Generating Negative Commonsense Knowledge

Tara Safavi
Computer Science & Engineering
University of Michigan, Ann Arbor
tsafavi@umich.edu

Danai Koutra
Computer Science & Engineering
University of Michigan, Ann Arbor
dkoutra@umich.edu

Abstract

The acquisition of commonsense knowledge is an important open challenge in artificial intelligence. In this work-in-progress paper, we study the task of automatically augmenting commonsense knowledge bases (KBs) with novel statements. We show empirically that obtaining meaningful *negative samples* for the completion task is nontrivial, and propose NEGATER, a framework for generating negative commonsense knowledge, to address this challenge. In our evaluation we demonstrate the intrinsic value and extrinsic utility of the knowledge generated by NEGATER, opening up new avenues for future research in this direction.

1 Introduction

Endowing machines with human-like commonsense knowledge is a major but elusive goal of artificial intelligence [3]. One way to capture such knowledge in machine-readable form is with commonsense knowledge bases (KBs), which contain semi-structured statements of implicit human knowledge like pre-conditions of events, properties of objects, and outcomes of actions (Figure 1).

As knowledge bases continue to grow in utility in AI, automating their completion has become crucial [11, 8, 2, 13]. Toward this direction, we contribute an analysis of the automatic commonsense KB completion task (§ 2), showing in particular that obtaining meaningful *negatives*—i.e., knowledge of what is explicitly false or non-viable with respect to a specific goal [10, 6]—is nontrivial. To address this challenge, we propose **NEGATER**, a framework for generating **Negat**ive Commonsense Knowledge **E**fficiently and **R**ealistically (§ 3). We demonstrate the intrinsic value and extrinsic utility of the knowledge generated by NEGATER in a preliminary evaluation (§ 4), opening up several avenues for future research in automatically generating and evaluating negative knowledge.

2 Preliminary analysis

Terminology A commonsense knowledge base (KB) comprises a set of (*head phrase*, *relation*, *tail phrase*) triples $x^+ = (X_h, r, X_t)$, where the superscript denotes that in-KB triples are assumed positive or true, e.g., (*sky*, *HasProperty*, *blue*). We define a negative x^- as any triple that expresses a **false or non-viable declarative statement**. Note that we distinguish negatives from linguistic *negation*, as negation may be present in positive triples, e.g., (*sky*, *HasProperty*, *not pink*). The head and tail phrases $X_h = [u_1, \ldots, u_h]$ and $X_t = [v_1, \ldots, v_t]$ are sequences of tokens drawn from a potentially infinite vocabulary, whereas relations r are symbolic and drawn from a finite dictionary R.

Experimental setup We motivate the need for negative knowledge with an instructive preliminary experiment. Following Li et al. [8], we define commonsense KB completion as the task of classifying novel commonsense triples with labels $y \in \{+1, -1\}$, and use their evaluation dataset drawn from the ConceptNet KB [16]. The dataset, which has been established as the primary benchmark for the task [14, 7, 4], is split into 100K/2,400/2,400 train/validation/test triples. In the evaluation splits, half

4th Knowledge Representation and Reasoning Meets Machine Learning Workshop (KR2ML 2020), at NeurIPS.

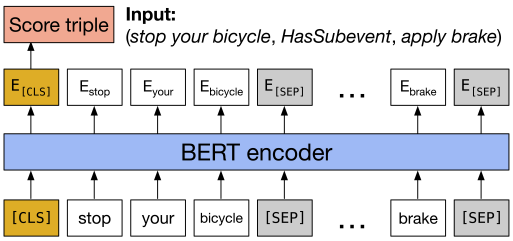


Figure 1: Fine-tuning BERT for commonsense classification with a triple scoring layer.

Table 1: Accuracy on the original (§ 2) and new (§ 4) ConceptNet datasets.

	Original	New	Diff.
Bilinear AVG [8]	0.9170	0.6695	-0.2475
DNN AVG [8]	0.9200	0.6410	-0.2790
DNN LSTM [8]	0.8920	0.6305	-0.2615
DNN AVG + CKBG [14]	0.9470	-	-
Factorized [7]	0.7940	0.7068	-0.0872
Prototypical [7]	0.8900	0.5586	-0.3314
Coherency Ranking [4]	0.7880	0.6387	-0.1493
BERT (ours)	**0.9537**	**0.7855**	-0.1682
Human estimate	0.95	0.86	-0.09

of all triples are positives x^+. The other half are negatives x^- constructed by replacing either the head, tail, or relation of positives x^+ with a randomly selected phrase or relation.

As baseline methods, we compare **Bilinear AVG** [8], **DNN AVG** [8], **DNN LSTM** [8], **DNN AVG + CKBG** [14], **Factorized** [7], **Prototypical** [7], **Coherency Ranking** [4], and **estimated human performance** by a native English speaker on a sample of 100 test triples. The first seven baselines are self-supervised, whereas Coherency Ranking is unsupervised. Appendix A.1 provides further methodological and implementation details for each baseline.

We also propose to **fine-tune the BERT language model** [5] for the completion task (Figure 1); we will demonstrate this model's strengths compared to existing baselines and ultimately use it as part of our NEGATER framework (§ 3). Given a triple (X_h, r, X_t), we construct an input sequence composed of BERT's special `[CLS]` token, the head phrase tokens X_h, the relation label r, and the tail phrase tokens X_t, delineating each slot with BERT's special `[SEP]` token. At the output of the BERT encoder, we take each triple's `[CLS]` token embedding as its respective representation, and pass these embeddings through a triple scoring layer $W \in \mathbb{R}^H$, where H is the hidden layer dimension. We fine-tune in a self-supervised manner common to the KB completion literature [1, 8, 12, 13], generating negative samples by randomly corrupting each slot per positive triple in turn and optimizing binary cross-entropy (BCE) loss. We classify triples by setting a decision threshold per relation on the validation set to maximize validation accuracy. Appendix A.2 provides details on model selection.

Results and discussion Table 1 gives classification accuracy in the "Original" column. Performance numbers for baselines are reported directly from their referenced citations. As shown in the table, our fine-tuned BERT outperforms all baselines, **setting a new record on this benchmark** and even marginally exceeding the human performance estimate. However, this does *not* mean that the problem is solved. Rather, **the evaluation task is too easy for BERT** for two reasons.

First, because concepts in commonsense KBs are often not disambiguated [16], many triples in such KBs **paraphrase each other**, expressing the same semantics with minor differences in surface forms [7]: For example, the triples (*plane*, *UsedFor*, *travel*) and (*airplane*, *UsedFor*, *travel*) appear in the train and test sets of the ConceptNet benchmark, respectively. To estimate the degree to which the benchmark tests paraphrasing, for each positive test triple we retrieve its top-5 nearest neighbors in the training set using L2 distance between `[CLS]` embeddings of triples output by our fine-tuned BERT encoder. Among a random sample of 100 test triples annotated by a native English speaker, ~52% have a near-duplicate paraphrase in train. Appendix A.3 provides our annotation guidelines, and Table 3 in Appendix A.5 gives further examples of paraphrases.

Second, because the validation and test negatives x^- are randomly generated, they are usually **ungrammatical or incoherent**, making classification easy based on grammar alone. In a sample of 100 negative test triples, ~40% have an incorrect part of speech in the head or tail slot. An example is (*clarinet*, *HasSubevent*, *orchestra*), which is incorrect because the *HasSubevent* relation requires events in the head and tail slots. Appendix A.4 provides our annotation guidelines, and Table 4 in Appendix A.5 provides more examples of grammatically incorrect negatives with explanations.

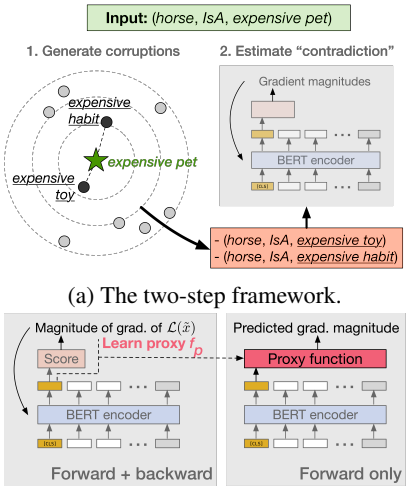
While paraphrases can be identified and removed using nearest-neighbor techniques, constructing high-quality negatives is more challenging because KBs typically do not contain negative statements. For example, the ConceptNet benchmark contains a small proportion (~3%) of explicit negatives, but only for a select few relations, such as *NotCapableOf* and *NotDesires*; other commonsense KBs

such as ATOMIC [15] do not contain negatives at all. Therefore, in the next section we propose an unsupervised framework for generating negatives from arbitrary positive statements.

3 NEGATER: A framework for generating negative knowledge

To introduce our NEGATER framework, we first note that the space of possible negatives in knowledge bases is much larger than the space of positives. We must therefore restrict the output of the negative knowledge generation task to a subset of the negative space. Following Minsky [10], who defined positive knowledge as "islands of consistency of commonsense" and negative knowledge as the "unsafe boundaries of those islands", we aim to generate negatives *on the boundary* of positive and negative knowledge, i.e., knowledge that *looks* plausible and is "almost correct", but would be misleading or harmful if considered true. Note that we exclude trivial one-to-one relations from this definition, e.g., erroneous dates of events or incorrect taxonomic classifications.

Methodologically speaking, one way to generate negative knowledge "on the boundary" is to corrupt positive triples in a way that preserves relational grammar but slightly alters semantics. Consider the positive ConceptNet triple (*horse*, *IsA*, *expensive pet*). We could, for example, replace the head phrase *horse* with another animal that is inexpensive to keep as a pet (e.g., *goldfish*). Likewise, we could replace *expensive pet* with a noun phrase that is similar in surface form but semantically distinct and incorrect in context (e.g., *expensive car*). We propose a two-step framework to achieve this (Figure 2a).



(a) The two-step framework.

(b) Improving the efficiency of Step 2.

Figure 2: Overview of NEGATER.

Step 1: Generate corruptions We first generate high-quality, plausible candidate negatives by corrupting positives. Given a positive $x^+ = (X_h, r, X_t)$, we retrieve the k-nearest neighbors by semantic similarity to head phrase X_h. Here, any top-k retrieval method works; we use L2 distance between phrases' `[CLS]` embeddings as encoded by (pre-trained) BERT. We replace X_h in the original triple by each of its k neighbors $\tilde{X}_h$ in turn, yielding a set of corruptions $\{\tilde{x}\}_{i=1}^{k}$, $\tilde{x} = (\tilde{X}_h, r, X_t)$. To preserve the grammar of the relation r, we discard the corruptions $\tilde{x}$ for which $\tilde{X}_h$ does not appear in the head slot of relation r at least once in the KB. We repeat this process for the tail phrase X_t, yielding $\leq 2k$ corruptions $\tilde{x}$ per positive x^+.

Step 2: Find contradictions Next, we assess the candidates by the level to which they "contradict" the positive beliefs of our fine-tuned BERT model from § 2. Let $\tilde{z}$ be the triple score of corruption $\tilde{x}$ as output by our model, and $\mathcal{L}(\tilde{x}) = -\tilde{y}\log\sigma(\tilde{z}) + (1-\tilde{y})\log(1-\sigma(\tilde{z}))$ be the BCE loss evaluated on corruption $\tilde{x}$ given a corresponding label $\tilde{y} \in \{-1, 1\}$. We feed $\tilde{x}$ to our fine-tuned model and compute the magnitude of the gradient of $\mathcal{L}(\tilde{x})$ with respect to BERT's fine-tuned parameters Θ using a *positive labeling* of $\tilde{x}$: $\tilde{M} = \|(\partial\mathcal{L}(\tilde{x}; \tilde{y} = 1))/\partial\Theta\|$ (note that we do not actually update Θ here). To generate n negatives, we take the corruptions with the top-n values of $\tilde{M}$.

Intuitively, the value of $\tilde{M}$ for a single corruption $\tilde{x}$ signifies the amount to which BERT's fine-tuned beliefs would need to be updated to incorporate this corruption as positive. Therefore, the higher the $\tilde{M}$, the more directly $\tilde{x}$ **contradicts or negates BERT's already fine-tuned beliefs**.[1] However, note that computing $\tilde{M}$ directly is costly because it requires a full forward and backward pass plus a summation over all gradients for *each corruption* $\tilde{x}$, precluding batching. We thus propose to **learn a proxy function** f_p **that predicts** $\tilde{M}$ **given** $\tilde{x}$, so that we can replace the gradient computation with f_p and avoid the backward pass altogether. Assume we have C corruptions $\tilde{x}$. We first evaluate $\tilde{M}$ for an initial sample of $c \ll C$ corruptions. We then use the `[CLS]` embeddings of these corruptions and their corresponding gradient magnitudes $\tilde{M}$ as training features and targets, respectively, to learn

[1] Note that a corruption $\tilde{x}$ with a high $\tilde{M}$ may not necessarily yield a low triple score $\tilde{z}$ or a high value of $\mathcal{L}$, since the model was trained to discriminate between positives and *randomly generated* negatives.

Table 2: Examples of generated negatives. Table 5 in the Appendix gives more examples.

Head phrase	Relation	Tail phrase
heater	*UsedFor*	*produce breeze*
computer program	*MadeOf*	*silicon*
fly kite	*HasPrerequisite*	*get skis*
muffin	*AtLocation*	*hot-dog stand*
butterfly	*HasProperty*	*hunted by humans for food*
theatre ticket	*UsedFor*	*get home from work*

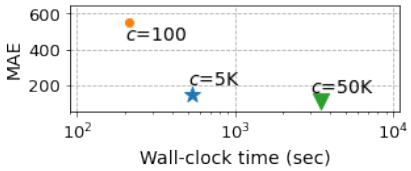


Figure 3: Wall-clock time versus MAE, varying the number of initial corruptions c. Lower left corner is best.

$f_p : \mathbb{R}^H \rightarrow \mathbb{R}$. After training, we simply substitute BERT's triple scoring layer with f_p (Figure 2b), allowing us to skip the backward pass and feed batches of corruptions to the model in forward passes.

4 Evaluation

We conduct three experimental evaluations: **(1)** A small-scale study of the intrinsic quality of the negatives generated by NEGATER; **(2)** A larger-scale comparison of how these negatives affect KB completion performance; and **(3)** An efficiency analysis. For our evaluation we construct a new dataset from the ConceptNet benchmark from § 2. We randomly re-split the positive triples into 95% train and 2.5% validation/test each and remove near-duplicate paraphrases (details in Appendix A.6), then double the size of the validation and test sets by adding negatives generated with NEGATER. We generate up to 10 corruptions $\tilde{x}$ per positive x^+ in the dataset. For simplicity, we implement the proxy f_p as a feedforward network with two 100-dimensional hidden layers, dropout probability 0.1, and a ReLU activation at the output, trained for 100 epochs over $c = 20$K corruptions using L1 loss. Our new ConceptNet dataset has 84,427 training triples and 5,120 validation/test triples each.

Intrinsic evaluation We first have a native English speaker annotate a small number of generated test negatives for quality. Among a randomly selected sample of 200 negatives, ~94.5% are grammatical and ~86% are true negatives (see Appendix A.4 for our annotation guidelines). Comparing these estimates to the original ConceptNet test split in which ~60% were labeled as grammatical and ~90% were labeled as true negatives using the same annotation guidelines, we can conclude that NEGATER yields **more realistic-looking negatives with little increase in the false negative rate**. Table 2 provides examples of generated test negatives. As intended by the design of our framework, these negatives are "close, but not quite" positive. For example, (*heater*, *UsedFor*, *produce breeze*) would be positive if *heater* were replaced by *fan*, a similar object with a different everyday usage.

Extrinsic evaluation We next compare the difficulty of our new dataset to its original counterpart for the KB completion task in Table 1.[2] Model accuracy drops on average ~21.77 percentage points, compared to ~9 points for the human annotator. While our fine-tuned BERT performs best (78.55% accuracy), there is a long way to reaching human performance (~86% accuracy), indicating that humans' commonsense generalization and discrimination abilities **still far exceed that of machines**.

Efficiency evaluation Finally, Figure 3 compares the efficiency (wall-clock time) of NEGATER to the accuracy of the proxy approach as measured by the mean absolute error (MAE) between the proxy function's predictions and the true gradient magnitudes $\tilde{M}$, varying the number of initial samples c for ~50K corruptions. Evidently, there are diminishing returns in error rate as the number of samples increases, suggesting that the proxy approach **learns a good model in a short amount of time**.

5 Conclusion

In this paper we motivated the need for negative knowledge and proposed NEGATER to this end. In the future, we plan to expand our intrinsic evaluation in terms of scope and number of samples considered. We will also explore various external usages of our generated negatives, for example toward improving agent reasoning and robustness in safety-critical AI applications.

[2]We exclude DNN AVG + CKBG as we were not able to obtain an implementation.

References

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, pages 2787–2795, 2013.

[2] Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. Comet: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, 2019.

[3] Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM*, 58(9):92–103, 2015.

[4] Joe Davison, Joshua Feldman, and Alexander M Rush. Commonsense knowledge mining from pretrained models. 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 1173–1178, 2019.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[6] Martin Gartmeier, Johannes Bauer, Hans Gruber, and Helmut Heid. Negative knowledge: Understanding professional learning and expertise. *Vocations and Learning*, 1(2):87–103, 2008.

[7] Stanislaw Jastrzębski, Dzmitry Bahdanau, Seyedarian Hosseini, Michael Noukhovitch, Yoshua Bengio, and Jackie Chi Kit Cheung. Commonsense mining as knowledge base completion? a study on the impact of novelty. In *Proceedings of the Workshop on Generalization in the Age of Deep Learning*, pages 8–16, 2018.

[8] Xiang Li, Aynaz Taheri, Lifu Tu, and Kevin Gimpel. Commonsense knowledge base completion. In *ACL*, pages 1445–1455, 2016.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] Marvin Minsky. Negative expertise. *International Journal of Expert Systems*, 7(1):13–19, 1994.

[11] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[12] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You {can} teach an old dog new tricks! on training knowledge graph embeddings. In *ICLR*, 2020.

[13] Tara Safavi and Danai Koutra. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8328–8350, 2020.

[14] Itsumi Saito, Kyosuke Nishida, Hisako Asano, and Junji Tomita. Commonsense knowledge base completion and generation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 141–150, 2018.

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

[16] Robyn Speer and Catherine Havasi. Representing general relational knowledge in conceptnet 5. In *LREC*, pages 3679–3686, 2012.

A Appendix

A.1 Baselines

We provide more details on each our baselines here. For the new ConceptNet dataset, all baselines except for DNN AVG + CKBG were run using the code provided by the authors.

- **Bilinear AVG** [8]: Triples are scored with a bilinear model $\mathbf{h}^\top \mathbf{R}\mathbf{t}$, where $\mathbf{h}$ and $\mathbf{t}$ are embeddings representing head phrase X_h and tail phrase X_t respectively, and $\mathbf{R}$ is a parameter matrix for relation r. Phrases are initially converted to vectors by averaging word embeddings obtained with word2vec [9]. The model is trained with binary cross-entropy loss, and negatives x^- are sampled by replacing the head, relation, and tail of each positive x^+ in turn (i.e., for each positive, three negatives are constructed). Scores are converted to binary labels by setting a decision threshold on the validation set to maximize validation accuracy. We use the recommended hyperparameters from Li et al. [8].
- **DNN AVG** [8]: Phrases are again initially converted to vectors by averaging word2vec embeddings. Then, the phrase and relation vectors are concatenated to form a single input $\mathbf{v}$, which is passed through a neural network with one hidden layer: $\mathbf{W}_2\text{ReLU}(\mathbf{W}_1\mathbf{v}+\mathbf{b}_1)+\mathbf{b_2}$. The model is trained in the same manner as **Bilinear AVG**.
- **DNN LSTM** [8]: The architecture is the same as **DNN AVG**, except the head and tail phrases are concatenated into a single sequence, then converted to a vector via a bidirectional LSTM.
- **DNN AVG + CKBG** [14]: The architecture contains a completion module similar to the **DNN AVG** baseline, but also contains a generation component that shares the bidirectional LSTM used to encode the head and tail phrases. The generation component uses another bidirectional LSTM as a decoder. The generation component is trained using cross-entropy. Negatives are sampled in the same manner as **Bilinear AVG**.
- **Factorized** [7]: Triples are scored with a two-way factor model $\alpha\langle\mathbf{A}\mathbf{h}+\mathbf{b_1}, \mathbf{B}\mathbf{t}+\mathbf{b_2}\rangle + \beta\langle\mathbf{A}\mathbf{r}+\mathbf{b_1}, \mathbf{B}\mathbf{t}+\mathbf{b_2}\rangle + \gamma\langle\mathbf{A}\mathbf{r}+\mathbf{b_1}, \mathbf{B}\mathbf{h}+\mathbf{b_2}\rangle$, where $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$ are representations for the head phrase, relation, and tail phrase respectively, $\mathbf{A}$ and $\mathbf{B}$ are parameter matrices, $\mathbf{b}_i$ are bias terms, and α, β, and γ are learned scalars. The model is trained in the same manner as **Bilinear AVG**. We use the recommended hyperparameters from Jastrzębski et al. [7].
- **Prototypical** [7]: Triples are scored in the same manner as the **Factorized** approach, but without the first head-tail term. The model is trained in the same manner.
- **Coherency Ranking** [4]: This approach mines commonsense knowledge directly from BERT's pre-trained parameters without fine-tuning. For each triple, candidate sentences are constructed by inserting the head and tail phrase into a set of manually defined relation templates, then the sentence with the highest log-likelihood according to a pre-trained language model is selected. All such sentences are then scored according to plausibility by using the point-wise mutual information of the head phrase X_h and tail phrase X_t conditioned on the relation r.

A.2 BERT implementation

We fine-tune the 12-layer BERT-BASE model. Following the recommendations of Devlin et al. [5], we search among the following hyperparameters (best configuration in **bold**):

- Batch size in $\{\mathbf{16}, 32\}$
- Learning rate in $\{10^{-4}, 10^{-5}, \mathbf{2 \times 10^{-5}}, 3 \times 10^{-5}\}$
- Number of epochs in $\{3, 5, 7, 10, \mathbf{13}\}$
- Number of warmup steps in $\{0, \mathbf{10K}, 50\text{K}\}$
- Maximum sequence length in $\{16, \mathbf{32}, 64\}$

All other hyperparameters are as reported in [5]. We fine-tune the model with the same hyperparameter configuration for the original ConceptNet dataset and the new dataset. Our code was implemented using the PyTorch transformers library and run on a NVIDIA Tesla V100 GPU with 16 GB of RAM.

A.3 Instructions for labeling paraphrases

The goal of this task is to determine whether a given statement has a *paraphrase* among five associated candidate statements. Each row of data you must annotate contains a "commonsense knowledge" statement in the form of a (*head phrase*, *relation*, *tail phrase*) triple along with five candidate statements, also in the form of commonsense triples. Wikipedia defines a paraphrase as a **restatement of the meaning of a text or passage using different words**. That is, statement A paraphrases statement B if statements A and B express the same meaning but with different words or different spelling.

For each statement, you must select `has_paraphrase` if the statement has one or more paraphrases among the given candidates, and `no_paraphrase` otherwise. You may explain your reasoning if need be in the optional explanation column.

Examples of statements that have one or more paraphrases among the candidates include (paraphrases in bold):

- (*pay bill*, *HasPrerequisite*, *have money*)
  - (*pay bill*, *HasPrerequisite*, *use money*)
  - **(*pay bill*, *HasPrerequisite*, *money*)**
  - **(*pay bill*, *HasPrerequisite*, *get money*)**
  - (*spend money*, *HasPrerequisite*, *money*)
  - (*buy*, *HasPrerequisite*, *have money*)
- (*internet*, *UsedFor*, *research*)
  - **(*go on internet*, *UsedFor*, *do research*)**
  - **(*go on internet*, *UsedFor*, *research*)**
  - **(*surf net*, *UsedFor*, *research*)**
  - (*internet*, *UsedFor*, *learning*)
  - **(*surf net*, *UsedFor*, *do research*)**

Examples of statements that do not have a paraphrase among their associated candidates include:

- (*peach*, *IsA*, *fruit*)
  - (*grapefruit*, *IsA*, *fruit*)
  - (*orange*, *IsA*, *fruit*)
  - (*cherry*, *IsA*, *fruit*)
  - (*chocolate*, *IsA*, *fruit*)
  - (*pumpkin*, *IsA*, *fruit*)
- (*play basketball*, *HasSubevent*, *dribble*)
  - (*prove your physical endurance*, *HasSubevent*, *run 10k*)
  - (*go shopping*, *HasSubevent*, *make impulsive purchase*)
  - (*prove your physical endurance*, *HasSubevent*, *lift weight*)
  - (*ride bicycle*, *HasSubevent*, *pedal*)
  - (*exercise*, *HasSubevent*, *increase your stamina*)

A.4 Instructions for labeling negatives

The goal of this task is to accurately label statements as TRUE or FALSE based on "intuitive commonsense reasoning". Each row of data you must annotate contains a "commonsense knowledge" statement in the form of a (*head phrase*, *relation*, *tail phrase*) triple.

Please read through the provided examples first before reading the task instructions.

Examples of true statements Examples of TRUE triples and corresponding explanations include:

- (*pay bill*, *HasPrerequisite*, *have money*): In order to pay for anything, one must have a form of payment, i.e., money.
- (*school bus*, *HasProperty*, *yellow*): While not all school buses are yellow, yellow school buses are very common in the public image, as the yellow school buses used in the USA and Canada are considered iconic (in fact, "school bus yellow" is even a standardized color.)
- (*basketball*, *HasProperty*, *round*): Basketballs are spherical (round) objects.
- (*salt*, *AtLocation*, *ocean*): While not all oceans are uniformly salty, they all contain salt.
- (*lie*, *HasSubevent*, *you feel guilty*): Lying often causes guilt in people, although of course the amount of guilt depends on the person.

Examples of false statements FALSE statements may have problems with grammar, factual content, or both. Some examples of grammatically incorrect statements include:

- (*pilot*, *HasProperty*, *land airplane*): The *HasProperty* relation requires a description (an adjective or adjective phrase) in the tail slot, whereas land airplane is an action.
- (*tie your shoelace*, *UsedFor*, *smart*): The *UsedFor* relation requires an action (verb) or thing (noun) in the tail slot, whereas smart is an adjective.
- (*clothes*, *Desires*, *drawer*) Clothes aren't sentient objects, so they can't desire anything.

Examples of grammatically correct but factually false statements include:

- (*store item*, *AtLocation*, *doorway*): Doorways are not storage locations; items are typically stored in closets or drawers.
- (*swim*, *UsedFor*, *travel on land*): The definition of swimming is to travel in water.
- (*zookeeper*, *AtLocation*, *jungle*): While a zookeeper might go to the jungle occasionally, you would normally expect to find a zoo keeper at the zoo.
- (*sit on vinyl seat*, *HasProperty*, *very painful*): While some people may consider sitting on vinyl seats painful, it's not a common perception of vinyl seats.

Task You must label each triple as either TRUE or FALSE. If you label a triple as FALSE, you must select one of the two dropdown options for false triples (grammatically incorrect or factually incorrect).

Note that in the TRUE examples, not all triples are true 100% of the time. For example, school buses are not always yellow, and not all people feel guilty after lying. However, for the purposes of labeling you should consider a triple TRUE if the statement it expresses is "usually" or "often" true or possible. Likewise, you should consider a triple FALSE if the statement it expresses if not "usually" or "often" true. You should not have to think too long about whether they're true or false unless you need to look up a definition. If you find yourself having to stretch your reasoning or logic in order for a statement to be true, label it FALSE.

If you are unsure of an answer, you may Google it. You may explain your reasoning if need be or provide sources to back up your answer in the optional explanation column.

A.5 Paraphrasing and grammar examples

Table 3 gives examples of paraphrases between train and test in the original ConceptNet dataset, and Table 4 gives examples and explanations of negative triples labeled as ungrammatical in each dataset. Note also that the original ConceptNet benchmark was split by decreasing confidence level in the ConceptNet database, which also contributes to the train/test overlap observed in § 2. Jastrzębski et al. [7] also explored this overlap and proposed to randomly re-split the data to mitigate the overlap, albeit without removing paraphrases or generating harder negatives.

Table 3: Examples of paraphrases in ConceptNet between train and test.

Train	Test
(*key*, *UsedFor*, *open lock*)	(*key*, *UsedFor*, *unlock door*)
(*bar stool*, *UsedFor*, *sit*)	(*bar stool*, *UsedFor*, *sit on*)
(*plane*, *UsedFor*, *travel*)	(*airplane*, *UsedFor*, *travel*)
(*frypan*, *UsedFor*, *cook*)	(*fry pan*, *UsedFor*, *cook*)
(*yo yo*, *IsA*, *toy*)	(*yo-yo*, *IsA*, *toy*)
(*staircase*, *UsedFor*, *climb*)	(*stair*, *UsedFor*, *climb*)

Table 4: Examples of ungrammatical negatives x^- in the original dataset, with explanations.

Negative x^-	Explanation
(*tie your shoelace*, *UsedFor*, *smart*)	The *UsedFor* relation requires a verb phrase in the tail slot.
(*have food*, *CapableOf*, *eat*)	The *CapableOf* relation requires a noun in the head slot.
(*frisbee*, *HasProperty*, *turn out light*)	The *HasProperty* relation requires a description in the tail slot.

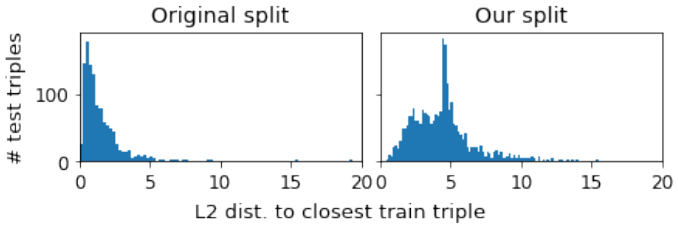


Figure 4: Quantifying paraphrasing by L2 distance from each test triple's to its closest training triple `[CLS]` embedding.

A.6 Removing paraphrases

To remove paraphrases, we represent each triple in the dataset by its `[CLS]` embedding at the output of our fine-tuned BERT model. Then, for each validation and test triple, we retrieve its top-5 nearest neighbors in the training set. We compute the mean L2 distance μ among all such validation/train and test/train pairs, and remove from the training set all triples with distance $\leq \mu$ from a validation or test triple.

To compare the amount of paraphrasing between train and test in the original dataset versus our new dataset, Figure 4 plots a histogram of the L2 embedding distance of each test triple to its nearest training triple. For the original dataset, distribution is highly right-skewed, as most test triples have a very similar training counterpart. By contrast, the distribution is approximately normal in our new dataset, indicating that it better tests generalization.

Table 5: Examples of ConceptNet test negatives generated by NEGATER.

Head phrase	Relation	Tail
internet	*IsA*	*secretary with many files*
gross domestic product	*HasProperty*	*abbreviated to ctbt*
communism	*AtLocation*	*paris*
u2	*DefinedAs*	*name of basketball team*
eat in fast food restaurant	*HasSubevent*	*read tabloid headline*
telegraph	*IsA*	*audible media*
rocket	*AtLocation*	*junkyard*
swallow	*IsA*	*insect*
go public	*HasSubevent*	*pound gavel*
potato	*AtLocation*	*bunch with other grapes*
go into trance	*HasPrerequisite*	*have unhealthy heart*
chemist	*AtLocation*	*orbit*
postmaster	*AtLocation*	*town hall*
nature	*IsA*	*small place*
hot air balloon	*IsA*	*rotary wing aircraft*
in winter gloves	*CapableOf*	*shade person from sun*