Open-Ended Generative Commonsense Question Answering with Knowledge Graph-enhanced Language Models

Final Project Report for CS229, Spring 2021

Hanson Lu
Stanford University
hansonlu@stanford.edu

1 Introduction

Natural Language Processing (NLP) has seen the development of large-scale pre-trained language models, which are trained on vast amounts of language data on unsupervised tasks (called *pre-training*. The model parameters obtained through pre-training are later fine-tuned on specific tasks. Famous examples of such models include BERT (Devlin et al., 2018), and more recently GPT-3 (Brown et al., 2020). Such a pretraining-finetuning paradigm has seen a lot of success in pushing the ability of NLP models to understand language, as we expect the unsupervised pre-training tasks allow the language models to learn of the syntax and semantics of language, as well as commonsense knowledge.

Nevertheless, all of such knowledge is only contained implicitly in the model's parameters. Would it be possible to combine external structured knowledge with the implicit knowledge of these language models?

We would like to explore possibilities to build a model that combines the power of large-scale pre-trained language models with information from commonsense knowledge graphs (KG) such as ConceptNet (Liu and Singh, 2004; Speer et al., 2016). A knowledge graph consists of nodes that correspond to real-world concepts, and are connected by directed edges that represent the relation between two concepts.

We focus on *open-ended question answering*, which is a language question and answer task that requires commonsense knowledge about the world. Specifically, we focus on a *generative* type of this task, where the answers must be spontaneously generated by a model, as opposed to a multi-choice question. We work on a dataset called ProtoQA (Boratko et al., 2020). This dataset contains questions regarding "prototypical situations", such as *"Name something that people usually do before they leave the house for work?"*, and provide a diverse range of ground-truth answers to train an evaluate supervised QA models.

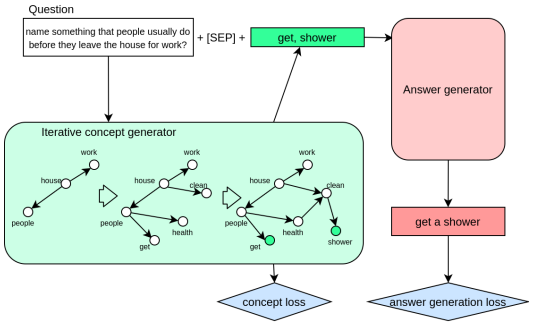

Figure 1: Planned Architecture of our model.

Existing models for the ProtoQA dataset[1] are all fine-tuned versions of generative large-scale language models such as GPT-2 (Radford et al., 2019), BART (Lewis et al., 2019) and T5 (Raffel et al., 2019), none of which explicitly incorporate information from KG.

The closest work that utilizes commonsense KG information in generative QA is Ji et al. (2020a). We followed their work and implemented a model that contains the following components, shown in Figure 1: 1) a *Concept Retriever*, a module that searches for KG concepts that consist the answer, and 2) an *Answer Generator*, which takes the question and KG concepts found by 1) as inputs, and generates an answer.

Unfortunately, we found out that this architecture yielded poor performance for the task. We identified one crucial problem: the answer generator was often distracted by noisy and wrong concepts provided by the concept retriever. We conduct a detailed analysis of whether we can make the answer generator more robust to noisy inputs.

[1]Its current leaderboard can be accessed here https://leaderboard.allenai.org/protoqa/submissions/public

The code for this project can be accessed here[2].

2 Previous Work

Text generation with commonsense KGs. The closest work that utilizes commonsense KG information in generative QA is Ji et al. (2020a). They focused on commonsense explanation generation, where a statement related to daily life was given, e.g. "*the school was open for summer*", and the task was to generate an explanation of why it did not make sense, e.g. "*Summertime is typically vacation time for school*". They developed a *bridge concept extractor* that encodes and scores triples in the knowledge graph to find a path to concepts in the explanation. They then append the retrieved concepts to the statement, and feed that into a pretrained LM to generate the explanation. The bridge concept extractor and the LM are trained jointly. We would like to use their model as a baseline for our task.

Other text generation tasks that incorporate knowledge from commonsense KGs include Ji et al. (2020b), generated story endings given a story context using a gate control method to output text from either a distribution over concepts or conventional neural text decoder. Guan et al. (2019) work on a similar task and encode each context sentence incrementally while attending to both textual and KG node representations, to generate the story ending. These works differ from our project in that they require their generated output is a full sentence as a story ending while ProtoQA is essentially an open-ended QA task that requires a diverse range of short answers.

Open-ended KG-based QA. For this type the question is provided in text form but the answer is more open ended and needs to be search within the entire KG. Datasets include *ComplexWebQuestions* (Talmor and Berant, 2018) and *MetaQA* (Zhang et al., 2018). One significant challenge of this is narrowing down the search space among a large number of nodes. Sun et al. (2020) derived an iterative procedure that incrementally traces the graph and adds new graph entities to the search space. This line of work is more similar to our task and we primarily draw inspiration from these methods for ProtoQA.

[2] https://github.com/hansonhl/GenerativeCommonsenseQA

3 Methods

There has been similar work on text generation and question answering tasks related to commonsense reasoning and knowledge graphs.

3.1 Task formulation

We are given a question statement $\mathbf{q}^{(i)} = q_1...q_N$ with N words, with a set of possible answers $\mathcal{A}^{(i)} = \{\mathbf{a}_1, ..., \mathbf{a}_k\}$ where each answer is composed of a string of words.

We define a knowledge graph $G = (V, E, \mathcal{R})$ where V is the set of concepts, $\mathcal{R}$ is the set of all possible relations between concepts, and E contains edges that are in the form of *triples* (h, r, t) where $h, t \in V, r \in \mathcal{R}$.

For each question $\mathbf{q}$ and answer $\mathbf{a}$, we find the concepts in the KG that can be found in the question and answer. For example, the question *"Name something that people usually do before they leave the house for work?"* may contain concepts `people`, `house`, `work`, `leave_house`, and the answer *"get a shower"* may have concepts `get`, `shower`. Use $\mathcal{C}_\mathbf{q}$ and $\mathcal{C}_\mathbf{a}$ to denote the sets of concepts in the question and answer respectively.

3.2 The Concept Retriever

The concept retriever traverses the graph iteratively, and ultimately outputs a score for each concept in a subset of concepts $\tilde{\mathcal{C}}$ in the entire graph. The score indicates how likely each $c \in \tilde{\mathcal{C}}$ is in the set of answer concepts $\mathcal{C}_\mathbf{a}$. Specifically,

$$P(c \in \mathcal{C}_\mathbf{a} \mid q, \mathcal{C}_\mathbf{q}) = \sigma(s_{answer}(c)) \qquad (1)$$

for each $c \in \tilde{\mathcal{C}} \subset (V - \mathcal{C}_\mathbf{q})$. σ denotes the sigmoid function. We select the top-K_1 concepts with the highest $s_{concept}$, which we denote as $\hat{\mathcal{C}}$ and treat those as the final predicted answer concepts of the Concept Retriever.

We only score a subset of all the nodes because it is infeasible to encode and score all the nodes in the graph. That subset $\tilde{\mathcal{C}}$ is a set of candidate nodes that are obtained as we traverse the graph. The traversal process is as follows.

Let $\mathcal{C}^{(t)}_{curr}$ be our current set of nodes at iteration t, and we initialize that to $\mathcal{C}^{(0)}_{curr} \leftarrow \mathcal{C}_\mathbf{q}$. At iteration t, we consider all the nodes that are connected to at least one node in $\mathcal{C}^{(t)}_{curr}$ and that we haven't visited

before. Denote this set of nodes as $\tilde{\mathcal{C}}_{curr}^{(t+1)}$. Then

$$\tilde{\mathcal{C}}_{curr}^{(t+1)} = \{t \in V - (\mathcal{C}_{curr}^{(0)} \cup ... \cup \mathcal{C}_{curr}^{(t)}) : \exists (h, r, t),\ h \in \mathcal{C}_{curr}^{(t)}\} \quad (2)$$

For each $c \in \tilde{\mathcal{C}}_{curr}^{(t+1)}$, we give a *routing score* $s_{routing}$ that indicates how likely it lies on any possible *shortest path* in the KG from the $\mathcal{C}_{\mathbf{q}}$ to $\mathcal{C}_{\mathbf{a}}$. To compute $s_{routing}(c)$, we individually score each *edge* on every path from any node in $\mathcal{C}_{\mathbf{q}}$ to c, and then average these scores over all edges. $\tilde{\mathcal{C}}$ is taken to be the top-K_2 concepts that have the highest routing scores.

All of the scoring functions mentioned above are based on transformations of GPT-2 embeddings of the concepts and the question. We refer the reader/grader to Ji et al. (2020a)'s original paper for the model details.

The training objective for $s_{concept}$ is a binary cross entropy loss to classify whether c belongs in the true set $\mathcal{C}_{\mathbf{a}}$.

$$\mathcal{L}_{concept} = -\sum_{c \in \tilde{\mathcal{C}}} \mathbb{1}[c \in \mathcal{C}_{\mathbf{a}}] \log \sigma(s_{answer}(c)) + (1 - \mathbb{1}[c \in \mathcal{C}_{\mathbf{a}}]) \log(1 - \sigma(s_{answer}(c))) \quad (3)$$

Note that $s_{routing}$ is ultimately derived from edge scores s_{edge}. We train $s_{edge}(h, r, t)$ on the binary classification task of predicting if the edge $e = (h, r, t)$ lies on any shortest path from $\mathcal{C}_{\mathbf{q}}$ to $\mathcal{C}_{\mathbf{a}}$:

$$\mathcal{L}_{edge} = -\sum_{e \in \tilde{\mathcal{E}}} \mathbb{1}[e \in \mathcal{E}_{\mathbf{q} \to \mathbf{a}}] \log \sigma(s_{edge}(e)) + (1 - \mathbb{1}[e \in \mathcal{E}_{\mathbf{q} \to \mathbf{a}}]) \log(1 - \sigma(s_{edge}(e))) \quad (4)$$

Where $\tilde{\mathcal{E}}$ is the set of all edges that we encountered during the iteration process, and $\mathcal{E}_{\mathbf{q} \to \mathbf{a}}$ is the set of all edges on a shortest path from $\mathcal{C}_{\mathbf{q}}$ to $\mathcal{C}_{\mathbf{a}}$.

3.3 The answer generator

The answer generator is essentially a GPT-2 model which predicts the final answer given the question tokens and the predicted answer concepts $\hat{\mathcal{C}}$ from the concept retriever. We concatenate the text form of all the answer concepts $\hat{\mathcal{C}}$ to the question, and use that as the context $\mathbf{c}$ for GPT-2 to generate the answer $\mathbf{a}$:

$$P(\mathbf{a} \mid \mathbf{c}) = P(\mathbf{a} \mid \mathbf{q}, \hat{\mathcal{C}}) = \prod_{t=1}^{M} P(a_t \mid \mathbf{a}_{<t}, \mathbf{c}) \quad (5)$$

In particular, the input to GPT-2 will be formatted in the following form:

`question` [SEP] `concepts` ...[SEP] `answer` (6)

where $\mathbf{c}$ consists of the question and concepts. To ensure permutation invariance between multiple concept tokens, each concept will start with the same position embedding.

The answer generation loss is computed as the negative log loss of the above probability: $\mathcal{L}_{generation} = -\log P(\mathbf{a} \mid \mathbf{c})$.

The training objective of the entire model is the sum of the above three losses: $\mathcal{L}_{final} = \mathcal{L}_{concept} + \mathcal{L}_{edge} + \mathcal{L}_{generation}$.

4 Experiments

4.1 Official metrics

Using the above setup, we trained the entire model and evaluated it using the official evaluation metrics from ProtoQA (Boratko et al., 2020). For each question, we provide a ranked list of answers to an evaluator, and the evaluator matches the content as well as the ranking of these answers. For our experiments, we focus on "WordNet" matching, which matches our predictions to the true answers using a fuzzy similarity function instead of an exact match. "Max answers n" only looks at the top n answers in the predictions, while "Max incorrect m" goes down the ranked list and considers every answer until m incorrect answers are encountered.

We used varying values of K_1 and and K_2. In these tests, we do not let the concept extractor expand the graph by itself, but force it to start at the ground truth intermediate concepts as $\mathcal{C}_{curr}^{(t)}$ for each step t, which can be understood as a type of teacher forcing.

Unfortunately, even under this condition, our initial experiments show that the results are much worse than if we simply trained a GPT-2 model on question-answer pairs without any information from KG. The results are shown in Table 1.

When we inspected some examples, as shown in table 2, we see that the concept retriever often retrieves irrelevant concepts. The answer generator is confused by them and generates the incorrect answers. It is necessary for the answer generator GPT-2 model to be robust enough to ignore wrong or irrelevant concepts generated from the concept retriever, because the number of ground-truth concepts is often smaller than K_1, which is the size of the set of retrieved answers.

Model type	K_1	K_2	Max answers 10	Max incorrect 3	Max incorrect 5
GPT-2 with gold $\mathcal{C}_\mathbf{a}$			*74.19*	*92.35*	*96.79*
GPT-2 with only $(\mathbf{q}, \mathbf{a})$ pairs			*70.86*	*74.06*	*86.07*
Our model	10	25	18.56	3.99	8.52
	5	25	11.87	2.90	9.08
	8	16	12.48	5.55	6.78
	5	10	7.03	4.84	6.31

Table 1: Main experiment results with the concept retriever-answer generator architecture

Ground truth answer	Gold truth concepts	GPT-2 output	Retrieved concepts from concept retriever
birthday date	`birthday, date`	candy	`move, land, cake, candy, flu`
their age	`age`	boss	`boss, age, coffee, science, drink`
zodiac sign	`zodiac_sign, zodiac, sign`	cooking	`run, cooked, student, computer_science, driver`
birthday	`birthday`	cake	`pregnant, cake, clothes, weeds, tough`

Table 2: Examples of outputs from the concept generator for the question *Name something that is hard to guess about a person you are just meeting.* The left two columns are independent from the right two ones.

5 Analysis

Training the answer generator on examples with noisy concepts To test the robustness of GPT-2 to incorrect answers from the concept retriever, we finetuned GPT-2 models on a similar task as our answer generator (see (6) for the input formatting), but for the concepts that are presented to it, we mix the ground-truth answers concepts $\mathcal{C}_\mathbf{a}$ with a set of other concepts. For this additional set of concepts, we tried: 1) putting together all the ground-truth concepts for *every* answer, and training the model to predict one answer; 2) a random sample of neighbors of $\mathcal{C}_\mathbf{a}$ in the KG; and 3) a random sample of concepts that appear on the a shortest path from $\mathcal{C}_\mathbf{q}$ to $\mathcal{C}_\mathbf{a}$. We also vary the sizes of these random samples.

We show the results in Table 3. We see that GPT-2 is not robust enough to generate the correct answer when given a noisy set of concepts, even though the ground-truth concepts is in that set. We see decreasing performance as the set of noisy concepts increases, and all have lower scores than GPT-2 trained on only $(\mathbf{q}, \mathbf{a})$ pairs.

Experimenting with different input formats The original GPT-2 paper (Radford et al., 2019) emphasizes the ability of GPT-2 to utilize its understanding of linguistic cues to complete tasks it is fine-tuned on. An example would be appending the phrase "*Summarize this paragraph:*" before the text of a summarization task. We use a similar trick for our task, where we add hinting in the form of natural language for the input:

`Question` : <*question tokens*>
`possible answers are` : <*concept tokens*>
`Answer` : <*answer tokens*>

We also turned off permutation invariance of the answer concepts' positional embeddings as an ablation experiment. The results are shown in Table 4. We see that hinting does not provide a significant advantage over the original way of using the [SEP] token to separate the question and answer. We also see that the permutation invariance setting is crucial when we do not use hinting. Interestingly, we also see that with natural language hinting, ablating permutation invariant position embeddings actually does not harm the model's performance to a great extent at all. This may mean that GPT-2 may have implicit knowledge on how to reason with situations such as sets and collections given the correct natural language context.

Curriculum learning In the above setting we only trained the answer generator on only examples with noisy concepts mixed with the correct concept. We found that such a task might be difficult since the model has to figure out itself from the beginning which concepts are the correct answer. Can we provide a more explicit signal by first training it to produce the correct answer given the ground truth concept, and then gradually introducing examples where the correct concept is mixed together with noisy ones? We experiment with *curriculum learning*(Bengio et al., 2009), where we

Type of noisy concepts	# noisy concepts	Max answers 10	Max incorrect 3	Max incorrect 5
GPT-2 with gold $\mathcal{C}_{\mathbf{a}}$		*74.19*	*92.35*	*96.79*
GPT-2 with only $(\mathbf{q}, \mathbf{a})$ pairs		*70.86*	*74.06*	*86.07*
All answer concepts	2	64.98	72.57	85.17
	5	62.43	68.74	76.48
	10	60.65	62.91	73.51
Path concepts	2	70.31	71.07	85.55
	5	69.60	64.80	79.69

Table 3: Answer generator robustness experiments results

Type of noisy concepts	Formatting	Permutation invariance	Max answers 10	Max incorrect 3	Max incorrect 5
GPT-2 with gold $\mathcal{C}_{\mathbf{a}}$			*74.19*	*92.35*	*96.79*
GPT-2 with only $(\mathbf{q}, \mathbf{a})$ pairs			*70.86*	*74.06*	*86.07*
Path concepts	SEP	Yes	70.31	71.07	85.55
	SEP	No	66.65	63.94	78.09
	hinting	Yes	68.73	69.10	83.59
	hinting	No	68.34	71.00	81.93

Table 4: Experimenting with input formatting and ablating permutation invariance

# eval noisy concepts	Curriculum learning setting	Max answers 10	Max incorrect 3	Max incorrect 5
GPT-2 with gold $\mathcal{C}_{\mathbf{a}}$		*74.19*	*92.35*	*96.79*
GPT-2 with only $(\mathbf{q}, \mathbf{a})$ pairs		*70.86*	*74.06*	*86.07*
2	No curriculum learning	70.31	71.07	85.55
	0 warm-up subepochs	71.73	73.17	85.74
	20 warm-up subepochs	**72.89**	**77.76**	**85.95**
	60 warm-up subepochs	69.17	68.59	81.23
5	No curriculum learning	69.60	64.80	79.69
	0 warm-up epochs	**70.58**	**70.83**	**83.20**
	20 warm-up epochs	68.73	66.03	80.07
	60 warm-up epochs	68.37	68.68	79.38
	Gradual increase # noisy examples	70.06	66.90	79.25

Table 5: Experiment results with curriculum training

gradually increase the ratio of noisy examples to non-noisy examples during training.

Specifically, we divide each training epoch into equally-sized *subepochs*, and after a certain number of "warm-up" subepochs, where the model only sees examples with ground truth concepts, the ratio of noisy examples to non-noisy ones in each subepoch is linearly increased until a fixed ratio of 1.0. We implemented a dataloader that takes batches from two separate datasets, one for non-noisy examples and the other for noisy ones, and shuffles those batches for each subepoch. In effect all examples in each batch that the model is trained on is of the same type – either noisy or non-noisy.

We show our results in Table 5. We see that curriculum learning in some cases indeed pushes the model's performance above the baseline with no concept information at all, in the setting with two noisy examples and 20 warm-up subepochs. We find that the timing of when the noisy examples are introduced is crucial for the model to learn well.

6 Conclusion and next steps

In conclusion, though our proposed concept retriever-answer generator two-step architecture does not perform well on the task. We identified one main problem being that the answer generator was not robust to noisy and incorrect concepts, and investigated ways of improving that robustness. We found that curriculum learning may be an effective way, and the next step is to incorporate these techniques together with the concept generator.

From the above results we can hypothesize that if we were to continue using our current setup for the answer generator, we must provide it with a small set of answer concepts with *high precision*. For this we have to think of ways to improve our concept retriever's performance. Possible directions include selecting a better set of paths to train the concept generator. Currently we use all possible shortest paths between the question and the answer in a KG, but many of these paths are often irrelevant to solving the problem.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Michael Boratko, Xiang Li, Tim O'Gorman, Rajarshi Das, Dan Le, and Andrew McCallum. 2020. ProtoQA: A Question Answering Dataset for Prototypical Common-Sense Reasoning. pages 1122–1136.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Jian Guan, Yansen Wang, and Minlie Huang. 2019. Story ending generation with incremental encoding and commonsense knowledge. *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pages 6473–6480.

Haozhe Ji, Pei Ke, Shaohan Huang, Furu Wei, and Minlie Huang. 2020a. Generating commonsense explanation by extracting bridge concepts from reasoning paths. *arXiv*.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2020b. A survey on knowledge graphs: Representation, acquisition and applications. *arXiv*, pages 1–26.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv*.

Hugo Liu and Push Singh. 2004. ConceptNet - a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226.

A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. pages 1–53.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2016. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. (Singh 2002).

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2020. PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, pages 2380–2390.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. *arXiv*, pages 641–651.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 6069–6076.