Nobody Writes Docstring: Game Item Description Generation with Pretrained Language Models

Ning Wan, Hongyu Zhang, Yuxuan Wang

April 4, 2022

1 Project Description

Nobody wants to write docstring. What if we can build a better description generator? Recall the games you played; the item descriptions contains information particular to the game settings or the game-world settings For example, you know the item is from *Dark Soul* or *Don't Starve* simply by reading the text: even when they are not! Therefore, our work is more developer-oriented, particularly convenient for those who want to develop mods for their favorite games and conform to a consistent world setting or style of writing. So, in this project, we would like to revisit Homework Two, the description generation. We want a generator that takes an item title as input and generates a corresponding "gamified" item description. However, instead of building an end-to-end system as we have done in Homework Two, we observe that such a task is two-fold:

- Generate a basic description for a simplified item name that depicts the main properties of this item. For instance, a sword is made of metal, a pointy weapon.
- Render the utterance to sound like a game item when we condition it on the game category and full item name like *King Arthur's Sword*.

To sum, we want to formulate this task as a combination of description generation and conditioned style transfer problem.

2 Related Works

Our project leverages the recent advancement of large **Pre-trained Language Models** (PLM), e.g., GPT-3, T5, and BART [1, 7, 5]. Further, it is shown that such PMLs can serve as knowledge bases, especially salient for simple factual knowledge [3, 6]. This inspires us of using large PLMs as base description generator.

Our formulation of game item description generation resembles the most to the **Text Style Transfer** (TST) task with some variation. TST aims to control certain attributes in the generated text, such as politeness, emotion, humor, and many others [4]. It involves a source style and a target style. In our case, the source style is the simple item description, where as the target style is the gamified description. Specifically, given the base description, we can construct a parallel dataset where standard sequence-to-sequence models can be applied [8].

However, we need to keep in mind that the renderer functions not only as a style transferer but also as a domain-specific knowledge base. For example, to generate fantasy-game-related item descriptions, a renderer needs to have prior knowledge about terms like Mana, Goblin, or Holy Grail. And we do not expect an out-of-the-box model has such knowledge pre-encoded [2]. Therefore, the specific methods to solve this problem are under discussion.

3 Methods

3.1 Proposed Architecture

According to our observation, we can decompose the game-item description generation as a two-phase process: base form description generation and rendering. The two models are collated together to form a game item descriptor.

3.1.1 Base Generator

First of all, almost every game item has a ”base form.” Such base forms can be weapons, garments, or food. The base form of a game item reveals its essential properties. For instance, *Excalibur* is still a sword, a pointy weapon (possibly) made of a piece of metal. Therefore, intuitively, to generate a description for a game item, we need a particular mechanism to help create a piece of text for its base form in the first place. Such a machine needs to possess common sense knowledge about the properties of various base forms, which naturally leads to large PLMs or models trained upon knowledge bases. Formally, a generator is a function with the signature,

```python
def generator(base_form: str) -> str:
    """The generator model API. It should take a single
        item like 'sword' and generate some common-sense-based
        description for that item, e.g. 'a pointy piece of metal'.
        The <model> here is an abstraction, it can be a large PLM
        like GPT-3 or specialized knowledge-base model like COMET-BART"""

    prompt = make_prompt_gen(base_form)
    base_desc = model_gen(prompt)
    return post_proc_gen(base_desc)
```

3.1.2 Game Renderer

Next, we believe that two extra pieces of information contribute to the ”game flavor” of the description. Firstly, where do these items appear? For instance, a cup in a dungeon may associate with adjectives like ”filthy” or ”rusty,” while a mug in a kitchen can be ”clean” or ”cute.” Secondly, the full name of that item, e.g., *King Arthur Sword* instead of *Sword*. We know that this particular weapon is from *King Arthur's Legend*. So the corresponding descriptions should relate to terms like *Avalon*, *Knights of the Round Table*, or *Holy Grail*. We can see that it also requires the renderer to have another set of common sense knowledge. However, such knowledge is more game-specific or world-specific: it is not likely to have a machine gun in greek mythology. In other words, the renderer serves as an (inverse) adapter, and the engineering advantages brought by the disentanglement are apparent, i.e., we do not need to re-train the often-humongous base generator when switching to a different set of settings or game.

```
def renderer(setting: str, full_name: str, base_desc: str) -> str:
    """The renderer model API. It rephrase/diversify the item
        description conditioning on the <setting>, <full_name>,
        and pre-computed <base_desc>"""

    prompt = make_prompt_ren(setting, full_name, base_desc)
    game_desc = model_ren(prompt)
    return post_proc_ren(game_desc)
```

3.1.3 Item Base Form Extractor

Another small but important piece of building block is the item base form extractor. Ideally, a user of our proposed system only needs to specify the full name of a game item, but a base generator only takes the **base form** as its input. Thus, we need a dedicated algorithm to extract/convert the game item to its base form. We currently explored two approaches. The first method is based on KeyBERT, which tries to find a subset of the tokens that best preserves the semantic information, measured by **cosine-similarity** between full name contextualized embedding and those of the sub-tokens'. However, the base form may not necessarily contained in the full name. So, another method we proposed is to, again, rely on large PLMs and text-generation. We experimented with generation using COMET-BART via *MadeOf* relation, and prompting-by-example with GPT-3. The GPT-3 model gives the best performance.

3.2 Model Development Procedure

We introduce both planned development procedure as well as preliminary experiments in this section.

3.2.1 Generator Construction

Preliminary Experiments As discussed above, the base description generation relies on a large PLM that can serve as a knowledge base. Therefore, we develop our baseline model based on **GPT-3** via **few-shots prompting**. We construct the prompt by concatenating ten `"item: <ITEM>, description: <DESC>"` pairs. An example is shown below,

```
prompt = f"""
item: Sunglasses, description: All of our sung...
item: Scarf, description: Feature -Great quality...
                    ...
item: Shoe, description: Kick back and relax...
item: {NEW_ITEM}, description:
"""
```

The item descriptions we used is acquired from the Amazon Review Dataset, which is discussed in §4. This simple method works surprisingly well, which is discussed in §6.

Planned Procedures Although the preliminary result is satisfactory, prompting `davinci` model is very costly. Thus, our next step is to collect sufficient amount of examples, roughly a hundred, and fine-tune a smaller model such as `ada` and `babadge`. Further, we have not yet fully experimented various crucial sampling hyperparameter such as the temperature and penalties.

3.2.2 Renderer Construction

Preliminary Experiments Again, we use **GPT-3 prompting** for the proof-of-concept purpose. An example prompt is shown below,

```
prompt = f"""
item: Wood table category: Tavern base_desc: The table is made of wood ...
                               desc: The table is old and dusty... ===
                    ...
item: {FULL_NAME} category: {SETTING} base_desc: {BASE_DESC} desc:
"""
```

The base description is generated from pre-constructed base-desc-generator, and target description is taken from the LIGHT dataset, discussed in §4.

Planned Procedures Since the prompting performance is under expectation, we plan to perform error analysis first and discuss what information is not captured by the model, or, if we need to fine-tune the model on full LIGHT dataset or other augmented datasets.

4 Data

4.1 Amazon Review (Meta) Data

This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning from May 1996 to July 2014. This dataset includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs) [1]. For this project, **we only use the metadata** which contains **item names** and corresponding **short descriptions**. But, we noticed that the dataset is noisy (for our purpose). For instance, it may include advertising text that is depicting the properties of the (base) item. Thus, we manually select a tiny subset of descriptions that are more relevant. Further, we plan to augment the dataset with Wikipedia descriptions.

4.2 LIGHT

We briefly discuss the LIGHT dataset since we frequently work with this dataset throughout this course. At this stage, we only use the objects dataset. For each one of the game items, there are `'name'`, `'in_room_ids'`, `'base_form'`, and `'descriptions'`. To formulate a style transfer task, we first apply the pre-constructed base generator to `'base_form'` to make a pseudo-parallel dataset. Next, for the rooms an item exists in, we use the `'category'` of the rooms to control the style transfer (use as `'setting'`). Finally, we have the required information for the renderer.

5 Evaluation Plan

The most important component of our system is the renderer. We primarily focus on evaluating lexical overlapping and semantic preservation. We use the **BLEU Score** and **ROUGE Score** to measure the lexical overlap. The reason for including the ROUGE score is to assess the recalled properties of gamified description. To evaluate the degree of semantic preservation, we plan to use SentenceBERT to compute the semantic similarity between the generated text and the ground truth. The perfect result would be the two pieces of texts being almost paraphrases of each other. We discard **Perplexity** evaluation because the game item descriptions may naturally be very complex because the ”uncommon words,” i.e., game-specific words, are very common.

Although our main focus is to evaluate the renderer, we also understand that the base descriptions are strong baseline. Thus, we will compare the generation quality against the ”raw descriptions.”

[1]https://jmcauley.ucsd.edu/data/amazon/

6 Preliminary Results

6.1 Base Generator

At the point of writing this milestone report, we have constructed baseline models for both of the generator and renderer based on GPT-3 prompting. The base generator achieves satisfactory generation quality, examples are shown below.

- **Knife**: *The knife is a tool that has been used by humans for centuries. It is a versatile tool that can be used for a variety of purposes, including cutting, slicing, and carving. A knife is typically composed of a blade and a handle. The blade is the sharpened edge of the knife that is used for cutting, while the handle is used for grip and control. There are a variety of different types of knives, including pocket knives, kitchen knives, and hunting knives.*

- **Necklace**: *This beautiful cowrie shell necklace can fit any neck size! It looks great and is strung on a comfortable and strong nylon cord, which is also waterproof. Imported from Africa.*

One interesting phenomenon is that when we plugin *King Arthur Sword*, the generator gives *"The King Arthur sword is a replica of the sword that was used by King Arthur in the famous legend. The sword is made of high quality stainless steel and has a polished finish. It is also very sharp and can be used for display or cosplay purposes."* It shows that, without rendering, the base descriptor produce items that are derived from day-today life. Even the king's sword only serves as a costume. This proves the necessity of a renderer.

Further, we experiment with COMET-ATOMIC-BART, but the out-of-the-box performance is not as satisfactory. Most generated descriptions are too short, i.e., not informative enough. We hypothesis that a more informative description requires a combination of relations more than just `MadeOf`, which needs heavy engineering, beyond the scope of prototyping.

6.2 Game Renderer

```
>>> baseline("king arthur sword", "magical realm", get_description("sword"))
'The sword is old and rusted, but still sharp. The blade is engraved with the
                                       words "Excalibur".'
>>> baseline("teeth of troll", "dungeon", get_description("teeth"))
'The teeth are sharp and white, and they look like they could belong to a troll.
                                       They are large and sharp, and would be
                                       dangerous if used as a weapon.'
>>> baseline("enchanted bone", "Graveyard", get_description("bone"))
'The bone is old and dry. It is long and hard, and can be used as a weapon. The
                                       bones are old and crumbling, showing a
                                       storied past.'
```

We randomly selected ten objects from the LIGHT dataset to construct the few-shots prompt. Then, we extracted the object name, the category (style) of the room that it was in, and the original object description for each object. The prompt is the object's full name, the room category, and the base description given the base form of the object name using the base generator we constructed above. The completion is the intended game object description.

7 Attribution

- **Ning Wan**, constructs the base description generator baseline as well as item base-form extractor; will focus on improving the quality of the sub-models.
- **Hongyu Zhang**, constructs the renderer baseline; will focus on further improving the rendering quality.
- **Yuxuan Wang** initiates idea and manages the progress of the project; also responsible for the report writing.

References

[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *CoRR abs/2005.14165* (2020).

[2] Gururangan, S., Marasovic, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., and Smith, N. A. Don't stop pretraining: Adapt language models to domains and tasks. *CoRR abs/2004.10964* (2020).

[3] Hwang, J. D., Bhagavatula, C., Bras, R. L., Da, J., Sakaguchi, K., Bosselut, A., and Choi, Y. COMET-ATOMIC 2020: On symbolic and neural commonsense knowledge graphs. *CoRR abs/2010.05953* (2020).

[4] Jin, D., Jin, Z., Hu, Z., Vechtomova, O., and Mihalcea, R. Deep learning for text style transfer: A survey. *CoRR abs/2011.00416* (2020).

[5] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR abs/1910.13461* (2019).

[6] Petroni, F., Rocktäschel, T., Lewis, P. S. H., Bakhtin, A., Wu, Y., Miller, A. H., and Riedel, S. Language models as knowledge bases? *CoRR abs/1909.01066* (2019).

[7] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR abs/1910.10683* (2019).

[8] Rao, S., and Tetreault, J. R. Dear sir or madam, may I introduce the YAFC corpus: Corpus, benchmarks and metrics for formality style transfer. *CoRR abs/1803.06535* (2018).