Docstring to Code: Project Reflection

Ashkon Aghassi, Fred Qin, Cindy Lay

GitHub Repository: https://github.com/cindylay/181yfinal-cinashfred

Initial Vision and Evolution:

Based on classwork in NLP, our initial thought was to expand on our classwork in this topic. Upon more research we saw many viral videos of projects using the famous and powerful GPT-3 pre-trained Neural Network. We read articles about GPT-3 generating news articles and also saw viral videos of GPT-3, that when given a prompt could generate accurate HTML code--whether these were buttons, charts, or tables--ready for deployment. This caused us to wonder if we might also be able use GPT-3 to generate code. And if we were able to do this, would this mean that we were 'coding our own jobs away?' or that software engineering was becoming more obsolete?

We decided to take on the answer to this question! When first attempting to do this, we found that GPT-3 required an extensive research license application. Since it was powerful enough to generate malicious artifacts like fake news articles, it made sense that they did not want just anyone messing around with it, so we set on using GPT-2 instead. We thought back to our CS5 days and remembered the strong emphasis on clear well formatted docstrings--which was also hammered home by CS70. :) Thus, we narrowed our objective to building something that when inputted a docstring, could generate accurate Python code with the help of GPT-2.

Progress:

First we had to train our model; we decided to use our CS5 Homework sets. Since individually formatting each homework would be inefficient, we used our CS181Y knowledge using os.walk, regex, and string processing to build a pre-processing script: `preprocessing.py`. This script, given a file path to our CS5 folder, opened the folder and formatted each homework as training data. We wanted to (a) clean out all the comments and other unnecessary hw jargon, (b) separate docstrings from their place inside functions, and (c) know which docstrings corresponded to each function. This helped us build our training data which you can see an outline of on our presentation slide. A major obstacle we encountered at this stage was parsing for docstrings which are specified by triple-quotes: `"""_docstring_"""`. This was hard to distinguish from our text file which was in itself one a large string, and we ended up searching the regex documentation for a fix that would escape our string properly.

After we formatted our training data we first experimented with Jupyter notebook, but ended up using Google Collab because training the model on our local machines took an exorbitant amount of time.! Although we love free GPUs provided by Google, collab had a steep

learning curve. We relied on online resources to help us train the model. Using tensorflow and gpt-2's 117M pretrained model, we retrained GPT-2 with our carefully formatted training data, as explained in the previous paragraph. This process started by encoding the training data by converting the words of our CS5 files to vectors that can be understood by the transformer model. Then after training GPT-2 with thousands of epochs, we were able to query the model.

We found that when we inputted a 'docstring' our program would spit back accurate python code that would compile. However, this wasn't always **the correct** python code according to the docstring prompt. :-) For instance, when we fed it the prompt "Reverse a Linked List", to our surprise, it actually outputted code that was almost correct. We also found that this would sometimes spit out random similar code from our training data. For example, we'd often get some sample of Connect_North_East from our connect4 game assignment in addition to the 'somewhat correct' python code.

For our finished progress, we have successfully completed an app that takes in a docstring and returns compiling Python code. We've also been working on building a Flask Web Application so that anyone online can input a docstring and receive estimated code from our GPT-2 model. For this, we have a working front-end but have been running into major deployment issues in an attempt to connect both the front-end and back-end because Flask does not seem to interact well with the version of tensorflow we are using. However, both pieces of the Flask app work. That is, we can successfully take input from the user on a working front-end, and we have a query-able model. They are just, unfortunately, not connected. When we get this to work, we plan to deploy our Flask app on Heroku.

Extending the Project:

In the future, which we may actually continue this summer because it would be a cool webapp to have, we would like to fix our issues with deployment to have a full-functioning web application. Perhaps using heroku, we create a publicly accessible url for people to test our application. To continue this, we'd have to finish creating an api with routes to different sections of our app. Secondly, we would also extend our training data set to improve upon the accuracy of code that is generated. Lastly, we might submit a research proposal to GPT-3 to get permission to use it. Using GPT-3 could dramatically impact our ability to output correct code; however, we still hold the sentiment that our jobs as software engineers are secure.

20-20 Hindsight:

In retrospect, if we were to do this project again we might have considered the reverse direction: given a set of functions, write docstrings that correspond to the code given. This has practical implications in the Tech Industry as properly commenting code is a pain. Additionally, this would make code more accessible to those who don't know how to code. The ability to translate a chunk of code into english would be powerful for beginner coders.