CIS 700 HW2: Generating Descriptions Report

Weiqiu You, Yifei Li

February 1, 2022

1 Experiment

Fine-Tuning We first fine-tune the five kinds of models using light environment data. All of them are processed by Babbage model, since it's cheap yet has a goodish performance. A bad example is Ada, which empirically did a bad job of generation in this assignment.

1. Describe a location: we basically use the given inputs, i.e. category and location name, and then output the location description. Most of the generated strings make sense.

2. List the items that are at a location: we feed the GPT-3 the category, location name, location description, number of items, and let it generate a list of item names. We notice the items are generally derived from the description. There might be some unnecessary repeating terms, such as "trees", a plural, repeating twice in a list. When we use this function, we randomly sample an integer between 0 and 19 (inclusive, as this is the range for the training data) and generative this number of items for a given location.

3. Describe an item: Given category, location name, location description, item name, output item description. The workflow of this task is basically the same as the fist task. The performance is good.

4. List connections from the current location: Given category, location name and description, and a list of existing locations, output a list of connections. We faced some obstacles. First, it's impossible to pass model the full list of locations given the token limitation. Second, given that the connections are generated randomly, it might lead to an isolated island or deadlocked node. Last, it's not guaranteed that a language model can generate the connections among locations that are well-defined on topology because it doesn't have and cannot understand the whole map. While there's no easy way to tackle the last issue, we partially solve the first two by the implementation of Breath First Search (BFS) and reverse connection. Specifically, we use BFS to catch the nearest locations within a depth of ten, then feed all of them to the model, hoping that some connections can be generated automatically among them. Then we prompt GPT-3 Babbage to generate the reverse connection, e.g. given "east", return "west", concatenated to the name of the upstream node, to somehow ensure the network connectivity and completeness.

5. Get an item's properties: Given item name and description, property (e.g. gettable), the model should determine it's True or False if the item has that property. The only tricky thing here is in the training corpus, the boolean value is expressed in float-string that can be any value between 0 and 1, therefore the threshold between True and False should be set as 0.5. Also, during model fine-tuning, the boolean value is better to be denoted as string ("True"/"False") rather than float-string, as the former is less ambiguous in token-expression. The model's attribute is showed in the Table 1. The model is relatively less confident on whether the item is drinkable or editable [1]. Decreasing the temperature of GPT-3, i.e. making the model more consistent and conservative, didn't make an obvious difference in the prediction of item's property. Additionally, it would be quite interesting to extract the GPT-3 embeddings of the item's description and name and then feed the statistical machine learning models such as K-Nearest Neighbors to predict the property, but we didn't have time to explore that.

[1] Also check the section 5.4-5.6 of https://www.cis.upenn.edu/~ccb/publications/masters-theses/Anna-Orosz-masters-thesis-2021.pdf

Property	Precision			Recall		
	$T = 0.7$	$T = 0.1$	Δ	$T = 0.7$	$T = 0.1$	Δ
is_gettable	0.89	0.89	0.0	0.82	0.84	0.02
is_weapon	0.85	0.85	0.0	0.53	0.53	0.0
is_surface	0.74	0.72	-0.01	0.48	0.50	0.02
is_container	0.69	0.68	-0.01	0.71	0.68	-0.03
is_wearable	0.88	0.88	0.01	0.64	0.68	0.05
is_drink	0.67	0.67	0.0	0.36	0.36	0.0
is_food	0.75	0.80	0.05	0.40	0.27	-0.13
AVERAGE	0.78	0.78	0.00	0.56	0.55	-0.01

Table 1: The precision and recall of predicting each item's property given the prompts via GPT-3 Babbage. Note that T indicates the temperature of the language model, and Δ indicates the metric difference between temperature 0.1 and 0.7.

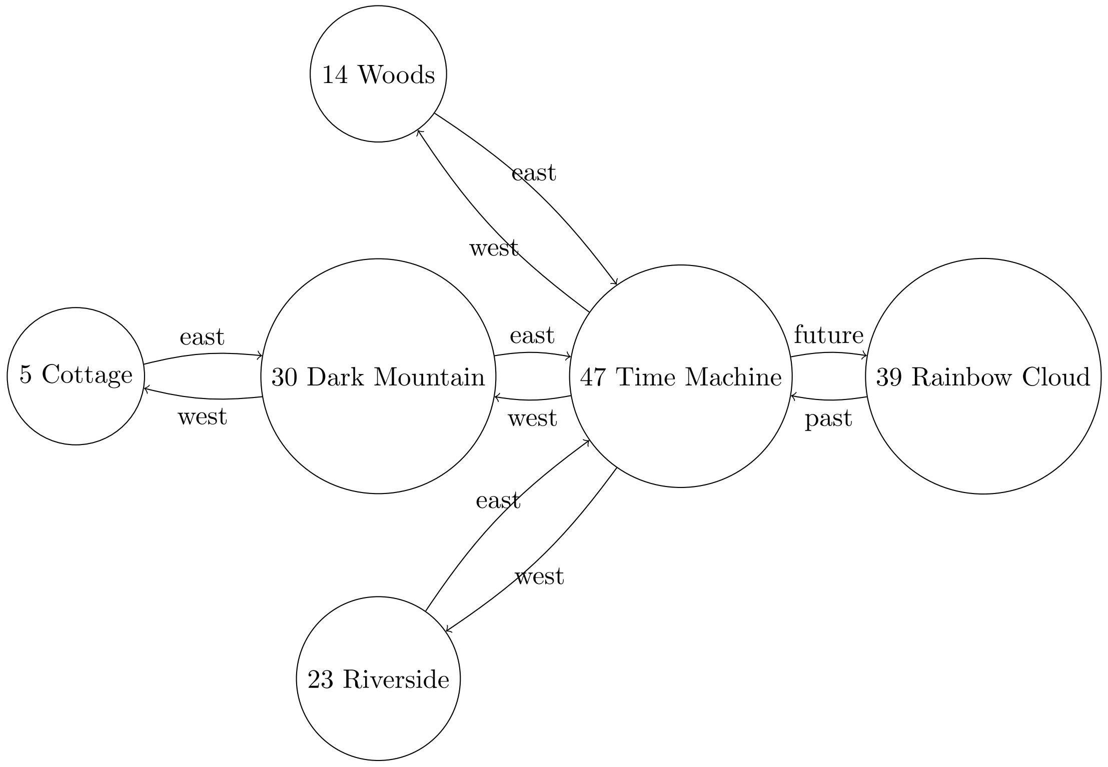
Game Generation The generated game is kind of convincing. Here are some intersting findings we found in the game generated.

1. One thing we notice for the generated game is that the category influences the content a lot. When we try to generate the description for "Cottage" (Figure 2, we see it's darker with the Dark Forest theme, but more lighthearted with colors of red and green when using the Magical Realm theme. Also for the Dark Forest theme, the description is talking about the forest instead of cottage, which is a little off-topic.

Dark Forest	The dark forest is a gloomy place. There are few lights and many shadows. The forest is very dense and hard to see because of the many trees.
Magical Realm	The magical realm is a cottage made of red brick. It is surrounded by a ring of magic that prevents evil from entering. A green candle lights its interior.

Table 2: Description of Cottage

2. The connections it generated are interesting and topologically non-conflicting. Very interestingly, for the Time Machine Location, the finetuned GPT-3 Babbage model generated a connection from Rainbow Cloud to Time Machine with direction "past", and we later have a prompted GPT-3 Babbage model generate the opposite connection from Time Machine to Rainbow Cloud to be "future". The model does understand some interesting properties, even directions in time.



2 Conclusion

Result By fine-tuning GPT-3 and making it generate the prompts continually and self-triggeringly along the chain, we empirically prove that a simple text game can be generated given only a category and a short list of initial location names. Despite it's far from perfect, it reveals the potentials that the text adventure games can be fully generated by AI.

Future Work However, due to the lack of logic and big picture, it's expected that the larger the game, the less accurate the game generated by GPT-3. We might resort to the knowledge graph to memorize and create some prior contents to the machine, empowering the language model to generate a more coherent and logical representations. Right now the model does not have a sense of topology when generating the graph, so there can be errors when connecting different locations. This is also an interesting direction to investigate when incorporating knowledge graphs.

Also, to make the generation even more automatic, we can finetune another model to generate location names from the theme. Then we only need a theme name to generate a game. The current setting, on the other hand, allows us to have more control over what locations should be present in the game. It is more interesting if a game designer has some specific locations in mind. We can also allow humans to edit the AI generated game and use them as inspirations rather than constraints. The editing data can also be used to train the model to learn a game designer's personal style.

Finally, we can also finetune models to generate NPCs and allow players to engage in dialogues with them. This would be more complicated if we want the dialogues to have a purpose and be coherent, probably needing a system for goal oriented dialogues.

The tricks in lots of NLP domains, such as question answering, common sense reasoning and robustness, can jointly contribute to improve the quality of the text game generation.