Adversarial Transformer Language Models for Contextual Commonsense Inference

Pedro Colon-Hernandez [a], Henry Lieberman [b], Yida Xin [c], Claire Yin [b], Cynthia Breazeal [a] and Peter Chin [c]

[a] *Media Lab, Massachusetts Institute of Technology*
[b] *Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*
[c] *Department of Computer Science Boston University*

Abstract. Contextualized commonsense inference is the task of generating commonsense assertions from a given story, and a sentence from that story. (Here, we think of a story as a sequence of causally-related events and descriptions of situations.) This task is hard, even for modern contextual language models. Some of the problems with the task are: lack of controllability for topics of the inferred assertions; lack of commonsense knowledge during pre-training; and possibly, hallucinated or false assertions. We utilize a transformer model as a base inference engine to infer commonsense assertions from a sentence within the context of a story. The task's goals are to make sure that (1) the generated assertions are plausible as commonsense; and (2) to assure that they are appropriate to the particular context of the story.

With our inference engine, we control the inference by introducing a new technique we call "hinting". Hinting is a kind of language model prompting [1], that utilizes both hard prompts (specific words) and soft prompts (virtual learnable templates). This serves as a control signal to advise the language model "what to talk about". Next, we establish a methodology for performing joint inference with multiple commonsense knowledge bases. While in logic, joint inference is just a matter of a conjunction of assertions, joint inference of commonsense requires more care, because it is imprecise and the level of generality is more flexible. You want to be sure that the results "still make sense" for the context. We show experimental results for joint inference, including three knowledge graphs (ConceptNet[2], Atomic2020[3], and GLUCOSE[4]). We align the assertions in the knowledge graphs with a story and a target sentence, and replace their symbolic assertions with textual versions of them. This combination allows us to train a single model to perform joint inference with multiple knowledge graphs.

Our final contribution is a GAN architecture that uses the contextualized commonsense inference from stories as a generator; and that discriminates by scoring the generated assertions as to their plausibility. The result is an integrated system for contextualized commonsense inference in stories, that can controllably generate plausible commonsense assertions, and takes advantage of joint inference between multiple commonsense knowledge bases.

Keywords: Language Models, Adversarial, Commonsense, Joint Inference, Controllable Generation

1. Introduction

Contextualized or discourse aware commonsense inference is the task of inferring coherent commonsense assertions or facts from a story context and a selected or target sentence. This framing (i.e., story and target sentence) is important because a story can be anything: an actual story, a procedure, etc. In the case of a story, such contextual commonsense inference can help with story understanding (e.g., it could give commonsense assertions that indicate a revenge plot[5]), and in the case of a procedure, it could help with explanations and step rephrasing. Such framing allows us to go sentence by sentence, inferring assertions, which requires the model to utilize and possibly maintain prior information for coherence. We give an example of this task in Figure 1.

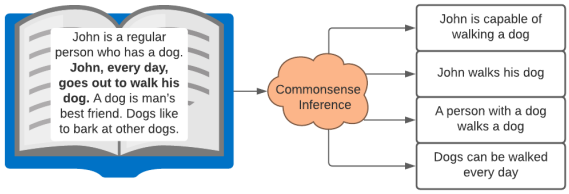

Fig. 1. Overview of the task of contextual commonsense inference. From the story on the left, and the **bolded** sentence, a system has to infer assertions such as the ones on the right.

This task is hard for modern contextual language models. It may rely on information that a model may not have seen during pre-training, or the model has to figure out what topic to infer information about. In the case of commonsense inference, this problem is exarcerbated as commonsense knowledge, present in everyone, tends to not be written explicitly in text [6–9]. In addition to this, the factuality or the correctness of the information that is generated by the models is hard to evaluate and usually involves a costly human-in-the-loop setup.

Prior work, such as COMET[10], has tried to do sentence-level commonsense inference. COMET was improved to work on a paragraph-level, namely ParaCOMET[11], by extending it with a recurrent memory and training it on an corpus of aligned stories and assertions. ParaCOMET builds a dataset to address the contextual commonsense inference task by aligning facts from a commonsense knowledge graph (i.e., ATOMIC[12]) with a story (i.e., sampled from ROCStories[13]) through a heuristic based on the ROUGE [14] metric. It goes a step further by utilizing the cross entropy of story tokens of a language model, conditioned on one of the matched facts as a measure of coherence to keep only matches that are coherent to the narrative. They additionally address the need for memory (i.e., for the model to remember prior events) by using and saving prior aligned assertions in a memory system. In the work that we present, it is possible to utilize this same memory system, however we leave testing of it for future work. In figure 2[11], we can see an overview of how ParaCOMET functions.

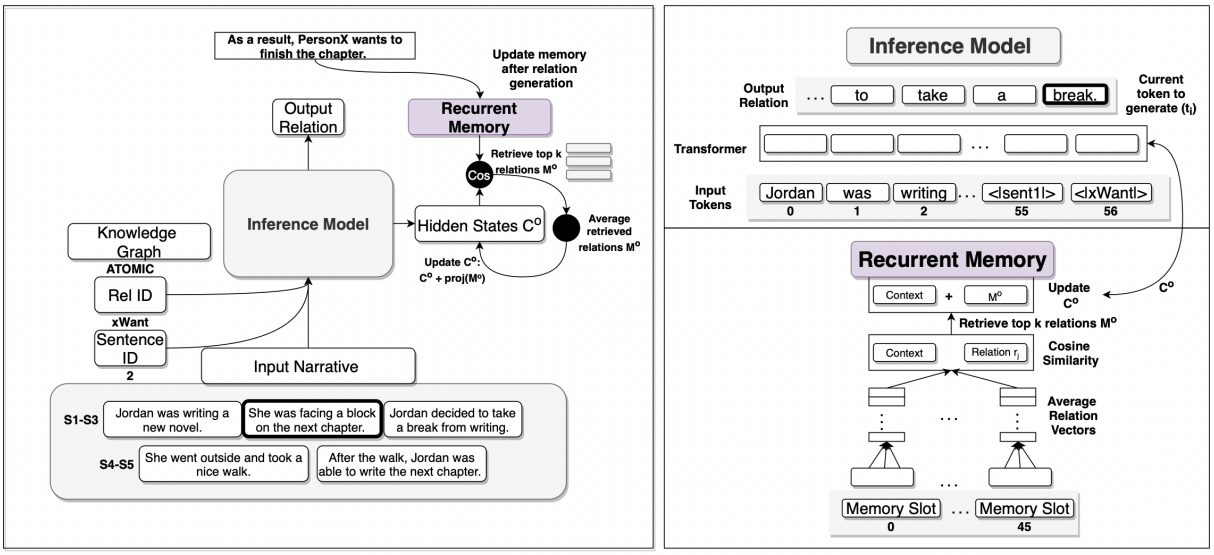

Fig. 2. Overview of the ParaCOMET architecture. Taken from [11].

Another parallel work that has tackled contextual commonsense inference is GLUCOSE[4]. GLUCOSE annotates the ROCStories[13] corpus along ten dimensions of commonsense. The authors categorize the knowledge into ten dimensions, however, to convert this into our format described in section 2.1, we utilize the connectives (e.g., Causes, Enables, etc.) that the GLUCOSE authors present, as the relation types in our tuple, and expect the model

to understand the different dimensions implicitly. From here on out, we utilize relation to refer as an assertion of a certain dimension with a specific connective. They annotate every sentence with an assertion that is either present or implied in it. Additionally they annotate each assertion with a general version of it which includes variables and descriptions of these. What this means is that any person(s) or object in the assertion are replaced with a token such as *Person_A*, etc. to represent a general version of the fact. Given the example story:

John is a regular person who has a dog. John, every day, goes out to walk his dog. A dog is man's best friend. Dogs like to bark at other dogs.

An example of a story specific assertion can be:

"*John* has a *dog*, causes, *John* to walk the *dog*".

An example of a general version of the story specific assertion, according to the GLUCOSE format, is:

"*Person_A* has *Something_A* (that is a dog), causes, *Person_A* to go walk the *Something_A* (that is dog)".

With this corpus of annotated stories, the authors train a T5[15] model to, given a dimension, target sentence, and story, generate both a story specific and general assertion.

However, none of these works address controllability in the generation, which means that the models can generate assertions that may be irrelevant to the sentence, or may not be about a topic needed for a downstream application. Additionally, these models are only trained on one dataset at a time, which can hinder a model's capability to infer knowledge if it has not seen the knowledge elsewhere. Lastly, these models do not score the factuality or correctness of the assertions; at most they can generate a beam score, which indicates the likelihood of the phrase being generated.

In this work, we address all of these shortcomings, by first constructing a dataset consisting of all of the assertions in ConceptNet[2], ATOMIC 2020[3], and GLUCOSE [4]. To construct this dataset, we align the ROCStories[13] with an assertion by generating sentence/paragraph embeddings for the stories and the assertions by using the sentence-transformers [16] library. We then use cosine distance to find the closest story for each assertion, and we repeat the process once more, only with the sentences in the nearest story, to find the closest sentence in the story to the assertion. Additionally, we contextualize the assertions in ATOMIC 2020 to make them even more relevant to a story. This contextualization and alignment puts all the datasets in the same universe. Secondly, we augment this dataset of aligned assertions, stories , and target sentences, with "hints" which are parts of a target assertion, along with a symbol identifying the parts. We define an assertion here as a tuple that represents a fact. This tuple contains at least a subject, a relation type, and a object (similar to subject-verb-object triples). We add an additional field to this tuple which is *specificity*. We define *specificity* as whether the assertion's content is tailored to the aligned story or if it is a generalized version of an assertion. This can be seen as whether the assertion is a *general* template with variables, or a *specific* instance of this template. Hinting helps tell the model that we want a part of an assertion to show up in the generation (i.e., if a hint has a certain subject, that subject should show up in the inferred assertion). Thirdly, we use this dataset to adversarially train two language models; one to infer assertions from the story and target sentence, and a second to validate the assertion given the story and the target sentence.

Altogether our contributions are:

- The utilization of a hinting mechanism to help condition and control a generative model for contextual commonsense inference
- A simple method for contextualizing assertions to a given text with the purpose of performing joint inference
- A method for adversarially training language models to infer and evaluate assertions from a story context

2. Hinting for Controllable Generation

2.1. What is hinting?

Recently, there has been work on exploring *prompting* strategies [1] for pre-trained, transformer-based language models [17, 18]. These are methods which alter the input to a language model such that it matches or approximates

templates that it has seen during pre-training and can reuse or exploit this information. Prompting helps achieve higher performance in tasks with less training data, can help with controllability in the case of text generation, and is more parameter-efficient and data-efficient than fine-tuning, in some cases [19]. One type of prompting is *prefix prompting* [19, 20]. Prefix prompting consists of altering a language model's input (i.e. prefix) by adding additional words. These words can be explicit hard prompts such as actual phrases or words, or they can be soft prompts, embeddings that are input into a model and can be trained to converge on some virtual template or virtual prompt that can help the model.

Prompting holds potential for improving contextualized commonsense inference. We utilize the idea of a *hint*, a hybrid of hard and soft prompts. We define a *hint* as the part(s) of an assertion that a model has to predict, along with special identifiers for these parts, wrapped within parenthesis characters. A forthcoming companion paper focuses on the hinting mechanism. We include a brief description of it here, but we note that the focus of this work is the adversarial generation with the joint inference training. Because hinting is an essential component for controllability of this model, we include this overview.

Syntactically, a *hint* takes the form of: "([specificity], [subject symbol,subject], [relation symbol,relation], [object symbol,object])" where the actual content of the hint between the parenthesis would be a permutation of all but one of the elements in the target tuple. Hints are provided during training by sampling a binomial distribution for each element in a minibatch, which determines whether to give a hint or not. The actual content of the hint would then be generated by randomly sampling without replacement up to all but one of the elements in a target tuple. We now give a complete example of an input to a language model with hinting underlined:

John is a regular person who has a dog. John, every day, goes out to walk his dog. ***A dog is man's best friend.***
Dogs like to bark at other dogs. (<specific>, <subj>A dog, <relation>is a)

Such an input on a model trained for contextual commonsense inference could predict the assertion:

<specific> <subj>A dog, <relation>is a <obj> animal

Here we can see more clearly that whenever a hint is provided, a model trained with hinting tends to produce generations that include the components given in the hint. We utilize hinting in training our models from here on out unless otherwise stated. The controllability that hinting enables can permit us to use models trained with it in downstream applications such as contextual knowledge graph generation.

2.1.1. Discourse-aware/Contextual commonsense inference

Commonsense inference is the task of generating a commonsense assertion. Discourse-aware/contextual commonsense inference is the task of, given a certain narrative or discourse, inferring commonsense assertions that are coherent within the narrative[21]. This task is particularly hard because commonsense knowledge may not be explicitly stated in text [7] and the model needs to keep track of entities and their states either explicitly or implicitly. Research into the knowledge that pre-trained language models learn has yielded good results in that they do contain various types of factual knowledge, as well as some commonsense knowledge[8, 9, 22]. The amount of commonsense knowledge in these models can be improved by supplementing sparsely covered subject areas with structured knowledge sources such as ConceptNet [2, 9].

Knowing that these pre-trained language models may contain some commonsense information has led to the development of knowledge models such as COMET[10]. This line of research has been extended from the sentence-by-sentence level in COMET, to the paragraph-level in ParaCOMET [21]. Contemporaneously, GLUCOSE [4] builds a dataset of commonsense assertions that are contextualized to a set of stories, and generalized (e.g., *John is a human* is generalized to *Someone_A is a human*).

Our general task of contextual commonsense inference can be formally described as follows. We are given a story S composed of n sentences, $S = \{S_1, S_2, \ldots, S_n\}$, a target sentence from that story, S_t, where $S_t \in S$, and a relation type R. Given all this, we want to generate a tuple in the form of $(specificity, subject, R, object)$ that represents an assertion, present or implied, in S_t given the context S, and the relation type R.

2.2. Experimental Setup

We ran some tests on applying hinting to the formulation that GLUCOSE has for its contextual commonsense inference. Recall that GLUCOSE's description of the contextual commonsense is that given a relation to predict along for, a story, and a target sentence, a model has to infer an assertion of that dimension type along with a generalized version of that assertion. The authors use a T5 model in a sequence-to-sequence manner. In this formulation, the source text is composed of a prefix of a dimension to predict $D \in 1, 2, \ldots 10$[1], followed by the story S, which is composed of n sentences, $S = \{S_1, S_2, \ldots, S_n\}$. In GLUCOSE, the target sentence that we want the model to focus and generate inferences on, S_t, is marked with $*$ before and after the sentence. An example input is: "1: The first sentence. *The target sentence. * The third sentence.". In this task we have to infer a general and context specific subject, object and a relation. For our hints we provide up to five out of these six things during training, along with a symbol that represents whether it is the subject, object, or relation, and another symbol that represents the specificity. We add our hint after the story S, utilizing the prefix "hint:" and supplying the *hint* between parenthesis.

We ran these experiments for 1 epoch on the data. Additionally, we utilize a linear warm up of 3000 steps. We utilize the ADAM optimizer [23] with a learning rate of 1e-5, a train batch size 4, and a max source length of 1024 and max target length of 384. We utilize the training and testing data given by the GLUCOSE work. In this set of experiments, we calculate the scores on the test set every 2000 iterations and these are the scores we utilize to report our aggregation. We report the scores given by SacreBLEU[24], ROUGE [14], and METEOR[25] using the datasets library [26] metrics system to get a sense of how good the commonsense inferences are, and we run the experiments with 4 different variations or seeds.

2.3. Effects of hinting

Model	BLEU		METEOR		ROUGE-1		ROUGE-2		ROUGE-L		ROUGE-L-SUM	
	No Hint	Hint	No Hint	Hint	Hint	Hint	No Hint	Hint	No Hint	Hint	No Hint	Hint
T5 (Avg. Max)	40.242	**44.367**	40.550	**43.069**	53.881	**56.209**	34.702	**36.817**	50.321	**52.61**	50.329	**52.617**
T5 (Avg.)	30.237	**32.147**	33.953	**34.761**	46.925	**47.562**	28.334	**28.791**	43.32	**44.08**	43.330	**44.083**
T5 (Median)	33.578	**36.002**	36.163	**36.787**	49.787	**50.496**	30.864	**31.479**	46.149	**47.257**	46.143	**47.293**

Table 1

Results hinting experiments utilizing the formulation for contextual commonsense inference. We include the average and median of the runs, and the average of the maximum values per run. The largest scores are **bolded**. Observe that on the "Hint" columns, which signify the scores that were produced by a model that was trained with hinting, the model achieves greater automated scores.

In this task, we notice two large benefits when we include hinting. Firstly, it improves automated metrics, which means it gives more accurate commonsense inferences than without it. We can see in Table 1 that for the GLUCOSE like formulation, simple hinting improves scores on average more than a point. We suspect that this can be attributed by the fragments of new information that the hint provides to the model to encode. Secondly, and most importantly, hinting serves as a control signal for the generative language model. What this means is that we can supply parts of a hint and have the model fill in whatever parts are missing.

3. Joint Inference of Commonsense Assertions

Given that we have presented hinting, which is a method to help with the controllability of contextual commonsense inference generation, we now look at how we can combine multiple knowledge bases for this task. We then look at combining both hinting and this joint inference towards our task of contextual commonsense inference.

[1]The definition for each dimension number is given in the GLUCOSE work

3.1. What is joint commonsense inference?

In this work we define *joint commonsense inference* as inferring commonsense knowledge assertions by leveraging knowledge from multiple knowledge bases. Recall the story that we have been utilizing as our example:

John is a regular person who has a dog. ***John, every day, goes out to walk his dog.*** *A dog is man's best friend. Dogs like to bark at other dogs.*

From this story, we want to infer the general version of the commonsense assertion of "John is capable of walking his dog", derived from **the second sentence**. This general version can look similar to "Someone_A who has an animal (that is a dog) enables Someone_A to walk the animal (that is a dog)". To generalize this, we must know that: *John is a person's name*, which we can find from a semantic tagger. A much more commonsensical fact needed to infer the assertion is that: *a dog is an animal*, which is a fact found in ConceptNet. Lastly, to infer the assertion, we need to know that: *A person having a dog has the effect that a person goes to walk their dog*, which is a fact that we could find from Atomic. Therefore to infer the general assertion that we presented, we must join information from *at least* two knowledge bases to infer our general assertion. Joint commonsense inference is useful because it could lead to implicitly applying or combining knowledge and/or analogies that might be present in the different knowledge sources, which may lead to better results when performing contextual commonsense inference.

3.2. Joint Inference Approach Overview

We apply this joint inference to the task of contextual commonsense inference, which we presented in section 2.1.1. To be able to perform joint contextual commonsense inference by leveraging multiple knowledge bases we propose the following approach:

1. *For each knowledge base that we have, we convert each facts found in them into a tuple format of* ***{subject, relation, object, specificity}***[2]*. We note that each part of the tuple must be text*[3] *(i.e., if a relation is symbolically "IsA", the text version would be "is a").*
2. *We align each knowledge base tuple with a story (e.g., the ROCStories corpus) and target sentence from the story. The target sentence is the sentence which is most likely to be used to infer the tuple. We give details of this alignment in section 3.4.*
3. *We combine into one list and shuffle, the aligned knowledge base tuples from multiple knowledge sources.*
4. *We replace the naming scheme of variables that may be present in general assertions with the naming scheme from GLUCOSE.*
5. *We train a contextual commonsense inference model on this dataset, whose inferences are joint inferences.*

By following this procedure, we will end up with a dataset of story aligned assertions. In this dataset, all of the assertions are grounded in the same set of stories. With this we can train models that can perform joint contextual commonsense inference. Now we will go into some details of this process.

3.3. Specificity in Assertions

Recall that we define *specificity* as whether the assertion's content is tailored to the aligned story or if it is a generalized version of an assertion. This can be seen as whether the assertion is a *general* template with variables, or a *specific* instance of this template. To make the difference between *specific* and *general* assertions clearer we give the following example. Using the same story as before:

John is a regular person who has a dog. ***John, every day, goes out to walk his dog.*** *A dog is man's best friend. Dogs like to bark at other dogs.*

[2] This follows a similar pattern to subject-verb-object triples, but has the added field of specificity which is whether the assertion is contextual to a story, or a generally applicable assertion

[3] This ultimately helps us express the assertion in a textual way (i.e., (a dog, IsA, animal) when converted to the tuple (a dog, is a, animal, specific) and passed to a string representation function can be expressed as "Specifically, a dog is a animal".

As before, we focus on the second sentence: **John, every day, goes out to walk his dog.**. From here, we can infer the *specific* assertion: "John is capable of walking his dog". The assertion is *specific* because it fills out a broadly applicable template, which we will present next, that speaks about John and his dog from the story. From the sentence, we can also infer the *general* version of the assertion: "Someone_A who has Something_A (that is a dog) enables Someone_A to walk the Something_A (that is a dog)". This latter assertion is *general* because it speaks in a template format (i.e.,broader terms) the same fact. A *general* assertion is not the story-dependent instance of the template, but the broader, story-independent template. These *general* assertions contain variables in them.

In this work we utilize ConceptNet, Atomic 2020, and GLUCOSE as our knowledge bases, and propose to combine them to perform joint inference. In table 2 we give the different available *specificities* for these knowledge bases. From this, we can see that ConceptNet does not have *general* specificity assertions. Although this may

Knowledge Base	General	Specific
ConceptNet	✗*	✓
Atomic 2020	✓	✗*
GLUCOSE	✓	✓

Table 2

Here we can see the available specificities in ConceptNet, Atomic 2020, and GLUCOSE. We mark with ✗ the specificities that are not available by default, and add * to those that can be generated.

sound counterintuitive, ConceptNet gives *specific*, untemplated, instances of assertions, in contrast to Atomic and GLUCOSE, which describe *general* versions of assertions. Atomic 2020 has the opposite problem, it gives *general* versions of rules, (e.g., PersonX participates in some event, has some effect on PersonX or Y around them), and does not give, within our contextual commonsense inference framework, the *specific* instance of the templates (e.g., filling out PersonX, PersonY, etc.). To remedy this lack of specificity within two of our sources, we mention ways to generate examples of the missing specificity, and implement the solution for Atomic 2020.

3.3.1. Generating Missing Specificity

ConceptNet To generate *general* assertions for ConceptNet, we could possibly run a classifier that would determine whether a given set of tokens is a, person, place, object, among others. With this information we could fill out, as an example, the template that GLUCOSE broadly utilize which is: {Category}({Description}), relation, {Possibly Other Category} ({Possibly Other Description}). From ConceptNet, we could find the relation: "a dog, IsA, animal". A *general* version of this assertion can be "Something_A (that is a dog), IsA, Something_B (that is a animal)". Although we describe this process, we do not implement it in our work.

Atomic 2020 To generate *specific* assertions for Atomic 2020 we can do the following. We can first identify variables (PersonX, PersonY, etc.) that are in the assertion. We can then replace these variables with a mask token from a language model that was trained with the masked language modeling objective [18, 27], and use the language model to fill in the Mask token similar to a cloze (i.e., fill-in-the-blanks) task. To give the model sufficient context, we insert the assertion to the right of the nearest aligned sentence (we describe the process to get this in the next section). In the case of Person*N* variable, this usually leads to the variable being replaced with a character from the story. In addition to this, Atomic 2020 contains blanks demarcated by underscore characters (i.e., ____), which we can once more replace with a mask token and have the model fill it out with the given context. We use this process in our work, filling in the blanks with a ROBERTA[27] large model.

3.4. Aligning Assertions with Stories

To align assertions with stories we do of the following procedure. On a high-level, we vectorize the stories and we vectorize the assertions and we then utilize the cosine distance to find the nearest story for each assertion. We then go into more detail and repeat the same procedure (i.e., vectorization) for each sentence in the previously found nearest story. Ultimately we are left with the story and sentence that is most relevant or similar to the assertion. On a low-level, we utilize the sentence transformers package along with the "paraphrase-mpnet-base-v2" model from the repository, to generate a representative vector for every story and for every assertion from each of our knowledge

bases. We then utilize the FAISS package [28] to perform fast approximate cosine similarity search to find, for each assertion, what is the nearest story. Once we have this nearest story we again utilize the sentence transformers model to vectorize every sentence in that story along with the FAISS package for the cosine similarity search, to find the nearest sentence to the assertion. This process can be visualized and figures 3 and 4.

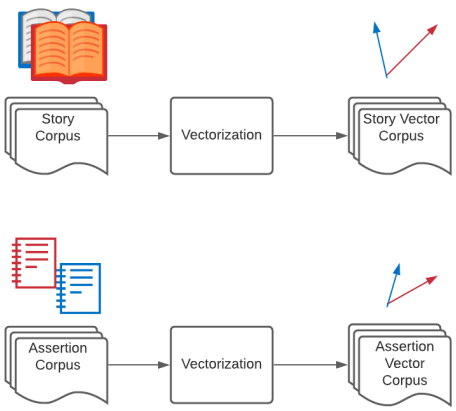

Fig. 3. Step 1: The story and assertion corpus are vectorized. In our work we utilize the sentence-transformers package [16] to achieve this.

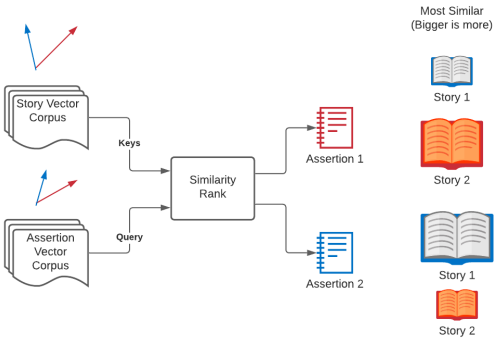

Fig. 4. Step 2: The resulting assertion vectors are utilized as queries, and the resulting story vectors are used as keys for a memory-like lookup. In this work we use the FAISS package for this. The output of the memory-like lookup is the nearest story for each vector. This process is repeated for the sentences in the nearest story, to align the assertion with a sentence.

3.5. Experimental Setup

To evaluate the effects of joint inference by combining multiple knowledge bases in the task of contextual commonsense inference we do the following. We generate a story aligned assertion dataset for each knowledge base individually (i.e., for ConceptNet, for Atomic 2020, and for GLUCOSE) as described in the previous sections. Once we have generated a dataset for each, we proceed to perform combinations of the datasets: ConceptNet-Atomic 2020, ConceptNet-GLUCOSE, GLUCOSE-Atomic, ConceptNet-Atomic-GLUCOSE. For the individual and the combined datasets we perform two sets of automated tests. One that includes hinting the specificity, subject, and relation during evaluation, and the other without these. The rationale behind these two setups is that we want to evaluate what the model infers without any guidance, and see what it infers with guidance and joint and with multiple sources. To train our models, we use a batch size of 50, on 4xNVIDIA A6000, a learning rate of 1e-5 for an ADAM[23] optimizer, and 3 epochs over the data.

We note that the data for ConceptNet we utilize is the dataset given by [29], specifically the data in the “train_600k.txt" which are approximately 600,000 examples of assertions from ConceptNet, and as a test set we utilize the “test.txt" that they provide. For Atomic 2020 we utilize the training and testing data files provided by the authors [3]. Lastly, for GLUCOSE we use the training and evaluation files also provided by the authors in the corresponding repository.

Additionally, we look into running a small Mechanical Turk evaluation of generated test assertions, because we suspect that automated metrics may hurt the model’s evaluation when not using hinting. We sample 100 entries from the testing files of each knowledge base (Atomic 2020, ConceptNet, and GLUCOSE), and run these through a set of models trained firstly with only one of the test knowledge bases (i.e. a model trained only ConceptNet, a model trained in Atomic 2020, and a model trained in GLUCOSE) and secondly a model trained with the combination of knowledge bases and evaluated with and without hinting. We take the generated inferences and ask 2 raters from Amazon Mechanical Turk to determine whether the assertion is acceptable, whether it is acceptable with the context that it was aligned with, and whether the gold standard assertion was acceptably aligned with the context. We mark as acceptable the answers that both human annotators agree as valid and the others as invalid.

Fig. 5 & 6 & 7 . Automated metrics (BLEU, Meteor, and ROUGE F1) for gaging effects of joint inference with hinting. Each set of colors (3 sets in total, Atomic 2020, GLUCOSE, ConceptNet) represents testing on a different test set, and from top to bottom, we remove knowledge bases. Overall, we can see that with hinting on the test set, the addition of knowledge bases for inference does not improve nor degrade substantially the performance. We suspect that with hinting the model is able to channel the specific knowledge base that pertains to the test set.

3.6. Effects of joint inference

From our experiments in this area we can see a couple of things. Firstly, when used with *hinting*, joint inference does not seem to improve the performance of synthetic tests. What this may mean is that *hinting* manages to somehow utilize the format of the knowledge base that it is tapping into for information. Additionally, some of the knowledge sources that we are using have a little overlap (GLUCOSE and ConceptNet had approximately 0.34% of overlap[4], and ATOMIC 2020 has approximately 9.4% of overlap with ConceptNet[3]), which means that once hinting is utilized to give control signals to the models, this lack of overlap may attribute to why the metrics do not decrease drastically. Secondly, without *hinting*, in the automated tests that we run, performance seems to degrade when we add knowledge bases. The reason for this may be that the model thinks that an assertion from another knowledge base may be more relevant than the test assertion types that we are evaluating. An example of this can be if we combined GLUCOSE and ConceptNet. When evaluating with automated metrics against a test set of GLUCOSE assertions, it may be the case that our joint model generates a ConceptNet-like assertion with different relations than the ones that GLUCOSE utilizes, which would lead to performance decreases in automated metrics. However, when we use *hinting* and combine our multiple knowledge sources to perform this joint inference, we are able to within one model, fit all of the knowledge bases, that we are evaluating. This has implications for downstream applications because they no longer require multiple models. With one model and *hinting* we can do what three separate models usually do. The results for our Mechanical Turk study can be found in table 3. We can see that without hinting, the joint inference model (i.e Atomic 2020 - ConcetpNet - GLUCOSE - No Hint) improves the acceptability, both with and without context, of assertions predicted in the Atomic 2020 test set. We can also see that performance does not degrade much in whether it produces assertions that are contextually acceptable throughout the test sets. We can see that with hinting however, the performance is decreased and becomes closer to what the

Fig. 8 & 9 & 10 . Automated metrics (BLEU, Meteor, and ROUGE F1) for gaging effects of joint inference without hinting. Each set of colors (3 sets in total,Atomic 2020, GLUCOSE, ConceptNet) represents testing on a different test set, and from top to bottom, we remove knowledge bases. Overall, we can see that without hinting on the test set, the addition of knowledge bases for inference decreases the performance. We suspect that this is not due to wrong inference, but that it may be more likely that an assertion from a knowledge base different than what the test set is from was inferred (i.e., when testing on GLUCOSE, the model predicted a ConceptNet assertion)

individually trained models can achieve. This suggests that with hinting, the model tries to channel the knowledge base that we are targetting, and aligns to what we see in the automated metrics.

We also note that on average our alignment technique has 60% approval rate for Atomic 2020, 68.3% for ConceptNet, and 69% for GLUCOSE, which gives us on average 65% approval for our alignment strategy of using sentence-transformers with the FAISS similarity search.

4. Adversarial Language Models

4.1. Adversarially training language models

In this work, our main contribution is providing and demonstrating the usefulness of a method for adversarially training language models for the task of contextual commonsense inference. In the broader literature of generative adversarial networks (GANs)[30], the adversarial training of models, leads to better results than training each model individually, possibly because of the generator network being updated with gradients flowing through the discriminator. Additionally, we get the benefit that our discriminator model can give us a 0-1 score of an assertion.

In a high level view, we want to use a generative language model (i.e., a generator) who is given a story and a target sentence, and is expected to infer commonsense assertions that are relevant to the target sentence within the context of the story. Additionally, we want to be able to tune this generator with a discriminator model whose inputs are the same story and target sentence along with the generated fact. The discriminator can first determine whether the fact came from a generative model or not and secondly can determine whether the fact is valid or not. This

Model	Acceptable	Contextually Acceptable	Alignment Acceptable	Acceptable	Contextually Acceptable	Alignment Acceptable	Acceptable	Contextually Acceptable	Alignment Acceptable
Atomic 2020 - No Hint	0.70	0.66	0.6	-	-	-		-	-
ConceptNet - No Hint	-	-	-	0.77	**0.71**	**0.72**	-	-	-
GLUCOSE - No Hint	-	-	-	-	-	-	**0.81**	**0.68**	**0.8**
Atomic 2020 - ConceptNet-GLUCOSE - No Hint	**0.76**	**0.68**	**0.63**	**0.83**	0.7	0.65	0.79	0.67	0.59
Atomic 2020 - ConceptNet-GLUCOSE - Hint	0.71	0.53	0.57	0.77	0.64	0.68	0.77	0.64	0.68

Table 3

Results for human annotation of 100 randomly sampled assertions from Atomic 2020, GLUCOSE, and ConceptNet test sets and the inferred commonsense from these. We have three sets of three columns Acceptable, Contextually Acceptable, and Alignment Acceptable. Each set of columns is color-coded to represent a knowledge base. Firstly, the Acceptable column is the ratio of whether humans thought that inferred assertions, without context, were acceptable commonsense or not. The Contextually Acceptable, column represents the ratio of whether humans thought that inferred assertions given the context, were acceptable or not. Lastly, the Alignment Acceptable column is whether humans thought that the gold standard (from a knowledge base) assertion was correctly matched to the context. We can see that without hinting, the joint inference model (i.e Atomic 2020 - ConcetpNet - GLUCOSE - No Hint) improves the acceptability, both with and without context, of assertions predicted in the Atomic 2020 test set. We can also see that performance does not degrade much in whether it produces assertions that are contextually acceptable throughout the test sets. We can see that with hinting however, the performance is decreased and becomes closer to what the individually trained models can achieve. This suggests that with hinting, the model tries to channel the knowledge base that we are targetting, and aligns to what we see in the automated metrics.

general architecture can be seen in Figure 11. One interesting aspect of this formulation, is that it becomes a kind of conditional GAN [31], which could reinforce the control signals given in hints.

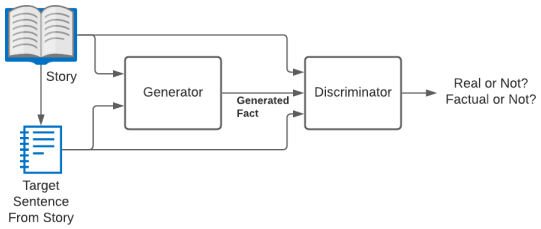

Fig. 11. Overview of the proposed GAN architecture. A story and a target sentence are fed into the generator, which infers a contextual commonsense fact. This fact, along with the story and target sentence, are passed into a discriminator to determine whether it is from a generator or not and whether it is factual or not.

To be able to achieve this architecture, we need to be able to connect a generative language model to a regular language model with some additional final layers that produce a score. In our work, we utilize a transformer-based encoder-decoder generative model. Specifically we use the BART model [32]for conditional generation, provided by the Huggingface Transformers library[33]. To discriminate, we utilize a regular BART model, and add some additional layers that compress the output representation and produce a single final score from 0-1. However, it is not as simple as conditionally generating, and passing into the discriminator the generated assertions. The generative process utilized is a recurrent next token generation. Recall that to pick a next token with this method, a non-differentiable *argmax* operation is utilized. This impedes the gradients from being calculated in backpropagation. The issue becomes even more complex, in that the generation process can utilize beam search to find even better generations, and each beam at the end of each generation step selects a next best token also with an *argmax*. To address this discontinuity, we utilize an approximation of the *argmax* (i.e., a soft *argmax*) described it in the next section. Finally, we pick two of the same types of base model (e.g., BART), in order for both the generator and discriminator to share a vocabulary. The reason for sharing the vocabulary is addressed in the next section, however this may not be necessary and we give an alternative way of being able to “splice" together different models for this task in section 4.4.

4.2. Addressing the Discontinuity in Generation

Recall that during recurrent conditional language generation, a next token, N, is selected by finding the *argmax* of a $softmax$ of all the vocabulary, after a language model is given the generated phrase up until step $N-1$. Also recall that an embedding layer is a neural network component that given an index i , returns a row vector, from a vocabulary matrix, that corresponds to i. This lookup operation can also be achieved by performing a dot product of a one hot vector that represents the index i and the vocabulary matrix. This essentially scales every row in the matrix by the corresponding vector component and sums all the vectors. In the case of a one-hot vector, it scales all but one vector to zero, therefore leaving only the desired i at the end of the summation.

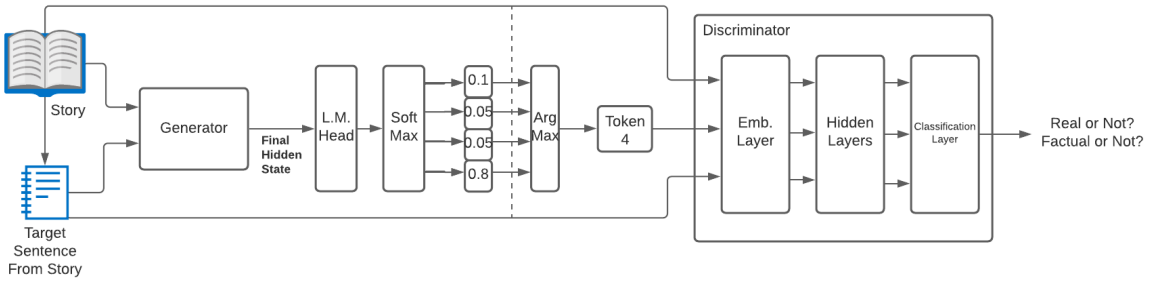

Fig. 12. We visualize an example that shows the discontinuity when combining a generative language model with a discriminative language model. The dashed line represents where the gradients are discontinued because of the non-differentiable *argmax* operation.

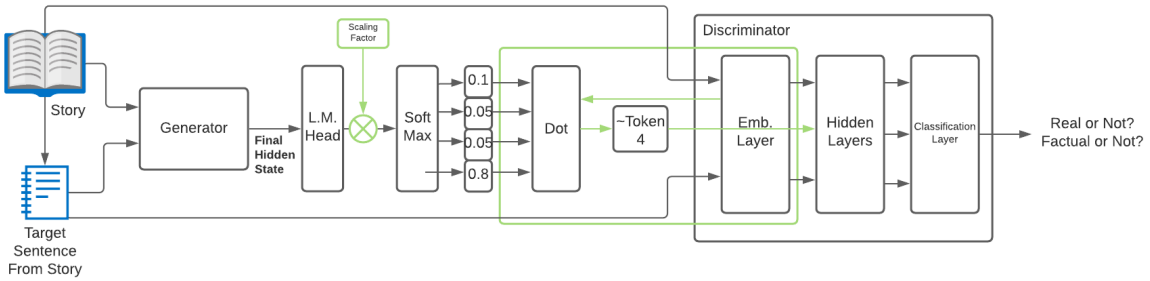

Fig. 13. We visualize an example that shows how we address the discontinuity by replacing the non-differentiable *argmax* with a dot product between the softmax and the embedding layer matrix. Additionally we highlight where the scaling factor is inserted to make the approximation more accurate. We mark our approach in green.

Now, to maintain the gradients, we need to connect the output of our generative model, which is the softmax, to the embedding layer of our discriminator model so that it can be input, scored, and backpropagated correctly. To do this, we simply replace the aforementioned one-hot vector that represents our index, with the softmax that the generative model produces at a given generation step, and perform a dot product of this softmax with the embedding matrix. This method is approximate, given that there may be noise from other non-zero elements in the softmax, and the top element is not an exact 1. To somewhat remedy this approximation, we can multiply the input of the softmax by a certain factor to essentially give a more approximate one-hot vector. However, this factor cannot be very large, because it may cause instability during backpropagation. In our work, we use a scaling factor of 1, as this seems to be accurate enough. We repeat this approximation for every generation step, and are left with a list of input embeddings for the discriminator that represent the output of the generator with usable gradients. Since we are using the same vocabulary, we can verify how accurate the output is, by training a K-Nearest Neighbors system, and finding the K=1 neighbor of the output of the softmax and embedding matrix dot product against the embedding matrix. We can use this test to determine an appropriate factor for scaling the softmax. Altogether, this approximation permits us to train our two language models adversarially by having gradients flow from the discriminator to the generator.

4.3. Addressing Different Generation Types

The aforementioned approximation for the discontinuity, as we described it, can be utilized for greedy selection of the next-token (i.e., we always pick the maximal one from the final softmax). We can also apply this technique to beam-search generation, and at every point in constructing the top scored beam, we utilize the softmax of the maximal scoring beam, essentially simplifying the problem back down to a greedy generation-like formulation. In this work however, we do not explore top-k generation, top-p generation, nor sampling during generation. Top-k and top-p generation can be seen as masking out with zeros, tokens that do not meet a certain criteria. Sampling is more complicated. To use our approximation with sampling, we would need to model the sampling function at every generation step with something like a recurrent neural network. We leave this line of research for future work.

4.4. Splicing different models

We come back to address the issue of having to utilize models that have the same vocabulary. The reason for this is that the soft argmax operation matches in matrix multiplication dimensions between models. We now give an alternative, although unexplored, option. Given that a generative model will produce a softmax vector of vocabulary size V, and we have another model that has a different vocabulary size of M, we can train decoding layers that can convert the output tokens of the generative model, into corresponding tokens from the discriminative model. However, this conversion layer would need to be trained beforehand, and may need to be frozen during the adversarial training, otherwise it would be a disconnect between the two models, and the input given to the discriminator may be corrupted. To train this conversion layer beforehand, one could use as a ground truth, the results that a tokenizer from model B, with the vocabulary size of M, would use as the targets in a cross entropy loss, and the results that a tokenizer from model A, with the vocabulary size of V, uses as the input to the layer.

4.5. Factuality in the Discriminator

Given that we can now adversarially train our models, we explore enhancing the discriminator with some way to determine factuality. We take a simple approach that in addition to the normal discriminator training objective (i.e., the discriminator is given a batch of generated text and a batch of real text and evaluated whether it inferred this correctly), we add a confounder loss. Our additional confounder loss is based on the confounder loss by [29], in that we shuffle around the subjects and objects and expect our model to determine that when shuffled objects are false. Since our generated outputs are structured (i.e., we have symbol tokens that delimit the different parts of assertions), we can do this shuffling easily. Although shuffling may incur in some false negatives (we may have a shuffled configuration that is factually correct), since we supply the story and target sentence, we expect the discriminator to be able to discern this correctly. We believe that we could also apply the max-margin loss utilized to great effectiveness by other language GAN literature [34, 35], although we leave this for future work.

4.6. Experimental Setup

For our GAN experiments we had the following setup. We built a joint inference dataset using the procedure given in section 3, and we augmented it with hinting. This dataset was composed of 1,479,811 training examples, and 30,183 testing examples. We fed this data to a model with the adversarial setup described in the previous sections. The generator model utilized was the BART-base for conditional generation, and the discriminator model was a BART-base model with 3 1D-Convolutional layers each of which compresses the final output by a factor of two right after the final hidden output of the model. This is followed by a pooling layer, and three linear layers which compress even further the representation. The output of the final linear layer is a vector of dimensionality 1 to produce a score. We note that we apply a non-linearity at the output of the convolutional layers, and at all but the last linear layer. We took this approach because it seemed like a useful way to filter through the tokens for important parts of information, and then compress this representation. For the sake of time, we run our model on only 1,000,000 examples, with a batch size of 50, on 4xNVIDIA A6000. We run only one epoch through the data. After this, we run our model on our test set without hinting and list automated metrics composed of ROUGE scores, METEOR scores, and BLEU

Model	METEOR	BLEU	rougeLsum	rouge1	rouge2	rougeL	Acceptable	Contextually Acceptable	Alignment Acceptable
Generator - Non-GAN	**63.961**	**36.473**	**42.345**	**45.758**	**15.033**	**42.35**	0.735	**0.74**	**0.68**
Generator - GAN	62.141	32.113	41.511	44.524	11.353	41.517	**0.74**	0.735	0.645

Table 4

Here we can see the results of GAN training in models. It appears that our naive GAN approach is bested by the non-adversarial generator trained only with the language modeling objective. We note however, that with human judgements, both models are on par in both Acceptable and Contextually Acceptable inferences.

scores to get an idea of the model's general capabilities. We train a model with and without the adversarial approach that we propose. We run an additional user study, which consists of randomly sampling 200 entries from our test data, and running the same setup as the study described in section 3, we ask a pair of human annotators to determine whether the generated assertions are valid, and whether they are valid for the given context; the same setup as for our joint inference.

4.7. Effects of adversarial training

After running automated tests and human annotated tests, we can see that our GAN system is on par with a non-GAN generator. We suspect that our approach may be too naive, and possibly an improved GAN formulation such as the Wasserstein GAN [36] used in [37] may help our results. We also note that we ran a quick test with the Spearman Rank Correlation coefficient [] and we found that the discriminator of the GAN system has some correlation (0.121) with the Contextual Acceptability measure. This may indicate that there is still room to improve in the Discriminator, and we may look into other training objectives for it. Looking further into the results, we see that the non-GAN model trumps our GAN formulation in all of the metrics, suggesting that the Discriminator could be restricting the Generator too much.

5. Related Work

5.1. Prompting

Recently, there has been a shift in Natural Language Processing from pre-training and fine-tuning a model, to pre-training, prompting, and predicting [1]. One reason for this shift is the creation of ever-larger language models, which have become computationally expensive to fine-tune. Prompting is a finding a way to convert a model's input sequence into another sequence that resembles what the model has seen during pre-training. Overall, most prompting research focuses on formulating the task as a *cloze* (fill-in-the-blanks) task. However, we consider the task of language generation, an open-ended formulation.

Recall that prefix prompting modifies the input to a language model, by adding either a hard prompt (additional words to the input sequence)[38] or a soft prompt (i.e., adding trainable vectors that represent, but are not equivalent to, additional words) [1, 19, 20]. Unlike classic prefix prompting, *hinting* uses both hard and soft prompts. The soft prompts are in the form of symbols that represent the different parts of the assertion (i.e., subject (<*subj*>), relation type (<*relation*>), and object (<*obj*>)), and the hard prompts are in the form of the actual parts of the assertion that are selected to be appended as part of the *hint* as seen in our example in section 5.1. Hinting is similar to KnowPrompt[39], except that they use a masked language model and soft prompts for relationship extraction. AutoPrompt [38] is also similar, but finds a set of "trigger" words that give the best performance on a *cloze*-related task, whereas we provide specific structured input for the model to guide text generation.

5.2. Controllable Generation

Controllable generation can be described as ways to control a language model's text generation given some kind of guidance. One work that tries to implement controllable generation is CTRL [40]. The authors supply control signals during pre-training of a language model. This approach is intended to provide a generally applicable

language model. A body of work in controllable generation has focused on how it can be used for summarization. Representative work that uses techniques similar to ours is GSum [41]. In contrast to GSum, our method is model independent, allows for the source document to interact with the guidance signal, and contains soft prompts in the form of trainable embeddings that represent the parts of a tuple. The GSum system gives interesting insight into the fact that highlighted sentences, and the provision of triples, does in fact help with the factual correctness of abstractive summarization.

5.3. Story and Assertion Alignment

The closest work to ours, with regards to constructing an story aligned assertion dataset is ParaCOMET and GLUCOSE. GLUCOSE uses human annotation to perform the alignment between stories and commonsense assertions. ParaCOMET takes an automated approach in which assertions are aligned either by giving the sentence to a COMET model as an input and producing a relevant inferred assertion, or by calculating the cross entropy of combining the story up until the target sentence with an assertion from a knowledge base. Our method differs from this in that we utilize cosine distance between semantic representations of the story and its sentences and an assertion from a knowledge base. Some possible differences that arise from this is that our method could match assertions that may not be explicit in a story to that story. Whereas ParaCOMET's approaches, which are based on cross-entropy for coherence, are likely to produce assertions that have parts that are explicit in text. Overall, our approach can match more abstract assertions to stories. Additionally, our method permits us to use the optimized FAISS library to scale up to billions of stories and assertions, and gives us the freedom to select how to embed the stories/sentences/assertions.

5.4. Commonsense Grounding and Reasoning

A similar line of work has been in grounding commonsense statements for inference. However, this line of work is more aligned for natural language inference rather than assertions. One work in this area is HellaSwag[42] which constructs a question answering dataset whose plausible answers are intended to be confounders to language models. Our work differs from this line of work in that we intend to produce structured outputs.

Other works look at reasoning with commonsense knowledge graphs. One work that utilizes the explicit graph structure to perform multi-hop reasoning is "Commonsense for Generative Multi-Hop Question Answering Tasks"[43]. The authors look to select grounded multi-hop relational commonsense information from ConceptNet via a pointwise mutual information and term-frequency based scoring function to fill in gaps of reasoning between context hops for a model they use. In contrast to this work, we are not explicitly looking at a graph structure for our inference, nor are in a task of question answering.

Some older work that looks at doing something similar to joint inference is blending[44]. This technique essentially consists of constructing and adding or blending together matrices of embeddings to find the commonalities between discrete knowledge sources and commonsense knowledge. This method however is hard to scale to large knowledge bases, and is not easily applied to the task of contextual knowledge inference. An even older work that looks into a certain kind of joint inference is Cyc[45]. Cyc uses the idea of "micro-theories"; there would be a small set of commonsense assertions that you could reason with, then combine them with the more general Cyc KB. However, this is not really joint inference in the sense that we use, but trying to address the problem of local vs. global inference.

5.5. Adversarial Language Models

Here we look into work that utilizes adversarial or close training with language models. One such work is [46] in which the authors utilize a GPT-3[47] model as a teacher in order to distill commonsense knowledge into a student model that is considerably smaller. This task is different from ours in that they do not explore contextual/discourse aware commonsense inference, instead they look at extracting the knowledge already found in a model.

Other work more similar to ours, albeit older, is [37]. In this work, the authors take a similar approach to our adversarial configuration, however they utilize the Wasserstein GAN objective [36] rather than the basic GAN

formulation that we use. The authors additionally use the same approximation that we utilize in section **??**. We note that they employ other strategies such as teacher helping, curriculum learning, and variable length that are worth looking into for future work. We note however that the authors tackle general language generation, rather than our task of contextual commonsense inference.

6. Contributions & Takeaways

In this work we have presented three things: a method for controlling contextual commonsense inference called hinting, a method for combining multiple knowledge graphs for joint contextual commonsense inference, and a adversarial and non-adversarial model trained with these techniques. Taken altogether, we can obtain one model that is capable of inferring on a topic given by a hint, the inference can be performed on any subject, relation, object, and specificity from source knowledge bases, and the model's discriminator is capable of scoring assertions. These works serve as a baseline to explore the area of contextual commonsense inference, and leave much room to explore avenues on hinting, joint inference, and adversarial training of transformer-based language models.

References

[1] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi and G. Neubig, Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, *arXiv preprint arXiv:2107.13586* (2021).

[2] R. Speer, J. Chin and C. Havasi, Conceptnet 5.5: An open multilingual graph of general knowledge, in: *Thirty-first AAAI conference on artificial intelligence*, 2017.

[3] J.D. Hwang, C. Bhagavatula, R.L. Bras, J. Da, K. Sakaguchi, A. Bosselut and Y. Choi, (Comet-) Atomic 2020: On Symbolic and Neural Commonsense Knowledge Graphs., in: *AAAI*, 2020, pp. 6384–6392.

[4] N. Mostafazadeh, A. Kalyanpur, L. Moon, D. Buchanan, L. Berkowitz, O. Biran and J. Chu-Carroll, GLUCOSE: GeneraLized and COntextualized Story Explanations, in: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, Online, 2020, pp. 4569–4586. doi:10.18653/v1/2020.emnlp-main.370. https://aclanthology.org/2020.emnlp-main.370.

[5] B.M. Williams, H. Lieberman and P. Winston, A commonsense approach to story understanding, in: *Thirteenth International Symposium on Commonsense Reasoning (Commonsense-2017)*, 2017. http://www.media.mit.edu/~lieber/Publications/Understanding-Stories-Commonsense.pdf.

[6] Z. Zhang, X. Geng, T. Qin, Y. Wu and D. Jiang, Knowledge-Aware Procedural Text Understanding with Multi-Stage Training, in: *WWW '21: The Web Conference 2021, Ljubljana, Slovenia, April 19–23, 2021*, 2021.

[7] H. Liu and P. Singh, ConceptNet—a practical commonsense reasoning tool-kit, *BT technology journal* **22**(4) (2004), 211–226.

[8] J. Da and J. Kasai, Cracking the Contextual Commonsense Code: Understanding Commonsense Reasoning Aptitude of Deep Contextual Representations, in: *Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing*, Association for Computational Linguistics, Hong Kong, China, 2019, pp. 1–12. doi:10.18653/v1/D19-6001. https://aclanthology.org/D19-6001.

[9] J. Davison, J. Feldman and A.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)*, 2019, pp. 1173–1178.

[10] A. Bosselut, H. Rashkin, M. Sap, C. Malaviya, A. Çelikyilmaz and Y. Choi, COMET: Commonsense Transformers for Automatic Knowledge Graph Construction, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[11] S. Gabriel, C. Bhagavatula, V. Shwartz, R. Le Bras, M. Forbes and Y. Choi, Paragraph-level Commonsense Transformers with Recurrent Memory, *Proceedings of the AAAI Conference on Artificial Intelligence* **35**(14) (2021), 12857–12865. https://ojs.aaai.org/index.php/AAAI/article/view/17521.

[12] M. Sap, R. Le Bras, E. Allaway, C. Bhagavatula, N. Lourie, H. Rashkin, B. Roof, N.A. Smith and Y. Choi, Atomic: An atlas of machine commonsense for if-then reasoning, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, 2019, pp. 3027–3035.

[13] N. Mostafazadeh, N. Chambers, X. He, D. Parikh, D. Batra, L. Vanderwende, P. Kohli and J. Allen, A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories, in: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, San Diego, California, 2016, pp. 839–849. doi:10.18653/v1/N16-1098. https://aclanthology.org/N16-1098.

[14] C.-Y. Lin, ROUGE: A Package for Automatic Evaluation of Summaries, in: *Text Summarization Branches Out*, Association for Computational Linguistics, Barcelona, Spain, 2004, pp. 74–81. https://www.aclweb.org/anthology/W04-1013.

[15] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li and P.J. Liu, Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, *Journal of Machine Learning Research* **21**(140) (2020), 1–67. http://jmlr.org/papers/v21/20-074.html.

[16] N. Reimers and I. Gurevych, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2019. https://arxiv.org/abs/1908.10084.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser and I. Polosukhin, Attention is All You Need, in: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 6000–6010–. ISBN 9781510860964.

[18] J. Devlin, M.-W. Chang, K. Lee and K. 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)*, Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. doi:10.18653/v1/N19-1423. https://aclanthology.org/N19-1423.

[19] X.L. Li and P. Liang, Prefix-tuning: Optimizing continuous prompts for generation, *arXiv preprint arXiv:2101.00190* (2021).

[20] B. Lester, R. Al-Rfou and N. Constant, The power of scale for parameter-efficient prompt tuning, *arXiv preprint arXiv:2104.08691* (2021).

[21] S. Gabriel, C. Bhagavatula, V. Shwartz, R. Le Bras, M. Forbes and Y. Choi, Paragraph-level Commonsense Transformers with Recurrent Memory, *Proceedings of the AAAI Conference on Artificial Intelligence* **35**(14) (2021), 12857–12865. https://ojs.aaai.org/index.php/AAAI/article/view/17521.

[22] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu and A. Miller, Language Models as Knowledge Bases?, 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)*, Association for Computational Linguistics, Hong Kong, China, 2019, pp. 2463–2473. doi:10.18653/v1/D19-1250. https://aclanthology.org/D19-1250.

[23] D.P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, *CoRR* **abs/1412.6980** (2015).

[24] M. Post, A Call for Clarity in Reporting BLEU Scores, in: *Proceedings of the Third Conference on Machine Translation: Research Papers*, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 186–191. doi:10.18653/v1/W18-6319. https://aclanthology.org/W18-6319.

[25] S. Banerjee and A. Lavie, METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments, in: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Association for Computational Linguistics, Ann Arbor, Michigan, 2005, pp. 65–72. https://www.aclweb.org/anthology/W05-0909.

[26] Q. Lhoest, A.V. del Moral, P. von Platen, T. Wolf, Y. Jernite, A. Thakur, L. Tunstall, S. Patil, M. Drame, J. Chaumond, J. Plu, J. Davison, S. Brandeis, V. Sanh, T.L. Scao, K.C. Xu, N. Patry, S. Liu, A. McMillan-Major, P. Schmid, S. Gugger, N. Raw, S. Lesage, A. Lozhkov, M. Carrigan, T. Matussière, L. von Werra, L. Debut, S. Bekman and C. Delangue, huggingface/datasets: 1.13.2, Zenodo, 2021. doi:10.5281/zenodo.5570305.

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, *arXiv preprint arXiv:1907.11692* (2019).

[28] J. Johnson, M. Douze and H. Jégou, Billion-scale similarity search with GPUs, *arXiv preprint arXiv:1702.08734* (2017).

[29] X. Li, A. Taheri, L. Tu and K. Gimpel, Commonsense knowledge base completion, in: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1445–1455.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, Generative Adversarial Nets, in: *Advances in Neural Information Processing Systems*, Vol. 27, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K.Q. Weinberger, eds, Curran Associates, Inc., 2014. https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[31] M. Mirza and S. Osindero, Conditional generative adversarial nets, *arXiv preprint arXiv:1411.1784* (2014).

[32] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov and L. Zettlemoyer, BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Online, 2020, pp. 7871–7880. doi:10.18653/v1/2020.acl-main.703. https://aclanthology.org/2020.acl-main.703.

[33] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz et al., Huggingface's transformers: State-of-the-art natural language processing, *arXiv preprint arXiv:1910.03771* (2019).

[34] E.M. Ponti, I. Vulić, G. Glavaš, N. Mrkšić and A. Korhonen, Adversarial Propagation and Zero-Shot Cross-Lingual Transfer of Word Vector Specialization, in: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 282–293. doi:10.18653/v1/D18-1026. https://aclanthology.org/D18-1026.

[35] P. Colon-Hernandez, Y. Xin, H. Lieberman, C. Havasi, C. Breazeal and P. Chin, RetroGAN: A Cyclic Post-Specialization System for Improving Out-of-Knowledge and Rare Word Representations, in: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, Association for Computational Linguistics, Online, 2021, pp. 2086–2095. doi:10.18653/v1/2021.findings-acl.183. https://aclanthology.org/2021.findings-acl.183.

[36] M. Arjovsky, S. Chintala and L. Bottou, Wasserstein Generative Adversarial Networks, in: *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y.W. Teh, eds, Proceedings of Machine Learning Research, Vol. 70, PMLR, 2017, pp. 214–223. https://proceedings.mlr.press/v70/arjovsky17a.html.

[37] O. Press, A. Bar, B. Bogin, J. Berant and L. Wolf, Language Generation with Recurrent Generative Adversarial Networks without Pre-training, *arXiv preprint arXiv:1706.01399* (2017).

[38] T. Shin, Y. Razeghi, R.L. Logan IV, E. Wallace and S. Singh, Autoprompt: Eliciting knowledge from language models with automatically generated prompts, *arXiv preprint arXiv:2010.15980* (2020).

[39] X. Chen, X. Xie, N. Zhang, J. Yan, S. Deng, C. Tan, F. Huang, L. Si and H. Chen, AdaPrompt: Adaptive Prompt-based Finetuning for Relation Extraction, *CoRR* **abs/2104.07650** (2021). https://arxiv.org/abs/2104.07650.

[40] N.S. Keskar, B. McCann, L.R. Varshney, C. Xiong and R. Socher, Ctrl: A conditional transformer language model for controllable generation, *arXiv preprint arXiv:1909.05858* (2019).

[41] Z.-Y. Dou, P. Liu, H. Hayashi, Z. Jiang and G. Neubig, GSum: A General Framework for Guided Neural Abstractive Summarization, in: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, Online, 2021, pp. 4830–4842. doi:10.18653/v1/2021.naacl-main.384. https://aclanthology.org/2021.naacl-main.384.

[42] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi and Y. Choi, HellaSwag: Can a Machine Really Finish Your Sentence?, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[43] Y.W. Lisa Bauer* and M. Bansal, Commonsense for Generative Multi-Hop Question Answering Tasks, in: *Proceedings of the Empirical Methods in Natural Language Processing*, 2018.

[44] C. Havasi, R. Speer, J. Pustejovsky and H. Lieberman, Digital Intuition: Applying Common Sense Using Dimensionality Reduction, *IEEE Intelligent Systems* **24**(4) (2009), 24–35. doi:10.1109/MIS.2009.72.

[45] D.B. Lenat, R.V. Guha, K. Pittman, D. Pratt and M. Shepherd, Cyc: Toward Programs with Common Sense, *Commun. ACM* **33**(8) (1990), 30–49–. doi:10.1145/79173.79176.

[46] P. West, C. Bhagavatula, J. Hessel, J.D. Hwang, L. Jiang, R.L. Bras, X. Lu, S. Welleck and Y. Choi, Symbolic Knowledge Distillation: from General Language Models to Commonsense Models, *arXiv preprint arXiv:2110.07178* (2021).

[47] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, Language Models are Few-Shot Learners, in: *Advances in Neural Information Processing Systems*, Vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan and H. Lin, eds, Curran Associates, Inc., 2020, pp. 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.