPT) and BERT introduce minimal task-specific parameters, and can be easily fine-tuned on the downstream tasks with modified final layers and loss functions (Stocks et al., 2019). In this work, we try to add a linear layer and softmax as the output layer to implement our task-specific fine-tuning.

3 Methodology

3.1 Validation Based on Sentence Perplexity

We start with a method without training. For every sample in the testset, we calculate the perplexities of both statements and pick the one with lower score as the correct statement. We choose BERT (Devlin et al., 2018), a state-of-art language model to complete this task. The details of calculating perplexity are as follows:

In the case of traditional language models(such as n-gram), given a sentence represented by $s = \omega_1, \omega_2, ...\omega_N$, the probability of the sentence can be calculated with chain rule:

$$p(s) = \prod_{k=1}^{N} p(\omega_k|\omega_1, \omega_2...\omega_{k-1})$$

However, since BERT is a bidirectional language model, the equation of sentence probability can be approximated as:

$$p(s) = \prod_{k-1}^{N} p(\omega_k|\omega_1, ...\omega_{k-1}, \omega_{k+1}, \omega_N)$$

That is, we mask each token in a sentence in turn, use the rest to predict the probability of this token, and eventually multiply these probabilities together to get the probability of the whole sentence. Then the perplexity can be represented as:

$$PP(s) = \sqrt[N]{\frac{1}{p(s)}}$$

3.2 Validation Based on Finetuning

We take this task as an answer selection or multiple choice task and implement BERT finetuning. More specifically, we take both statements in each sample as the input to the pre-trained BERT model, which is aimed to output the probabilities that each statement is correct. Each statement is divided into a token sequence with a $[CLS]$ and a $[SEP]$ at beginning and end respectively, which can be represented as: ([CLS], ω_1,..., ω_N, [SEP]) . Then the tokens are converted to corresponding indexes which are prepared for pre-trained embeddings. The final hidden state C of the first token $[CLS]$, which is the aggregate representation of the whole sequence, is used for classification at sequence level. During finetuning, parameters are added for a linear classification layer W. All the parameters of the pre-trained BERT model along with W are fine-tuned jointly to maximize the probability of correct labels. The final output can be represented by:

$$P = \sigma(CW^T)$$

where $\sigma(x)$ is $\frac{1}{1+e^{-x}}$, aiming to confirm the sum of the two probabilities equals 1. The complete procedure of finetuning with pre-trained BERT model is as follows:

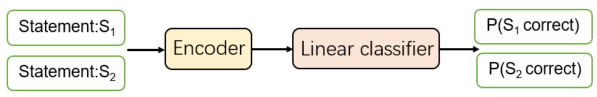

Fig1. The overall workflow of fine-tuning framework with pre-trained model.

The encoder serves as the function of dividing sentences into token sequences and obtaining pre-trained embeddings of tokens as input features, which are supposed to be input into a linear classifier to learn the parameters and output the probabilities that sentences are correct.

We also implement finetuning on this dataset with XLNet. The main difference is that special tokens $< sep >$ and $< cls >$ are added at the end of statements. Beyond that, there are no other changes to the finetuning process.

3.3 Utilizing External knowledge

3.3.1 Commonsense Knowledge Database

The commonsense database we investigate in this experiment is ConceptNet (Speer et al., 2017), which provides a large semantic graph that describes general human knowledge and how it is expressed in natural language. We download the *conceptnet-assertion-5.6.0.csv* from `http://www.conceptnet.io/`. The origional database has more than 9 GB of data, with dozens kind of languages and more than 47 types of relationships. Since this task only involves English text, we delete all the data that does not contain English relationships, and only retain information of relational triples *(head, relation, tail)* and corresponding weight. That is, after data cleaning, each piece of data in this commonsense database is formatted as follows:

$$(relation, head, tail, weight)$$

We observe a significant difference in the number of occurrences of each type of relation in the knowledge database. For example, the relation "$relocatednear$" appears only 49 times in the knowledge database, while "$atlocation$" appears 27,746 times. So we decide to merge relations that have similar meanings. Besides, for relations with opposite meanings, such as "hasa" and "partof", we swap the head and tail of "hasa", so that all data with relation "hasa" can be incorporated into data with relation "partof". After the step of merging relationships, the total capacity of the whole knowledge database is about 90MB and 2,700,000 pieces of information with 17 types of relations, which allows us to query relations from it more quickly.

3.3.2 Entity Finding

We utilize a simple method to find every entity in each sentence. We divide a sentence into unigram token sequences. For example, a sentence "*he put an elephant into the fridge*" will be divided into $[he, put, an, elephant, into, the, fridge]$. Besides, we adapt lemmatization to convert a token into its general form(e.g. *ate —> eat*). To achieve this, we use the pos_tag function from package $nltk$ and implement lemmatization for each token according to its tag. To further reduce noisy token which has little meaning, we establish a set for stopwords such as *the, he, is, etc*. Since entities in ConceptNet can appear as bi-gram or tri-gram, we also split a sentence into bi-gram and tri-gram token sequences. Our method of finding the real entities from these sequences is naive: If this token appears in ConceptNet, then it is an entity; otherwise not.

3.3.3 Relation Extraction

After we get entities from a sentence, we can find the path between each two entities. The path length is defined as the number of other entities contained in it. For example, a sentence has entities $S = [a_1, a_2, a_3]$. Given the limits of max length of path, we retrieve the paths from a_1 to a_2, a_1 to a_3, a_2 to a_3. Then we use these paths to construct a schema graph for this sentence. The example is shown below:

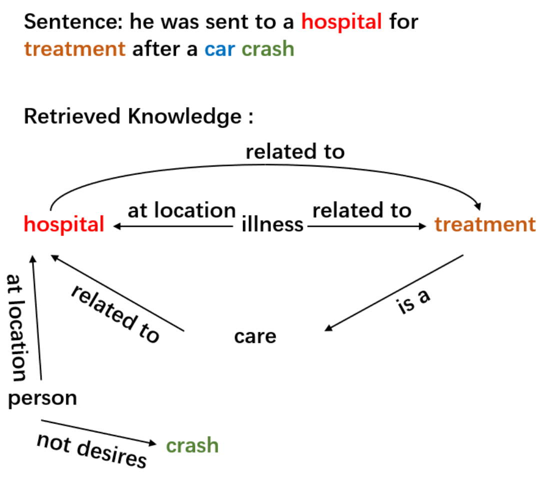

Fig2. Example: The construction of schema graph with paths extracted from ConceptNet.

We assume it inefficient to retrieve paths between each two entities in the sentence since given a pair of sentences from dataset, the number of different entities may be only one or two. Thus, we name the different entities from the two sentences "target entity" T_1, T_2, and the same entities "source entity" S. We only retrieve the paths from a source entity in S to a target entity in T_1 to construct the schemic graph for sentence 1; And we follow the same rule to construct schema graph for sentence 2. In this way, we can better concentrate the difference of two schema graphs and measure the plausibility of different entities in relation to the same set of entities.

3.3.4 Path Pruning

We aim to reduce the noise of schema graphs by scoring each path according to the common ground criteria and pruning based on these scores. Although ConceptNet provides an origional weight for every relational triple, we do not score paths according to these weights. Instead, We re-implement and train TransE model (Han et al., 2018), which is a simple but effective Knowledge Graph Embdedding(KGE) technique. we utilize TransE to train the embeddings of each entity and relation, then score a relational triple by the cosine distance based on the pre-trained embeddings. We borrow the codes from `https://github.com/thunlp/OpenKE`, and implement it on ConceptNet for 10000 iterations. Then we score every path we extract from ConceptNet based on the pre-trained embeddings. If the score of a path falls below a fixed pruning threshold, it would be deleted. We use and compare multiple pruning thresholds in experiments.

3.4 Augmenting Finetuning with Knowledge

Since dataset of this task is limited, we expect to introduce more unsupervised data relevant to our dataset. We believe that the key to this task is to introduce and let the model learn more prior knowledge. The more prior knowledge the model obtains, the higher the probability that it will give the correct result. Therefore, we utilize extracted relational paths of sentences firstly to augment the fine-tuning method. For each sentence, we choose the top three paths that score the highest and add them as new explanations to the end of this sentence. More specifically, every relational triple would be changed into a more natural representation. For example, based on some simple fixe rules, a relational triple*(illness, relatedto, treatment)* is converted to the form of natural language: "*illness is related to treatment*".Then it is added to the end of original sentence *he was sent to a hospital for treatment after a car crash.* After explanations are added to the dataset, we do the same thing as the fine-tuning. In the experiments(4.1), we set the max path length as 2 and select the top 3 paths with highest scores for every sentence as new knowledge to add to the original dataset.

3.5 Knowledge Graph Network

Given a statement S, we define our model to calculate the probability that statement S is correct. The whole architecture is illustrated in Fig 3.

We utilize a deep learning model consists of graph convolutional networks(GCN) and bi-LSTM module. We use GCN to encode the schema graphs, training the embeddings of entities and relations pre-trained by TransE with particular schema graphs. The bi-LSTM is aimed to capture the information between two entities. In final, the output of LSTM is fused with the output of finetuning framework.

In summary, the final output of our model is the combination from the output of finetuning framework(f_1) and the knowledge graph network(f_2):

$$P(S) = \lambda f_1(S|text) + (1-\lambda) f_2(S|knowledge)$$

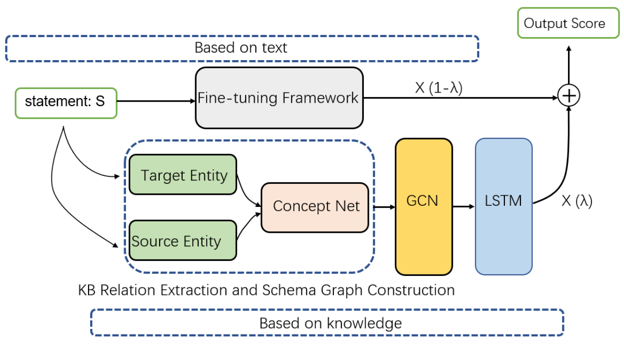

Fig3. The overall framework of scoring system.

3.5.1 GCN

Graph convolutional networks (GCNs) encode graph-structured data by updating node vectors via pooling features of their adjacent nodes (Kipf and Welling, 2017). In this case, we use GCN to continue training embeddings of entities and relations pre-trained by TransE so that ambiguation could be removed to some extent (Lin et al., 2019). To effectively uilize the information from multi-hop relations (Schlichtkrull et al., 2018), we construct unlabeled, non-directional schema graph as the origional input to GCN. The update rule we adapt is from work of kagnet (Lin et al., 2019), represented

Model	dev-Acc.(%)	test-Acc.(%)
Random	50.0	50.0
Trigram	53.9	52.6
Perplexity-BERT-base	73.0	69.8
Perplexity-BERT-large	79.9	75.5
FT-BERT-large	91.14	85.17
FT-XLNet-large	91.65	87.70
FT-XLNet-large+GCN+LSTM	91.71	**87.80**

Table 1: Experiments Results of Different Methods

by

$$h_i^{(l+1)} = \sigma(W_{self}^{(l)} h_i^{(l)} + \sum_{j \in N_i} \frac{1}{N_i} W^{(l)} h_j^{(l)})$$

where the embdedding of concept i at $(l+1)$-th layer is updated by itself at l-th layer and its neighbors' embedding at $(l+1)$-th layer with sigmoid function.

3.5.2 Relational Path Encoding

In this part, we utilize work of kagnet (Lin et al., 2019) and use LSTM to encode each path as a sequence of relational triples which is composed of the concatenation of corresponding vectors of *(head,relation,tail)* We also simplify the relational representations of a pair of target entity i and source entity j as:

$$T_{i,j} = MLP([c^{(i)}; c^{(j)}])$$

where $c^{(i)}$ and $c^{(j)}$ are the embeddings of i and j. The rest of the module is modeled after work of Kagnet.

4 Experiment

In this section, we present the baseline methods introduced by (Wang et al., 2019a), and go further with some experimental results. In the experiments, we implement traditional methods of calculating sentences' probabilities(e.g. trigram) as well as state-of-art pre-trained language models(e.g. BERT and XLNet). Our best result is achieved by the finetuning-XLNet-large with graph network, reaching 87.80% on the testset. The results of all methods we implement are listed in Table 1 and Table 2.

4.1 Comparison between BERT and XLNet

In this section, we compare the performance of finetuning between two outstanding pre-trained language model, BERT (Devlin et al., 2018) and XLNet (Yang Z. and Le, 2019). XLNet is one of most advanced development that reaches state-of-art performance in many downstream tasks such as question answering and sentence classifcation.We also augment finetuning with knowledge(3.4) and test the result of our model(3.5). As shown in Table 2, XLNet-large outperforms BERT-large in devset and testset with only finetuning. However, after external knowledge is added to original datset, the finetuning results of the two pre-trained models do not improve significantly. (e.g. BERT-large 91.14% v.s. BERT-large+knowledge 91.26% on devset). Our assumption about this outcome is that the new knowledge we extract is still with noise, which would also be learnt as features of sentences by pre-trained models, therefore the effects of agumentation are not significant.

Our model(fintuning+GCN+LSTM) perform better than only doing fintuning. It improves 0.4% for BERT and 0.1% for XLNet on testset.

Model	dev-Acc.(%)	test-Acc.(%)
FT-BERT-base	89.04	82.20
FT-BERT-large	91.14	85.17
FT-BERT-large+knowledge	91.26	85.17
FT-BERT-large+GCN+LSTM	91.37	85.57
FT-XLNet-base	90.34	84.73
FT-XLNet-large	91.65	87.70
FT-XLNet-large+knowledge	91.09	
FT-XLNet-large+GCN+LSTM	91.71	**87.80**

Table 2: Comparisons between finetuning on pre-trained language models with graph networks

4.2 Comparison among Different Max Path Length

By changing the max path length to constrain the number of relational triples a sentence can obtain, we observe the changes of our model's final result. We fix the pruning threshold as t = 0.2, and set the max path length as 1,2,3 in turn. For the text model, we use the finetuning-BERT-large. The results is shown in Table 3.

As shown in Table 3, our model performs best when the max path length is 2. When max path length is 1 (which means a target entity and a source entity exist in the same relational triple), there are only 2,100 relational triples extracted for testset, compared with 4034 sentences in total. Therefore, many sentences cannot construct schema graph, which seems too sparse for GCN to effectively learn information of structural patterns.

	Num-Relational Triples		
max path length	train	test	test-Acc.(%)
1	7,500	2,100	85.32
2	380,000	112,000	85.57
3	2,180,000	632,000	85.23

Table 3: Comparison of test accuracy under different max path length. Fix pruning threshold = 0.2. Finetuning-BERT-large's accuracy is 85.17%

	Num-Relational Triples		
pruning threshold	train	test	test-Acc.(%)
0.2	2,180,000	632,000	85.23
0.22	1,145,000	338,200	85.37
0.24	616,000	184,000	85.57

Table 4: Comparison of test accuracy under different path pruning threshold. Fix max path length = 3. Finetuning-BERT-large's accuracy is 85.17%

4.3 Compariosn among Different Pruning Thresholds

We fix the max path length as 3 and set the pruning threshold as 0.2, 0.22, 0.24 in turn, to figure out the effect on the model's final output.The original num of relational triples extracted in trainset and testset is 31,156,000 and 9,173,000. With pruning threshold as 0.2, over 90% of relational triples on trainset and testset are filtered.

As shown in Table 4, as the pruning threshold increases, the number of retained relationships decreases, and the final model results can be slightly improved. The reason we think it can get improvement is that as the threshold goes up, more noisy paths are effectively filtered out. In this way, GCN can better capture the fixed patterns in schema graphs and improve its generalization ability (Lin et al., 2019).

5 Discussion

5.1 Comparison with Knowledge-Aware Graph Networks(KagNet)

We also implement the method proposed in KagNet (Lin et al., 2019) on this dataset and get 86.05% accuracy on testset. Our model adopt result fusion-based augmentation, which combines the output of language model with the output of graph model(GCN+LSTM) to achieve better performance, while KagNet combines the relational representations both from language side and graph side in the module of LSTM-based path encoder, which realizes fusion at feature representation level.

5.2 Unsolved Problems

We find that one of the major flaws in our graph network is that it can't handle two sentences that have exactly the same entities. For example, *Sentence 1: pets are animals vs. Sentence 2: Animals are pets.* Both sentences have entities *animal, pet*. No different entities means schema graphs cannot be constructed. We find 480 similar samples in trainset and 120 in testset belong to this type of problem. Although we have made special treatment to this sample, such as setting default value of empty path, the problem has not been solved. Another difficulty of applying graph network to this dataset is that there are sentences like: *A person doesn't want moldy bread* and *A person want moldy bread* The difference between positive and negative sentences with the same sentence pattern seems unapplicable to our method, which still needs more experiments and research.

6 Conclusions

We present traditional methods as well as utilizing state-of-art pre-trained language models with deep learning graph networks for the Sen-Making task. Our major contribution is to use ConceptNet as the external knowledge database and apply the relation extraction to augment the pre-trained language models. Results show that both BERT and XLNet have satisfactory performance in this task. Still, we can make further improvement by incorporating external knowledge to our model. We also verify the negative influence of the extracted relation on the graph network when it is too sparse or noisy. In the future, we plan to look for other suggestions on similar tasks, from which we can learn more elegant and meticulous methods to accomplish this task. We also plan to apply this approach to other challenging datasets that require commonsense validation and reasoning.

Acknowledgments

We would like to thank Han, Xu and Cao, Shulin and Lv Xin and Lin, Yankai and Liu, Zhiyuan and Sun, Maosong and Li, Juanzi, whose open source code allow us to utilize the KGE techniques without a problem.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. Openke: An open toolkit for knowledge embedding.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks.

Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. *arXiv preprint arXiv:1909.02151*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*.

Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi. 2019. Atomic: An atlas of machine commonsense for if-then reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3027–3035.

Michael Sejr Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks.

Robert Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Shane Stocks, Qiaozi Gao, and Joyce.Y CHai. 2019. Recent advances in natural language inference: A survey of benchmarks, resources, and approaches.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Cunxiang Wang, Shuailong Liang, Yue Zhang, Xiaonan Li, and Tian Gao. 2019a. Does it make sense? and why? a pilot study for sense making and explanation. *arXiv preprint arXiv:1906.00363*.

Xiaoyan Wang, Pavan Kapanipathi, Musa Ryan, Yu Mo, Talamadupula Kartik, Abdelaziz Ibrahim, Chang Maria, Fokoue Achille, Makni Bassem, Mattei Nicholas, and Witbrock Michael. 2019b. Improving natural language inference using external knowledge in the science questions domain.

Yang Y. Carbonell J. Salakhutdinov R. Yang Z., Dai Z. and Q. V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*.