**Assignment 2: Training a Transformer model for Predicting if statements**

I had said last time it was an ordeal, but a fun challenge, and that was true but this proved to be even more challenging. From the outset it seemed simpler, instead of predicting the next token, which could be anything, we were predicting a specific thing. A targeted approach means a smaller scope right? Well, not quite. I had come in confident but was taken aback at 150,000 instances meaning functions, not lines. Intimidated by such a number, I downloaded quite a lot of data, well over what was needed, but extra data is only a good thing right?

With a few hundred million lines of python downloaded, it was time to get it all into one place. Initially I made a bash script to simply move all .py files over to a raw\_files directory to be processed, but I noticed after it was finished it had copied over a lot of .pyc and a fair number of useless files. I tried debugging for a while before realizing I was spending more time on it that it was worth. I opened a finder window, searched for kind: python script, and dragged everything into the folder, deleting the non .py files after. With them all in one place, now it was time to start processing to get what we need.

The assignment required minimum 150k functions for pre-training and 50k for fine tuning, so I need 200k functions at a minimum, the majority of which need to have if statements so the model can learn the syntax and make predictions. I began writing extractor.py, a simple script to loop through the directory of python files, strip comments and special characters, and then insert the functions only into a master txt file to house every function. This took a bit of work, surprisingly, but looking back at assignment 1 I should have expected the encoding issue. My initial plan was to utilize the ast library to identify python code and be able to separate the code gracefully, however, many of the files were not utf8 encoded so I hit errors quickly. I came up with a few ways to try and fix this, but after making some progress I had many files error due to all kinds of syntax issues, so I decided ast wasn’t the best option. Instead, I shifted to converting to utf8 and reading the files as text like I did before. I took one of the python files from the raw\_files and used it to make the loop that runs the clean\_code function, a simple helper function to remove single and multiple line comments as well as special characters. After this produced expected results, it was time to isolate functions and copy them to a different location. To do this, since function start with def, I can use a regex to identify the start of a function and then use the re.findall function to get me the list. Putting these parts together, I now have a loop to open a file, get all the functions in the file, and then send each function to be cleaned to ensure I have what I want. I tested using a few different files, making sure to use files from a couple different repos from different users to be more confident this would work on the larger data set. Once satisfied, I had part 1 of 2 in processing the data to be ready for training.

Part 2 of this process would be to convert the master txt file into a list that could be used by the huggingface model and to categorize the functions between functions with an if statement and those without. To do this, I made another script to process the extracted files, so process\_extracted.py was made. This one would be harder as I didn’t know exactly what all huggingface needed the input to be like, so I spent some time googling to understand. After reading through a decent few posts in: <https://discuss.huggingface.co/latest> their forums, I came up with a plan. Before writing to a csv, I would need to open the master file and separate the functions once more, this time paying attention to any “if” conditions inside the function. To accomplish this I decided on separating the file into two arrays. I created a helper function to loop through the file, read for the function definition and then read every line after based on indents as indents are important in python. I made a simple Boolean to keep track of whether we are in a function or not so we can keep an eye out for if statements. If we encounter one, append this function to the if array, if not and we reach the end, append to the non-if array. With this basic loop done, I tested using the a fraction of the large file I had manually counted out to know if it came to the correct answer. At first it wasn’t giving the right numbers, also errored when reaching end of file but after some tweaks I had a working loop. With this, I now just needed to save this to a csv and I’d have my input data.I gave this a few tries, but I was noticing vastly different sizes of csv files, from 200 MB to 1.6 GB. I added a number of logging statements to track why and found more issues in the lines in the file. The problem with grabbing so much data is I can’t possibly check all of it for these errors, I needed to go back and rework the master file creation to better scrub the lines before inserting into the master file.

With these tweaks I saw a final total of 430911 functions with an if statement and 1271187 without one. This data is heavily skewed towards functions without, that’s not good for the quality of the data so I need to rethink how I’m going to do this. After thinking on it and experimenting with changing my process, I think it’s easiest/best not to change anything. It’s a csv after all, I can just select the data on the other end. When I got split the data into the pre-training and fine tuning, I’ll select all the if functions and a percentage of the rest as my total set. After some further adjustments, I have my master file at 1.03 GB and the processed csv at 1.05 GB. Now it’s time for the actual mode building and training.

I have never used hugging face before or any of these models, or anything like it really, so I elected to consulting their guides on it: <https://huggingface.co/docs/transformers/en/tasks/masked_language_modeling> and other of their resources. Consulting a number of these links I learned that were a few parts I would need, certain structures for the training arguments, all of the tokenizer parts, and steps to process everything were learned from this website and a few stackoverflow/youtube videos. However, to start I would need to separate that data into the pre-training and fine tuning, further separating the fine tuning into 10% testing and 10% evaluation. I got to work by first splitting the data into the if’s and non-if’s, then decided to add a percent option to take a percentage of the total count of if functions and take that number of non-ifs. I decided on 5%, shouldn’t be too much to interfere but enough that it’s exposed to it and can handle it. With decided, we merge the 2 series together to begin our model creation process.

We now really begin to create trainer.py, the code to build our model. I called it this believing we would need to separate into 2 scripts like the The instructions made it clear that 150k would be pre-training and 50k for fine tuning, so 75% and 25%. Reading through the W3 schools on pandas dataframes and series as well as following along with the guides in the hugging face developer documentation and forums, I was able to set up the basic information to run a bert-base-uncased model. I had set up the 2 sets of data and converted it to hugging face datasets in preparation to be tokenized. I created a tokenizer helper function from following: <https://huggingface.co/learn/nlp-course/chapter7/3?fw=tf>, as well as the data collator to apply the 15% masking to the pre-training data.

With this all done, I set up the trainer and begin testing. I made a lot of assumptions, merely following along with the documentation as this is all foreign to me, so of course this immediately goes bad. For a while it’s simply lack of downloaded libraries and installations. After pip installing a lot of things and making a few revisions, I start to see progress, although it’s unbelievably slow. Hours pass with not even 10% progress through the set, so I look to how to speed things up. There are a few things I can find to tweak, however, one of the methods appears to be another version of bert entirely. I wasn’t aware there were so many, but someone mentioned switching to distilbert-base-uncased to dramatically improve the speed. Follow several recommendations like limiting to 1 epoch, setting the learning speed to 3e-5 and a batch size of 8 would make this move a more reasonable pace. Running this I now see meaningful progress, with half a day producing over 50%. Given that I have nearly a week until this is due, however, I decide to experiment a bit by using a mix of distilbert and regular bert, making use of tools like Google colab, to run several different instances of the model training method with different arguments to produce a few versions to see which is best. I let these run and check back in periodically. When I see they’ve completed, it’s time to modify the script to handle the fine tuning.

For fine tuning I made another script, fine\_tune.py to handle this. I largely copied what was done before, but with no percent masking and instead masking one if per instructions. With the model made, it’s time to load it and the fine tune dataset we had created initially. I had made several models so for this I ran an instance per folder, labeling each with a different name to keep them separated. I examined and compared the print outs from the evaluation runs, satisfied that it run correctly and so I chose the best performing out of the lot to submit for this assignment. I didn’t think to save the prints from past runs, but I remember it didn’t perform as well as I had hoped, guessing incorrectly enough that I mentioned in my email I believed it had to be an error in the dataset construction/processing phase. Unfortunately, it’s Monday night and I’m out of time so, while I wish I could further refine the process and get better results, I have to let it go for now. My final project is a continuation of this, however, so there’s hope for that yet.