Attempting to use MT5

Christopher Lee

Abstract

This document is a documentation of my experience with the MT5 model as a project over a period of a month leading up to the end of my Senior year. For this project, I used the "base" checkpoint and researched how i could use it for my own purposes, which I will outline later on. Then for the data set, I chose 4 sets of data taken from Tatoeba.org and between those sets were five languages: English, Hungarian, German, Spanish, and Russian. In the end, I did fail to get MT5 working, but nonetheless got some results even if unusable, which I will elaborate later.

1 Introduction

For my final project of my Natural Language Processing Course, i had many choices for tasks, but I chose the one that I thought I would have found the most interesting along with success. Rather unexpectedly I had received the best score I had gotten in the course on an assignment where the objective was bilingual translation between English and German. So, I thought that not only would I have a good chance of success, but I would also have fun in doing something that I felt I had a relatively good understanding in. This however did not end as I had hoped which can be attributed to many factors. One of which I can blame on my own folly or misunderstandings and some I simply cannot and yet do still have some ideas about only without the time to explore before the project is required to be submitted.

The first and major factor in my failure draws on a misconception that since I never thought to be incorrect, I never thought to try and correct. This misconception was simply that I had to implement a certain change to the pre-trained model whether it be interesting or beneficial. Then my second was due to some errors in training. Both of these factors were major contributors in my failure. Since due to my misunderstanding, I wasted time on trying to understand how I could change or modify the model, and so I settled on just trying to see how badly results could dip if one were to throw the just the last decoder hidden state and just see if I would get just the one word or if I would get a sentence with a repeating word. This led to having another time sink in order to create a working correctly in a training loop. This of course then led to less time in bug testing and making sure the default model could work. This of course did not leave me empty handed in the end as I ended up learning about two models T5 and mT5, and how both work and supposed to be trained. For my solution I chose to work with the "base" checkpoint of the model uploaded to Hugging Face. In a brief explanation my train loop was designed to throw the input_ids, attention_mask, and labels to the mt5 model, get back its loss directly, and then continue on from there for training the base model. This of course became more complicated as time went on with my gpu not being able to run with a batch size with more than 6 as even at 7 after a few iterations the model would eat up too much memory and stop. This required that I implement gradient accumulation in order to compensate. The results were abysmal and unsatisfactory for lack of a better professional term as I will elaborate later.

2 Related Work

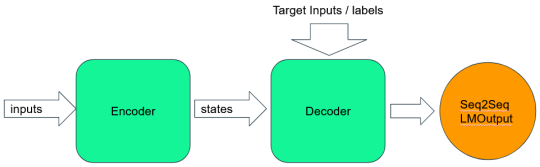
When I began, I immediately went to get a better understanding of MT5. So, I of course went to the original research and findings of Linting Zue and several others Which then also led me to the research and findings on T5 by Colin Raffel and 8 other researchers Then after trying to understand the way the models worked I turned to how they were trained via their data sets mC4 or multilingual C4 and C4. Then, I went to mT5's documentation page on Hugging Face and then its Github repository which held the pre-trained component of mT5. This led to more research as when viewing even the source code of mT5, mt5 showed to override T5 implementation meaning I then turned to the T5 documentation on Hugging Face and its source code Github repository implementation. All this meant that in the end, I was really researching two different models to an effect trying to understand and implement them. This of course was not helped by the fact T5 was also structured to return a certain type of output, Seq2SeqLMOutput, which I was not familiar with and had to also research and learn about. After finally, gaining that understanding, I developed a training loop and as stated fell into a batch size problem which I resolved with gradient accumulation, which I learned and referenced from an article, *Gradient Accumulation in PyTorch* and a stack overflow forum post which were both recommended by the TA upon a request for helpful advice.

3 Methodology

3.1 Methods

3.1.1 Model

Upon reading the documentations of T5 and mT5, I understood two things, mT5 had no recommendations of how to train the model, but T5 did and mT5 overrode T5 meaning that it could likely be trained in the same way especially when mT5 was stated within Xue and his associates' paper to be created while trying to keep close to the T5 coding and thus the overriding. With that for each iteration I threw the input_ids, attention_mask, and labels to the model to compute just as the T5 documentation stated to do with the regular t5 model. As such for my understanding of how T5 worked and by extension mT5 could be represented by the following figure:



Then using the fact that the model returns the Seq2SeqLMOutput object, I calculated the loss immediately after computation. Finally as stated I reduced the loss by the accumulation iterator [1] while stepping the optimizer and scheduler each time the batch number met a multiple of my accumulation iterator.

3.1.2 Data set

My data set is actually a collection of four data sets. Each data set was taken from the Tatoeba corpus which is a collection of sentences taken from Tatoeba.org [2]. These data sets originally were English to Hungarian, German to Hungarian, German to Spanish, and Spanish to Russian, but I had decided to implement back-translation and so what was four data sets became 8 sets of data. Moreover, each set of data was average of approximately 100k sentences or rather samples meaning my data set in total was around 800k samples. Those 800k samples were mapped and then split in a 9:1 train test split ratio except separately as 8 data sets, which I then merged and saved for usage later. Furthermore as I mapped, I added a simple prefix token that would indicate the target language for that sentence sample. For example. if an English sentence was to be translated to Hungarian I would add "<hu>" as a prefix and vice versa

[1] the desired batch size and the batch size one's GPU could handle

[2] Tatoeba is a website with thousands upon thousands of members who translate sentences from one language to another

but with "<en>" as the target language would then be English not Hungarian

3.2 Experimental Set-Up(s)

3.2.1 Model

As stated, mT5 had no instruction on how it was to be trained and so the mere fact I tried to train it under the same methodology of T5 was experimental. Then coupled with the fact I was making a branch model that took the last hidden state from whatever the original model outputted I had an interesting set up even if it did not work. For whenever an evaluation step was reached not only would the model perform extremely poorly it would also return to a massive spike in train loss. This spike was extreme and often would almost set the train loss back to what originally was. To try and remedy this I did a multitude of fixes and monitoring. I told wandb to watch the model and found no odd occurrences in the gradients, and even if the gradient had gotten high there were previous moments where the gradient had also achieved the same height, but no massive change in train loss occurred. Then I attempted to monitor both the outputs of the mT5 generate function and change the evaluation frequency in order to change the batch it would evaluate, so that I could see two things: how the model actually and if it was possibly just that certain batch the model trained on before evaluating that was the cause. In both cases, I found no issues or major peculiarities. Furthermore, I had even shrunk my learning rate constantly till I reached 0.000001 as it never made any change. Ultimately, I could not get the base checkpoint of the mT5 model to train properly and as a result could not find the time to train any other model to work as a benchmark, which would have been useless considering any result in bleu or generational length was abysmal.

3.2.2 Data set

For the data set, I did not have that much of an experimental data set, but it was certainly different. For one, the data set was a merged data set and two I used custom prefixes, that were different from the kind used to train T5. The documentation of T5 shows within an example that the prefix to use in training would be a medium sized fragment of a sentence such as "translate English to Hungarian:," but for my prefix I chose a simple token which I defined via usage of a dictionary of column names to tokens. This of course was a risk, but one that would have been interesting to see if it had worked considering instead of the prefix being four tokens given the four words used it would have been a single token.

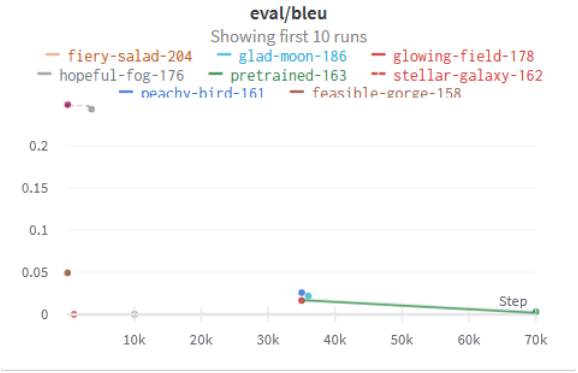
3.2.3 Evaluation

Originally, I had intended to evaluate this model using sacrebleu, bleurt, and after research possibly F1 score and accuracy considering that is what the mt5 paper had used. As time went on and my model continued to fail to train I was forced to just use sacrebleu as at that time it was the only metric I was truly familiar with with regards to using it on language modelling. Bleurt, while I had considered it from the very beginning like sacrebleu, I did not fully understand how to implement it. Then with F1 score and accuracy, I had absolutely no idea how it worked with languages modelling as both would have relied on comparing strings rather than comparing numerical scores. So in the end, all I had for the evaluation of my model was just sacrebleu. Even then that itself was not without issues as I ran into an apparently well known error involving None or empty strings which even if removed still got flagged and so required that I downgrade the version in order to avoid as one could see as i list version 1.4.14 as the required version of sacrebleu, which I list within the requirements text file in the project repository.

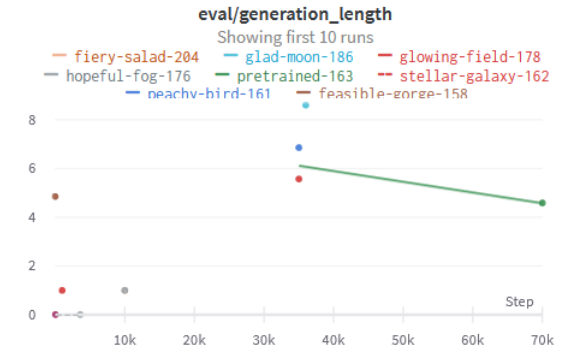
4 Results

I received no valuable results in this project as I could not successfully train my model in the end. As stated I tried many methods, but in the end my scores in sacrebleu were within a range 0.016 to 0.22, meaning 0.016 to 0.22

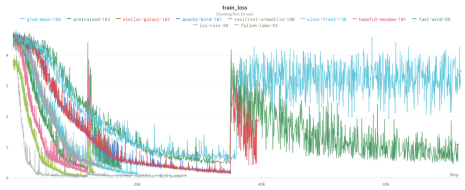
percent i.e., not even 1 percent. In every run, nothing proved to work to improve the bleu score as whatever had been causing train loss affected the bleu score as well and if I were to let any of them run they only showed to grow worse:



Even the generation_length appeared to face a similar situation only slightly better as the generational_length could reach a little above 8:



These evaluations even without comparisons can be seen as abysmal to the point I spent most if not all my time on trying to rectify the training loss problem so that something that had worked and thus guaranteed to somehow work, worked. Unfortunately all I have to show for my efforts is nothing of use or even worth comparing other models to considering no matter what changes or configurations I chose the training loss always spiked whether it be a different batch, a different learning rate, or even a different batch sizes both normally and the desired batch size like so:
Even with all that I did even going down so far as to 1e-6, nothing changed and the spike



never got any smaller. And so in the end I never managed to train my pre-trained model well enough to compare nor to say I really gained any to begin with.

5 Future Works

The reason behind the training loss is still a mystery to me, and while I do have some ideas I could not test them all in time due to commitments with other classes, graduation, and job hunting. Given more time I would like to return back to the documentation and Github repositories of mT5 and T5 to better understand how they work, and see what I could have missed or misunderstood. Then I would go back to my own code run through it again and create a debug function that works similar to the one from assignment 5 of this course where a number samples are used to train, test, and debug. After making said function, I would re-implement the logging I had to better keep an eye on what exactly was happening with mT5-base's outputs. Beyond that if the problem still remains I could try and see if the single token prefix opposed to 4-token prefix could be the key.

6 Conclusions

In conclusion, I did not complete my task or finish my project, but I learned a fair bit from attempting to do so and also received some interesting results that apparently one other group also got which was the training loss spike. As it stands, I found this project a lot of fun to be honest and if I had more credits on the Google Cloud Platform as well as more time I would definitely return to attempt recreating or even improving the results of the pre-trained mT5-base checkpoint model from Hugging Face.

A larger-Sized charts