Jeff Da at COIN - Shared Task: BIG MOOD: Relating Transformers to Explicit Commonsense Knowledge

Jeff Da
Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA
jzda@cs.washington.edu

Abstract

We introduce a simple yet effective method of integrating contextual embeddings with commonsense graph embeddings, dubbed BERT Infused Graphs: Matching Over Other embeDdings. First, we introduce a preprocessing method to improve the speed of querying knowledge bases. Then, we develop a method of creating knowledge embeddings from each knowledge base. We introduce a method of aligning tokens between two misaligned tokenization methods. Finally, we contribute a method of contextualizing BERT after combining with knowledge base embeddings. We also show BERTs tendency to correct lower accuracy question types. Our model achieves a higher accuracy than BERT, and we score fifth on the official leaderboard of the shared task and score the highest without any additional language model pretraining.

1 Introduction

Recently, wide-scale pre-training and deep contextual representations have taken the world by storm. Peters et al. (2018) underscored the importance of bidirectional contextual representations by using traditional neural networks trained on a large corpus of text. Devlin et al. (2018) used transformers (Vaswani et al., 2017) and word masking to pre-train on another large corpus of data, reporting human-level performance on one commonsense dataset (Zellers et al., 2018). Yang et al. (2019) achieves state-of-the-art on RACE (Lai et al., 2017) with a Transformer-XL based model (Dai et al., 2019).

The key to success in the performance of many of these models is their ability to train on extremely large datasets. BERT (Devlin et al., 2018), for example, trains on the BooksCorpus (Zhu et al., 2015) and English Wikipedia, for a combined 3,200M words. Other iterations increased the amount of knowledge used during pre-training, such as RoBERTa (Liu et al., 2019). Training large-scale models on these massive datasets has drawbacks, such as significantly increased carbon pollution and harm to the environment (Schwartz et al., 2019; Strubell et al., 2019).

We present a methodology of combining queries from commonsense knowledge bases with contextual embeddings, **BIG MOOD** - BERT Infused Graphs: Matching Over Other embeDdings, and abbreviated for its relationship to human knowledge awareness. Our methodology achieves a increase without significant additional fine-tuning or pre-training. Instead, it learns a separate representation from commonsense graphical knowledge bases, and augments the BERT representation with this learned explicit representation. We introduce several methods of combining and querying knowledge base embeddings to introduce them to the BERT embedding layers.

2 Related Work

2.1 Knowledge Graphs

Significant research has been put into representing human knowledge in various ways (Lenat and Guha, 1989; Auer et al., 2007; Chambers and Jurafsky, 2008). ConceptNet (Speer and Havasi, 2013) contains various aspects of commonsense knowledge through a knowledge graph.The knowledge is collected from crowedsourced resources (Meyer and Gurevych, 2012; Havasi et al., 2010; von Ahn et al., 2006) and expert-created resources (Miller, 1992; Breen, 2004). WebChild (Tandon et al., 2017) is a collection of commonsense knowledge automatically extracted from web contents. The database is constructed similarly to ConceptNet, and intended to cover concepts that ConceptNet does not cover. ATOMIC (Sap et al., 2018) focuses on inferential

Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing, pages 85–92
Hongkong, China, November 3, 2019.

Passage: I had decided that I wanted to visit my friend Paul whom lives quite a distance away. With this and my fear of air travel in mind I decided to take a train. After researching and finding one online I was well on my way to going to see my friend Paul. I drive to the station and decide that I am going to purchase a round trip ticket as this would be cheaper than just buying both tickets separately. Whenever my train arrives I have to get in line as they process our tickets. After all this is done I decide to take a seat by the window. I sit and fall asleep a bit as I ride on the train for hours. After a couple hours we finally reach the destination and I get off the train, excited to see my friend.
When did they wait for their train?
a) before buying the ticket
b) after buying a ticket

Table 1: Example of a prompt from the shared task dataset, an everyday commonsense reasoning dataset. Questions often require script knowledge that extends beyond referencing the text.

$If - Then$ relations, built for everyday commonsense reasoning.

2.2 Knowledge Integration

Knowledge graphs have been applied in various natural language processing applications, such as reading comprehension (Lin et al., 2017; Yang and Mitchell, 2017) and machine translation (Zhang et al., 2017). ERNIE: Enhanced Representation through Knowledge Integration (Sun et al., 2019) appends knowledge to the input of the model and learns via knowledge masking, as well as entity-level masking and phrase-level masking. TriAN (Wang, 2018), the top public model on the MCScript (Ostermann et al., 2018) shared task, uses ConceptNet embeddings to highlight relationships between the question, text, and answer.

3 Model

We present our model for this shared task. Our model has three major components: language model adaptation, knowledge graph embeddings, and attention for classification.

3.1 Data Preprocessing

Before model usage, we preprocess the data in two ways to make it easier for the model to understand. For language modeling, we create training data similar to those in BERT (Devlin et al., 2018). For knowledge graph use, we preprocess language to create commonsense object and relationship vocabulary and to match as many related commonsense objects as possible.

3.1.1 Language Model Preprocessing

We prepossess each passage for training. We use this process for each training epoch, since it allows for the most dense pretraining framework.

Commonly known as a *cloze task*, Devlin et al. (2018) introduced a framework that pretrained transformers (Vaswani et al., 2017) based on masked token prediction. First, we preprocess the tokens with WordPiece embeddings (Wu et al., 2016). Then, we append special $[CLS]$ and $[SEP]$ to each datum. We append $[CLS]$ to the beginning of each datum, and $[SEP]$ to separate the question with the answer, as such:

$$[CLS]\ passage\ question\ [SEP]\ ans.\ [SEP]$$

Then, we randomly mask 15% of all WordPiece embeddings. Unlike Devlin et al. (2018), we run the randomization script once per each training epoch. Otherwise, we follow the procedure in Devlin et al. (2018). 80% of the time, we replace the word with the $[MASK]$ prediction, to be replaced through cloze task prediction. 10% of the time, we replace the word with a random word. 10% of the time, we keep the word unchanged.

Combined with the above cloze task, we process the data for next sentence prediction. We do this process after the cloze task masking, similar to Devlin et al. (2018). For each datum, we randomly pick either a sentence labeled correctly as the next sentence 50% of the time, or a random sentence 50% of the time. We ensure that the random sentence is not the next sentence.

3.1.2 Knowledge Graph Processing

We preprocess the data in the shared task along with knowledge graph preprocessing. The purpose of this procedure is to reduce the number of items in the knowledge graph, to speed up fine-tuning since the knowledge graphs are extremely large, and also to ensure matching between as many different types of knowledge graph edges that are relevant as possible.

First, we create an index of *(start, end, edge)* relationships that match vocabulary within the shared task prompt. For each *(start, end, edge)*, we

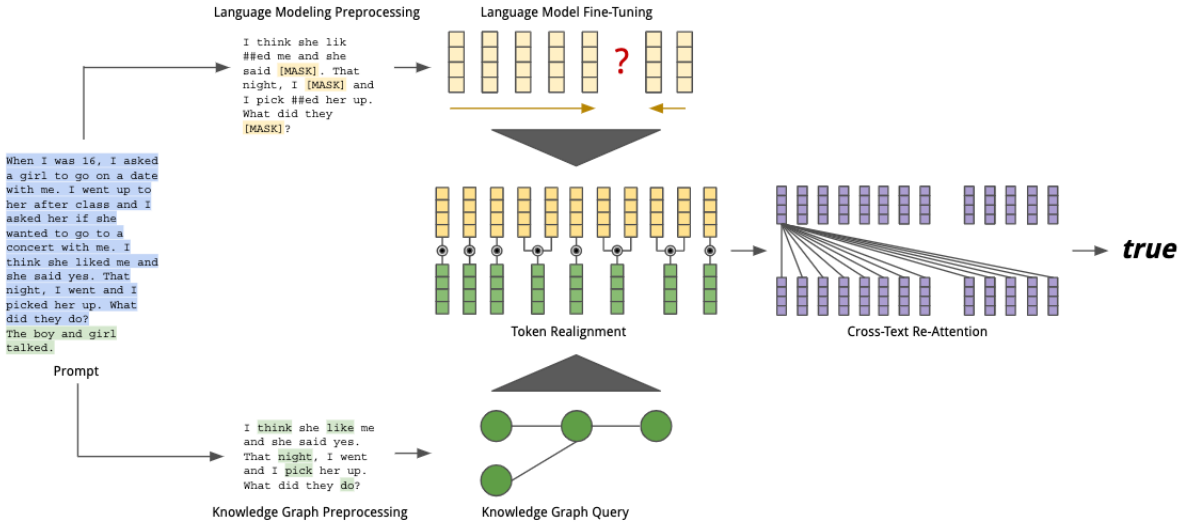

Figure 1: Our model architecture. Our design mimics (Vaswani et al., 2017). Since the queries work on whole words only, one knowledge base embeddings may be integrated with one or more language embedding. Several self-attention encoding layers are used.

Algorithm 1: Knowledge Graph Vocab Creation

```
for prompt in dataset do
    for KG in knowledge_graphs do
        for (start, end, edge) in KG do
            if start in prompt & end in prompt
              then
                add((start, end, edge))
                index_as_relationship(edge)
            end if
        end for
    end for
end for
```

check to see if there are any matching prompts in which *start* is present in the text and *end* is present in the text. If so, we store the *(start, end, edge)*, and note the edge as a relationship. We also index the relationship (edge), giving an index for each unique relationship.

For longer sequences, we allow matches between any trigram, and store an index for each trigram matched. In addition, we stem words beforehand, to ensure that the different word endings do not effect the result of the matches. We use the Porter Stemmer (Porter, 1980) to stem each word in both the text and the knowledge graph. Note that we only use the stemming to match different words, and do not keep the stemmed words for later use in the process, as to keep comparability between embedding types. We also stem words in knowledge bases, to allow for comprasion. Algorithm 1 shows our process for matching sequences.

3.2 Knowledge Graph Usage

We query each of three knowledge bases to create an embedding layer, for each word, for each knowledge graph. Here, we describe our procedure for querying each knowledge graph. We stem words beforehand, to allow for matches agnostic of linguistic postfixes (Merkhofer et al., 2018).

3.2.1 ConceptNet

ConceptNet (Speer and Havasi, 2013) represents everyday words and phrases, with edges between the commonsense relationships between them. We first preprocess ConceptNet, keeping only the vocabulary present in the shared task. Then, for each edge, we store a tuple $(agent, dependent, relationship)$ that describes the commonsense relationship mentioned in the knowledge graph.

During fine-tuning, we check the text for any present $agent$, $dependent$ pairs. If any word in the text is an $agent$, and the $dependent$ is present in the text, we add that $relationship$ index as input into the embedding layer. (For $agent$s that span more than one word, such as the phrase "apple pie", we apply the index to the first word, as long as the entire phrase is found in the text). We randomly generate a length 10 embedding for each relationship, and if more than one relationship is

matched, we randomly pick one.

3.2.2 WebChild

WebChild (Tandon et al., 2017) is a large collection of commonsense knowledge collected from various sources on the web. The format is similar to ConceptNet, which allows us to follow a similar process. WordNet instances are split into categories $part - whole$, $comparative$, $property$, $activity$, and $spatial$. For each category, we capture the $(agent,\ dependent,\ relationship)$ tuple, which is usually defined as properties such as $x_{disambi}$, $y_{disambi}$, and $sub - relation$, but is slightly different for each category. We ignore the WordNet (Miller, 1992) relation (some categories will contain subjects such as $bike\#n\#1$, and take only the stemmed word. For fine-tuning, we follow the same procedure as ConceptNet, creating an additional 10-length embedding for each word.

3.2.3 ATOMIC

ATOMIC (Sap et al., 2018) is a resource that focuses on inferential knowledge via $If - Then$ relations. ATOMIC separates its relationships into nine different types ($xNeed$, $xIntent$, $xAttr$, $xEffect$, $xReact$, $xWant$, $oEffect$, $oWant$). For each of the nine categories, for each datum in the given category, we search our text for relationships that match the defined $If - Then$ relationship. Since each relationship is nearly a full sentence, we allow a match to be any trigram matched between the given datum and the text. Then, we append an index $[0, 8]$ to the embedding layer of the first word in the selected trigram based on the type of relationship matched. For fine-tuning, we follow the same procedure as ConceptNet and WebChild, creating an additional 10-length embedding for each word.

3.3 Architecture

Out modeling procedure consists of three parts. First, we query each knowledge graph, allowing us to create embeddings for each specific graph. Then, we describe our word-level knowledge fusion procedure, creating augmented embeddings for each word. Finally, we describe our fine-tuning procedure for the shared task dataset. We modify pytorch-transformers[1].

3.3.1 Language Model Fine-Tuning

Contrary to Devlin et al. (2018), we do language model fine-tuning in addition to classification fine-tuning. We find that this generally provides better results, and allows for more stable accuracy since the shared task involves a small dataset. For each prompt, we use the previous preprocessed data to create tasks for our model to predict. We do this before token realignment, so this happens before any extra knowledge graph embeddings are added to the model architecture. For masked tokens, we predict that token through bidirectional context, the same as Devlin et al. (2018). For next sentence prediction, we use the unbiased method previously introduced as well as in Devlin et al. (2018).

3.3.2 Token Realignment

We do a word-level fusion to incorporate knowledge embeddings into the BERT model. First, we collect word embeddings from BERT. We sum the last four layer of BERT together, as suggested by "The Illustrated BERT, ELMo, and co." [2]. We fuse these embeddings with the embeddings gathered from querying each of the three databases. For each word, we take the dyadic product, or linear fusion, of the contextual BERT embeddings with the concatenation of the three graph embeddings. When there is no related embedding (if the word did not match any edges during querying, or if the word is a BERT-specific token such as $[CLS]$, we do not do any dyadic fusion. Finally, to get a single linear layer, we concatenate each dimension of the result of the dyadic fusion with the original BERT embedding. Algorithm 2 shows a detailed explanation of our token realignment process.

3.3.3 Re-Attention

To get a final result, we do a few more necessary steps. First, we do a single layer of self-attention over the text, allowing each of the word-level embeddings to interact with one another. For this self-attention, we follow the same process as in (Vaswani et al., 2017). We compare each token with each other and do token-level fusion with each other to learn an attention embedding layer. Then, we use the sequence embedding for classification. We add a simple linear layer over the sequence embedding for classification, and softmax over the given choices. Note that we do not freeze any weights along the process, allowing the transformer and perceptron to

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

[2] http://jalammar.github.io/illustrated-bert/

Algorithm 2: Psuedocode for the token realignment algorithm, a method of finding token alignments between two different sequences.

```
token_realignment(seq_1, seq_2):
alignment_dict = dict
seq_1_i = 0
seq_2_i = 0
while seq_1_i <len(seq_1) & seq_2_i <len(seq_2) do
    if seq_1[seq_1_i] is seq_2[seq_2_i] then
        alignment_dict[seq_1_i].append(seq_2_i)
        seq_1_i++
        seq_2_i++
    end if
    if seq_1[seq_1_i] in seq_2[seq_2_i] then
        alignment_dict[seq_1_i].append(seq_2_i)
        seq_1_i++
    end if
    if seq_2[seq_2_i] in seq_1[seq_1_i] then
        alignment_dict[seq_1_i].append(seq_2_i)
        seq_2_i++
    end if
end while
return alignment_dict
```

be fine-tuned during this process. We also allow the knowledge embeddings to be modified through this back-propagation. Hyperparameters are noted in Section 4.1. We also ablate our use of this extra attention layer, showing that it is important to learn comparisons between knowledge embeddings. For BERT baselines, we use the process in Devlin et al. (2018), and use the $[CLS]$ token, without attention, for classification.

4 Analysis

4.1 Hyperparameter Tuning

For hyperparameter tuning with BERT, we find that grid search is the best method. We tune various hyperparameters, including batch size, learning rate, warmup, and epoch count (for hyperparameter details, see appendix). Graph 2 shows the results of several hyperparameters on BERT with our additional knowledge bases. We find that B. MOOD seems to correct its deficiencies as it gets closer to the maxima. Interestingly, B. MOOD seems to be naturally good "What" questions, which commonly require commonsense inference. This could be explained by the effect of the commonsense knowledge graphs, showing that is picking up on commonsense attributes. However, for "Where" questions, which it requires more information from the text, B. MOOD needs to learn and thus experiences a greater gain as the accuracy gets closer to its maxima.

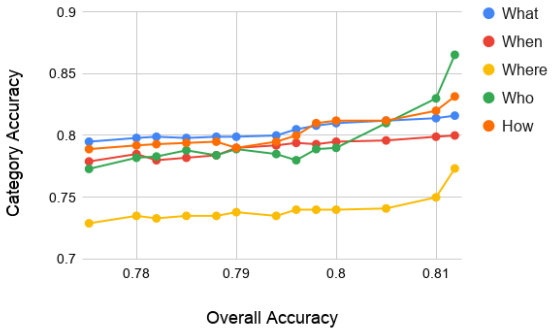

Figure 2: Example of B. MOOD accuracy across categories during hyperparameter turning. Values to the right are closer to the maxima.

We also compare to TriAN (Wang, 2018), the previous state-of-the-art. Table shows our results. For the majority of categories, it seems to begin to be 50/50 between TriAN and MOOD, with TriAN showing more strength in commonsense categories. However, B. MOOD begins to get large jumps in accuracy in categories that it is beat in (such as "Who" and "Where"). For knowledge

System	Accuracy	
	Dev	Test
Human	97.4	98.0
Logistic Baseline	-	60.8
TriAN (Wang, 2018)	76.1	-
BERT$_{LARGE}$	82.3	-
B. MOOD (with ConceptNet)	83.1	-
B. MOOD (with WebChild)	82.7	-
B. MOOD (with ATOMIC)	82.5	-
B. MOOD (w/o final attention)	82.4	-
B. MOOD (with all KB)	**83.3**	**80.7**

Table 2: Results with B. MOOD on task dev and test set. "with all KB" describes results using all ConceptNet, WebChild, and ATOMIC embeddings. "Human" and "Regression Baseline" accuracy is from the shared task paper (Ostermann et al., 2018). TriAN (Wang, 2018) uses ConceptNet as features.

Category	System Accuracy		
	TriAN	BERT	B. MOOD
What	79.3	81.6	84.5
When	69.4	80.0	81.3
Where	75.1	77.3	78.3
Who	79.4	86.5	86.6
How	76.8	83.2	83.4
Overall	76.1	82.3	**83.3**

Table 3: Question type comparison between different models on the shared task: previous state-of-the-art TriAN (Wang, 2018), BERT$_{LARGE}$, and B. MOOD (with all 3 knowledge bases).

embeddings, we use a size of 10 for each knowledge graph, combining for a size 30 knowledge graph embedding. We randomly init each embedding, and if there is more than one embedding for token, we pick one at random (Wang, 2018). For BERT fine-tuning, we use a maximum sequence length of 450, a train batch size of 32, four epochs, $1e-5$ learning rate, and a 20% warmup.

4.2 Results

We show our results and give analysis for MOOD. We show that each of the knowledge bases help the accuracy of our model, and our strongest model involves the union of all three knowledge bases. ConceptNet gives the largest increase, likely because there are the most matches between the prompts and ConceptNet, since ConceptNet covers everyday concepts that are relatively more common. WebChild gives a boost also, but not as large as ConceptNet. ATOMIC gives the smallest boost, likely because 1) ATOMIC queries are the longest, and thus, least likely to match, and 2) there is not as much inferential commonsense present.

We also note that the base B. MOOD accuracy is higher than the base TriAN (Wang, 2018) accuracy, the previous state of the art. By appending similar knowledge embeddings, we find that we can bring the TriAN accuracy up to 77.8%, which is more comparable with MOOD. This shows that the additional knowledge bases (ATOMIC, WebChild) contribute to the overall accuracy even without the contextual embeddings. However, we find that the knowledge bases combined with TriAN still do not provide an improvement above that of MOOD, and thus, the knowledge bases alone are not enough to capture the necessary information. Instead, the knowledge graphs must be used through combination with contextual embeddings for the most effective model. This shows that BERT may lack the complete amount of information needed to understand this dataset. We also show that the attention is needed to understand the knowledge graphs alongside BERT, showing the importance of learning the different knowledge base embeddings within the text. This highlights the fact that using the knowledge base embeddings is helpful, and also comparisons between different sections of text is helpful for reading comprehension tasks.

5 Conclusion

We introduce a method of fine-tuning with graphical embeddings alongside contextual embeddings, MOOD. Our method uses three different knowledge bases, and introduces ways of improving both learning speed and knowledge embedding effectiveness. First, we preprocess the dataset, showing that both language model preprocessing and knowledge graph preprocessing is important to the final result. Then, we tune our language model on the shared task, stabilizing the hyperparameter search. We create knowledge graph embeddings and concatenate the embeddings via token realignment. Then, we introduce a final layer of attention that learns both contextual and explicit graph embeddings through contextualization. We show the effect of various knowledge bases, and show our accuracy across various question types. Our model gets fifth on the task leaderboard and outperforms BERT across all question types. We

hope that this investigation motivates and furthers additional research in combining commonsense knowledge awareness with transformers.

Acknowledgments

The author thanks Maxwell Forbes, as well as the anonymous reviewers, for their helpful and insightful feedback. The author also thanks the shared task organizers for evaluation on the test set. JD is supported by NSF Multimodal.

References

Luis von Ahn, Mihir Kedia, and Manuel Blum. 2006. Verbosity: a game for collecting common-sense facts. In *CHI*.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*.

Jim Breen. 2004. Jmdict: A japanese-multilingual dictionary.

Nathanael Chambers and Daniel Jurafsky. 2008. Unsupervised learning of narrative event chains. In *ACL*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *ArXiv*, abs/1901.02860.

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

Catherine Havasi, R. Speer, Kenneth C. Arnold, Henry Lieberman, Jason B. Alonso, and Jesse Moeller. 2010. Open mind common sense: Crowd-sourcing for common sense. In *Collaboratively-Built Knowledge Sources and AI*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *EMNLP*.

Douglas B. Lenat and Ramanathan V. Guha. 1989. Building large knowledge-based systems; representation and inference in the cyc project.

Hongyu Lin, Le Sun, and Xianpei Han. 2017. Reasoning with heterogeneous knowledge for commonsense machine comprehension. In *EMNLP*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar S. Joshi, Danqi Chen, Omer Levy, Miranda Paige Linscott Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Elizabeth M. Merkhofer, John C. Henderson, David Bloom, Laura Strickhart, and Guido Zarrella. 2018. Mitre at semeval-2018 task 11: Commonsense reasoning without commonsense knowledge. In *SemEval@NAACL-HLT*.

Christian M. Meyer and Iryna Gurevych. 2012. Wiktionary: A new rival for expert-built lexicons? exploring the possibilities of collaborative lexicography.

George A. Miller. 1992. Wordnet: A lexical database for english. *Commun. ACM*, 38:39–41.

Simon Ostermann, Ashutosh Modi, Michael Roth, Stefan Thater, and Manfred Pinkal. 2018. Mcscript: A novel dataset for assessing machine comprehension using script knowledge. *ArXiv*, abs/1803.05223.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. 2018. Deep contextualized word representations. *ArXiv*, abs/1802.05365.

Martin F. Porter. 1980. An algorithm for suffix stripping. *Program*, 40:211–218.

Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2018. Atomic: An atlas of machine commonsense for if-then reasoning. *ArXiv*, abs/1811.00146.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green ai.

R. Speer and Catherine Havasi. 2013. Conceptnet 5: A large semantic network for relational knowledge. In *The People's Web Meets NLP*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. *ArXiv*, abs/1906.02243.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019. Ernie: Enhanced representation through knowledge integration. *ArXiv*, abs/1904.09223.

Niket Tandon, Gerard de Melo, and Gerhard Weikum. 2017. WebChild 2.0 : Fine-grained commonsense knowledge distillation. In *Proceedings of ACL 2017, System Demonstrations*, pages 115–120, Vancouver, Canada. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Lawrence Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Liang Wang. 2018. Yuanfudao at semeval-2018 task 11: Three-way attention and relational knowledge for commonsense machine comprehension. In *SemEval@NAACL-HLT*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

Bishan Yang and Tom M. Mitchell. 2017. Leveraging knowledge bases in lstms for improving machine reading. In *ACL*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *ArXiv*, abs/1906.08237.

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

Jiacheng Zhang, Yang Liu, Huanbo Luan, Jingfang Xu, and Maosong Sun. 2017. Prior knowledge integration for neural machine translation using posterior regularization. *ArXiv*, abs/1811.01100.

Yukun Zhu, Jamie Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

A Appendices

A.1 Hyperparameters

Seen in Table 4 is a list of hyperparameters for our experiments. We use the same parameters for both uses of explicit knowledge embeddings.

Explicit Knowledge Embeddings	
Embedding size	10
Knowledge bases used	3
BERT Fine-Tuning	
Maximum sequence length	450
Train batch size	32
Learning rate	1e-5
Epochs	4
Warmup	20%
TriAN Parameters	
Optimizer	adamax
Learning rate	2e-3
Batch size	32
Hidden size	96
RNN type	lstm
Embedding dropout	0.4

Table 4: Hyperparameters used throughout experiments. TriAN parameters are used for TriAN comparison only.