NLP Summarization Using Multiple Languages

By: Dylan Palmer

Abstract

As an attempt at making Amazon reviews more readable for consumers, I followed hugginface's Summarization project and tried to increase the accuracy of their results. While there was some success in changing hyperparamters, most trials ended in failure due to hardware limitations or given parameters being already optimized.

1 Introduction

Summarization is a task used for taking long or complicated documents and finding the most useful or the most relevant information from said documents, then condensing that into a short, readable form. This is especially used to help make dense, domain specific knowledge like research papers or medical papers more accessible. Summarization is also useful for making any other knowledge more digestible.

I am trying to make it easier to extract relevant information from product reviews. It is difficult to sift through products and reviews at the best of times; by simplifying reviews, it allows someone to get a general consensus on a product. To achieve this result, I am using the huggingface Summarization task code. [5]

2 Related Work

There are multiple types of summarization: extractive, and abstractive. Extractive summarization finds the most important words of the original document or text then strings them together as the summary. Abstractive summarization creates new phrases based on the original text using trained models to predict relations between original texts and their summaries. This is more difficult than extractive summarization and therefor runs into more problems because of that.

With the rise in transformers, the rise in abstractive summarization has followed.[1] These researchers experimented with ways to increase the effectiveness of summarization. They found attention to be better than bag-of-words, though one researcher found that combining the old, extractive, bag-of-words method with the "new" attention increased accuracy.[2]

3 Model

3.1 Math

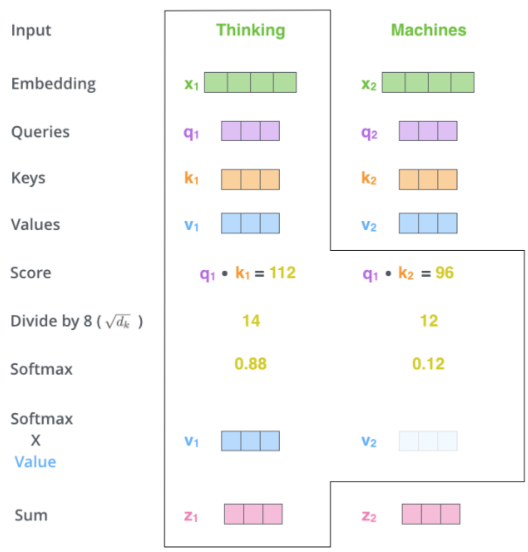

Figure 1: Attention equation [0]

The two most important pieces of math (other than matrix multiplication in general) used in my model are Attention and Backpropagation. Attention is the mechanism from which transformers get information from text. The first step is to create three vectors from the embedding by multiplying the initial input with the three matricies that are updated during training. These vectors are called query, key, and value vectors. Next, multiply the query and key, divide that number by the square root of dk, then apply a softmax to get a most likely outcome for what is important. Multiply this by the value vector for the final result.

This attention picks out specific words or features that are important and passes that on to the feed forward network as shown below

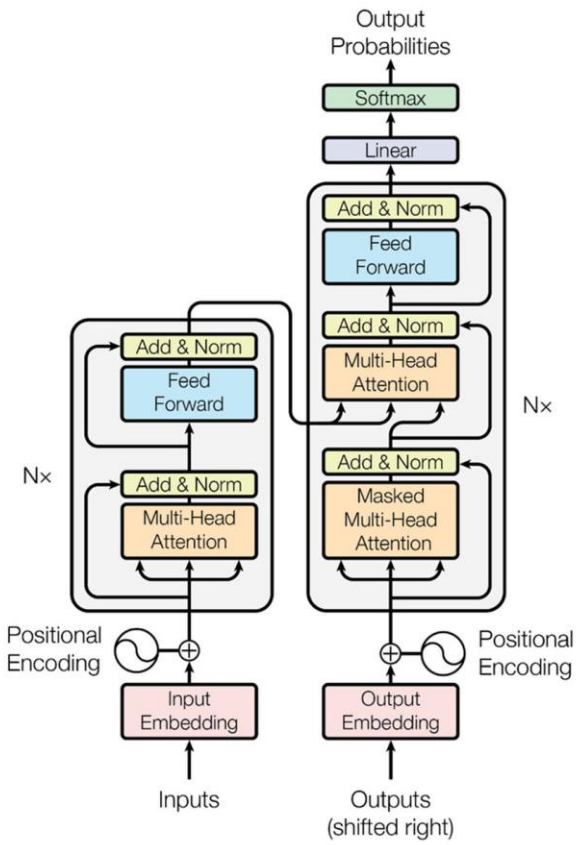

Figure 2: Transformer Layout [3]

When using multi-head attention, each "head" performs the attention process, hopefully picking out different features from the encoding to pass on.

Backpropagation is an important operation for any machine learning network. It involves taking a function, one of several are commonly used, and feeding the result of the function to the last layer, and successive derivatives to each layer before that.

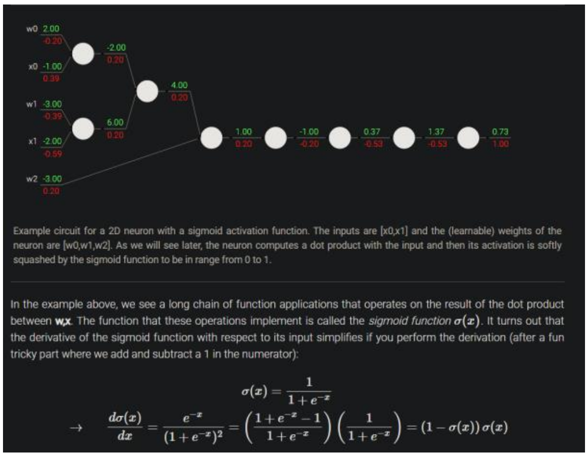

Figure 3: backpropagation [6]

By having weights that update backwards it helps train the layers faster.

3.2 Model and Preprocessing

This model is using the multilingual Amazon review dataset, which consists of 200,000 training, 5,000 test and 5,000 validation samples in five languages. These reviews span all categories from Amazon products, from books to beauty. I will be focusing on using two languages, English and Spanish.

As it takes too long to process all 200k samples, I first stuck both the English and Spanish datasets into datagrams, then filtered them by category to get a subsample. I concatenate, then shuffle these two subsamples to get a single mix of English and Spanish. Lastly, I remove titles that are too short for better results.

After finishing setting up the data, next is the tokenizer. As I am using the mt5-small model, I also use the mt5-small tokenizer. I create a function to tokenize both the body and the title of the reviews and map it to the processed data. Next, I create the model using mt5-small preprocessed model. MT5 is used due the need to train multiple languages. The small version is used to save on GPU memory and make training times faster. The last things needed are setting up the necessary arguments such as learning rate and weight decay, using a data collator, and finally, using a sequence-to-sequence trainer.

4 Experimental Results

4.1 Setup

As all values had already been chosen for the initial run, my first job was only getting the code to train, then getting it to train on GCP. Only then did I start trying to improve the results.

4.2 Process

Converting the code to run on GCP was a matter of understanding and switching some jupyter notebook conventions to work with a python script. As there there were installs inside the script I simply pre-installed those packages in GCP and removed them from the code. Since the code was from huggingface, it used the huggingface site to save and load data about the models trained. While I could have gotten this to work on GCP, it was easier to get the model to run without using the huggingface hub at all.

Since all initial values were picked, my task became changing each hyperparameter to see if that gave better results. Trying mt5-base caused the GPU to run out of memory. So did moving from a batch of 8 to 16. I tried using gradient accumulation to combat this error, but either I was using it wrong, or it just didn't help. Both the ROGUE and the loss were worse. Something else I noticed is the sample rate was only a quarter of a normal run. I assume this is due to it only updating the gradient every four runs. To combat this, I increased the epochs both with and without gradient accumulation. The result was a worse ROGUE and the loss again for both. I was later informed that early stopping code is needed for higher epoch numbers, but that's not something I learned how to do. I also never figured out how to zero out the weights of a pretrained model, so I do not have a baseline.

4.3 Results

The only thing that did work was increasing the sample size of the subsample for training. When trying the full dataset, the training time was 32 hrs, an unreasonable number, so I just added another category to the filtering. This caused the original ROGUE-L to go from 17 to 18.5

Run	ROGUE-L	Loss
Baseline	17.1	3.03
40 - Epoch	18.45	3.13
Extra Samples	18.56	2.63

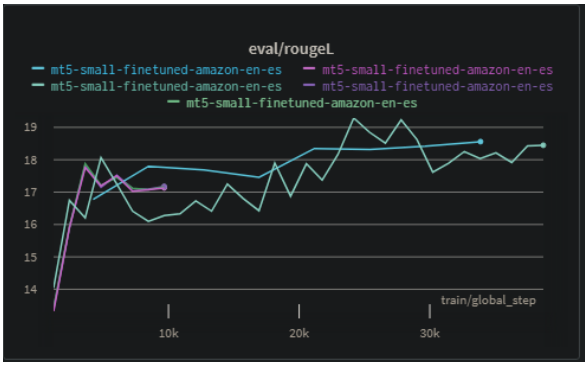

Figure 3: Rogue-L graph for runs [4]

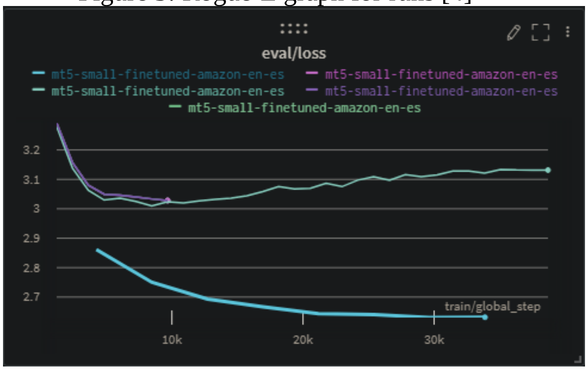

Figure 4: Loss graph [4]

I do not have saved data for the gradient accumulation as that used a different model in the code that both ended up being less accurate for some reason and did not get logged with wandb. It was putting out ROGUE-L 11 normally, with 9-10 with gradient accumulation.

5 Conclusion

Summarization is a task that takes large amounts of GPU, data, and time to get proper results with. Larger batches increase accuracy, as does longer sequence length. Both of those take more GPU. Working within the confines of the hardware I have access to means fiddling with every hyperparameter to see how that affects the results. While there is not much more I can do with this code, I can take the knowledge of optimization I learned here and apply it to future machine learning models.

References

Rush, Alexander & Chopra, Sumit & Weston, Jason. (2015). *A Neural Attention Model for Abstractive Sentence Summarization*. [1]

Yang, L. (2016). *Abstractive Summarization for Amazon Reviews.* [2]

Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). *Attention Is All You Need.* [3]

Palmer, D. (n.d.). *Weights & Biases – developer tools for ML.* Weights & Biases – Developer tools for ML. Retrieved May 5, 2022, from https://wandb.ai/site [4]

Summarization - hugging face course. Summarization - Hugging Face. (n.d.). Retrieved May 5, 2022, from https://huggingface.co/course/chapter7/5?fw=pt [5]

Convolutional Neural Networks for Visual Recognition. CS231N convolutional neural networks for visual recognition. (n.d.). Retrieved May 5, 2022, from https://cs231n.github.io/optimization-2/ [6]