The E&C Recipe Generator

Eric Tang
tang@ericdotcom.com

Connor Johnston
otevaeccfnlaxmrrcd@tmmbt.net

Abstract

We sought to create an NLP model capable of producing original food recipes. The goal of the model is to generate a title, a list of ingredients with amounts, and step-by-step instructions for cooking a recipe given a list of ingredients.

The model uses a pre-trained GPT-2 component for its encoder and decoder, and uses a pre-trained GPT-2 tokenizer. It uses BLEU as an evaluation metric. After fine-tuning the model on a recipe dataset, we found that it is able to successfully generate recipes.

1 Introduction

The question of what to cook is a frequent inconvenience. Scenarios often arise where people have food items in their pantries or fridges, but they are unsure of what they could make with what they have. To address this problem, we sought to create an NLP model that, given a list of ingredients, generates recipes.

Existing solutions for this issue include apps and websites that fetch recipes from the web. While these solutions are effective in most cases, we believe that an NLP model could produce results that are more tailored to users' needs.

The model does not simply fetch preexisting recipes. Instead it is trained on a dataset containing a large number of recipes collected from the Internet. After training on this data, it generates original text containing a recipe title, a list of ingredients along with their measures, and step-by-step instructions for making the recipe. Our goal was for the model to generate recipes that were followable and ideally of high enough quality that a person would want to eat the dish.

2 Related Work

The model we created is based on the RecipeNLG white paper (Bień et al., 2020). While the primary purpose of their paper seemed to be to advertise the new dataset they had put together, the authors also describe a model that they created aiming for a similar goal of structured text generation in a recipe format.

Early development of our model followed in their steps but, as we'll detail in the Methodology section, we made significant deviations from their previous work as we continued implementation.

3 Dataset

The dataset we used was the RecipeNLG dataset found on Hugging Face. This dataset was a collection of 2,231,142 recipes; some of the recipes were from an original Recipe1M+ set and the rest were found by scraping through the internet.

The dataset was provided as a CSV. There were several columns: the first was for titles; these were the names of the recipes associated with the row.

Next were the ingredients; these were the ingredients used for the recipe. They also included measurements for each respective ingredient used.

The third row was the row for instructions; these were instructions used to prepare the recipe. Instructions were broken down in discrete steps; there may have been multiple sentences in a single instruction.

Finally, there was a fourth row listing the crucial ingredients found in the recipe. This column was a truncated version of the ingredients column; it only listed out the major ingredients without any measurements.

4 Methodology

4.1 Model Construction

We started with a pretrained GPT-2 (Radford et al., 2019) component from Hugging Face. GPT-2 is a well known NLP component and is well suited for general text generation. We considered BERT (cased and uncased) but due to the more granular

input (defined lists of ingredients and steps) we wanted a model which considered less context. We also wanted to use a model which we hadn't interacted with before.

Initially, we started with a pretrained RecipeNLG tokenizer, but this caused issues later down the road during evaluation. The RecipeNLG tokenizer was associated with specifically the RecipeNLG model and included embeddings which we hadn't accounted for with our new model. The solution to this was to switch to a GPT-2 tokenizer and manually insert the tokens we needed.

4.2 Formatting the Input

Once we had our model more or less sketched out, we had to figure out how to get the data from the data set into the machine. As previously stated, the data was given to us as a CSV with separate columns. This is not what typically goes into a model like this; we had to format the inputs somehow in order for them to be used.

The first step was simply to concatenate all of the inputs into a single string. We took the NER (important ingredients column), full ingredients, instructions, and title and combined them all into a single string. Referencing the original paper, we used special tags to delimit each of the inputs and input sections.

Later on, it was decided many of the inputs were redundant. Furthermore, the high amount of tokens was causing inflated evaluation metrics. To resolve this, we culled the amount of tokens we were using to only what was necessary.

4.3 Training

Now that we had our input, we could begin training.

Our first attempts at training involved simply passing in the input in its entirety (including the NER, ingredients, instructions, title) and attempting to have the model copy it to the output. We thought that learning the associations across the entire recipe would be enough for the generator to understand what to fill in when needed. However, this was too far from the task we were trying to do, and the model wasn't learning the task. When we attempted to generate a recipe, it became apparent that the model hadn't even learned the structure. A redesign of the training task was needed.

Thinking about what our task was specifically, it was inputting a small list of ingredients and translating it to a whole recipe. What we settled on

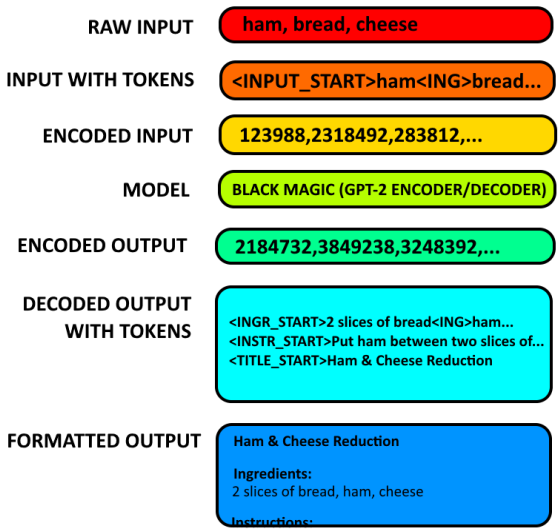

Figure 1: A diagram illustrating the structure of the model.

for training was a greatly truncated input involving only the NER, the most important ingredients. This was then compared against the whole recipe associated with those ingredients, measurements and all. This yielded much better results; not only did the model pick up the kinds of instructions and measurements associated with each of the important ingredients, it learned the expected format of the output as well.

We trained for 80,000 iterations over 10 epochs. We used a linear learning rate optimizer which started from 5e-5 and decreased to 0 over the course of training. We experimented primarily with sequence lengths during training. Longer sequence lengths had the opportunity for lengthier and more interesting recipes. However, due to the lack of resources, we had to limit our runs to a max sequence length of 128. This was a compromise between a reasonable generation length and an interesting recipe.

5 Results

5.1 Scoring Metrics

The original RecipeNLG model used cosine similarity between generated recipes and known valid recipes as an evaluation metric. The LanguageCheck spell and grammar checker was also used to evaluate the amount of linguistic mistakes. Additionally, BLEU, GLEU, and WER were used as translation metrics.

Our model only used BLEU as it was much simpler to implement. We understood that it was a

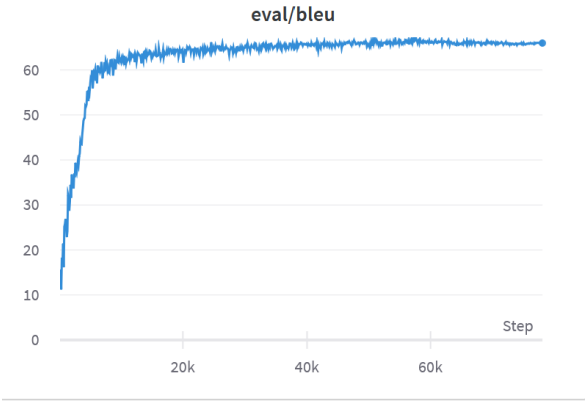

Figure 2: The bleu scores associated with the training. Our model reached around 65.

good metric for text generation and didn't see the need to complicate it further.

After training we found that the fine-tuned GPT-2 model greatly outperformed the base model with regard to this metric, but that the BLEU score was unrealistically high. We believe that this is due to the locations of special tokens being disproportionately easy to learn in comparison to the tokens that compose a recipe. To address this issue, we cut several special tokens from the input and output, only keeping those that were necessary to define the structure of a recipe.

Even after doing so, we observed BLEU scores that were unrealistically high. Our best run reached a score of 65, which normally indicates superhuman performance. Although the factor of special tokens continued to affect our BLEU results, we were satisfied with the curve of the BLEU score during training as well as the results seen during manual testing.

In addition to the BLEU score rising throughout training, we observed a decreasing loss curve as well as an increasing batch word accuracy curve. These metrics indicate that the model successfully learned the training task.

Measuring and evaluating BLEU, loss, and batch word accuracy were good strategies for confirming that the model was learning, but to get a better picture of how well it would perform from a practical standpoint we did manual testing as well. This involved giving the trained model ingredients for input and evaluating the output based on whether we felt the recipe was followable and edible.

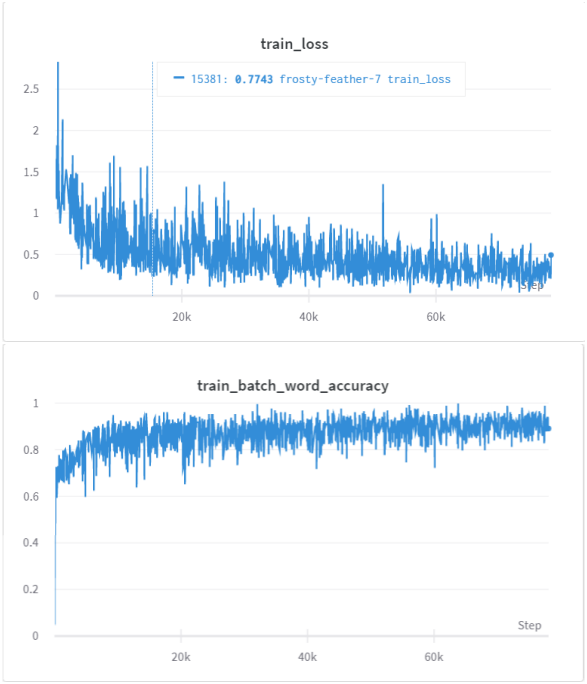

Figure 3: Loss and batch word accuracy during training.

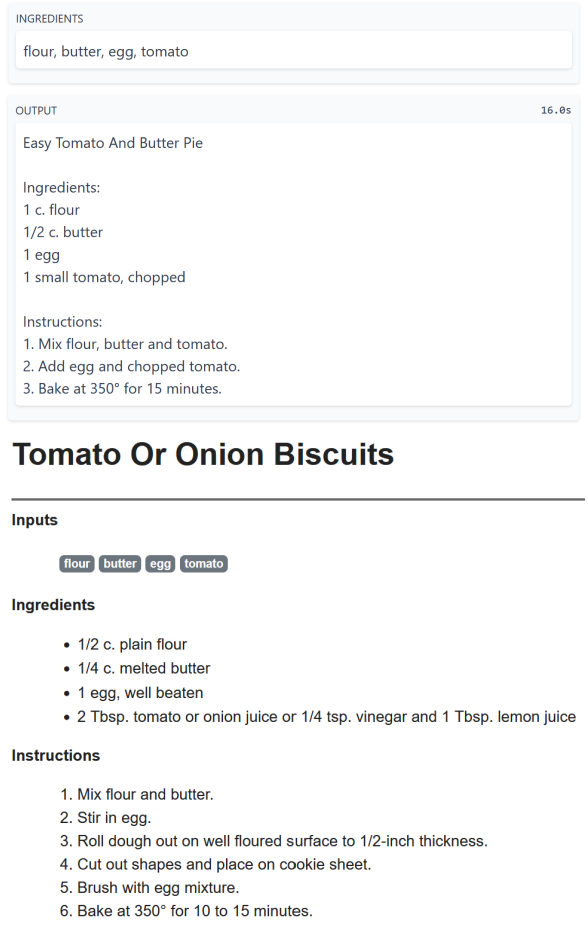

Figure 4: Top: Our model's output. Bottom: The original RecipeNLG's output.

5.2 Human Evaluation

In addition to BLEU, we manually generated recipes after each training exercise and evaluated the output to verify what was expected.

While most recipes generated were short, they satisfied those criteria for the most part. The short length of generated recipes can be attributed to a low sequence length during training. When we trained the model, we used a maximum sequence length of 128. We expect that with a higher value, recipes will be lengthier and potentially of higher quality. Due to time constraints as well as hardware limitations we have only evaluated model performance with this length however.

Eyeballing the output of our model compared to that of the original RecipeNLG output, we found that our output was either on par with or exceeded in terms of coherence and relevance. In addition to this, you can specify any ingredients for our model; RecipeNLG only has a predefined list of ingredients. Finally, our demo doesn't require you to submit an e-mail making it much more convenient.

6 Conclusion

After evaluating the results of our model, we can conclude that GPT-2 can be fine-tuned to learn how to generate cooking recipes. We found that this is possible even with fairly limited hardware. Our final model was only trained for roughly 4 hours using a batch size of 2 on an RTX 3070, yet the performance was still acceptable with regard to human evaluation. Using higher grade hardware to train with a greater sequence length and batch size would likely yield even better results.

An idea for a possible modification of the model is to generate recipes based on a title as input. Perhaps a user could input something like "banana pancakes" into an application powered by a model similar to this one and be given a recipe. The only thing that would need to be modified would be the training task; instead of inputting the NER you could just input the title or some other string.

Overall, we're satisfied with our results and look forward to cooking some incredible dishes using it.

Acknowledgements

Thanks to the authors of the original paper (cited below) and Vlad for helping us out.

References

Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. 2020. RecipeNLG: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, Dublin, Ireland. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.