From Text to Table:

Sequence-to-Sequence Modeling for Recipe Instruction Generation

W 266: Natural Language Processing
UC Berkeley School of Information
bheuberger@berkeley.edu

Abstract

Creating new and compelling recipes from unique combinations of ingredients is a challenge that has preoccupied home cooks and professional chefs alike for centuries. Developments in deep learning have opened the door to new methods of text generation that have promising applications to the culinary world. In this study, I build sequence-to-sequence models using Long Short-Term Memory (LSTM) networks to generate novel recipe instructions from title and ingredient inputs. Models are able to produce coherent and interesting recipe instructions, particularly when integrating an attention mechanism. I further explore how beam search and recipe length affect output quality.

1 Introduction

In recent years, with the advancement of natural language processing (NLP) techniques and deep learning models, the generation of structured text has garnered significant attention. Cooking recipes represent a unique form of structured text, characterized by long, interrelated sequences of ingredients, instructions, and culinary techniques. The generation of high-quality recipes is a challenging task that requires approaches capable of not only capturing language semantics but also a nuanced comprehension of culinary knowledge and contexts. Success requires the explicit representation of ingredients in outputs, coherence of instructions, and the ability to capture the implicit relation between ingredients and preparation methods. Automated recipe generation has many practical applications and can serve as a valuable tool for meal planning, personalized recipe recommendations, and culinary education.

In this paper I develop a pair of recipe generation models using advanced recurrent neural network (RNN) architectures and attention mechanisms. I then analyze the effectiveness and performance of these models in generating coherent recipes across varying lengths and complexities. My work contributes to the growing body of research in automated text generation and provide insights into the capabilities and limitations of current recipe generation models. It also has practical implications for home cooks, food enthusiasts, and recipe developers, in additional to future innovations in culinary AI applications.

2 Project Overview

2.1 Background

While initially applied to the task of machine translation [7], in recent years researchers have used encoder-decoder architectures for text generation in the context of cooking recipes [2]. Gona et al. (2020) built an ensemble architecture that uses bi-directional LSTMs to classify existing recipes according to user preference dietary preferences and generate new recipes from existing recipes [3]. Salvador et al. (2019) used a convolution network encoder that processes food images and a transformer decoder that generates ingredients and instructions [6]. Other approaches include bidirectional gated recurrent units with reinforcement learning [1] and fine-tuned pre-trained transformer models [4].

There is good theoretical justification for applying attention mechanisms to the task of recipe generation: ingredients are often listed in the order in which they appear in the instructions, with ingredients most essential to the recipe often appearing earlier.

One question I faced when deciding how to integrate attention was whether to have two separate attention layers for each of the instructions and title inputs, or to concatenate their encoded vector representations and have a single attention layer over this. Similar prior research has found the latter lead to better results, which is what I opted to do in my model [6].

2.2 Dataset and Preprocessing

I used a dataset of over 100,000 recipes web-scraped from FoodNetwork.com, Epicurious.com, and Allrecipes.com, available at: https://eightportions.com/datasets/Recipes/
Each recipe contains a title, list of ingredients, and instructions for preparation.

Significant preprocessing was done to prepare the data for modeling. Over 27k recipes contained erroneous duplicate sentences, which were removed, as were any recipes that were missing either title, ingredient, or instruction data (approximately 700). Because many recipes contained common cooking abbreviations (e.g., tbsp for tablespoon), I mapped common cooking abbreviations to their full-word equivalents. I further normalized text by removing accents from accented characters, removed punctation other than periods, and removed all parenthetical text from recipe instructions because these were typically asides that were not essential to the core of the recipe.

I also examined the distributions of recipe ingredients and instruction lengths separately, both of which were right-skewed. As RNNs can struggle with longer text sequences, I removed recipes with ingredient or instruction lengths that exceeded thresholds determined based on the distributions. Finally, I included special tokens: <NUM> tokens replaced numerical values, <PAD> tokens ensured that inputs had a consistent length, and <START> and <END> tokens were used to bound the instruction text.

2.3 Embedding and Tokenization Strategy

Initially, I experimented with training my own Word2Vec model, but ultimately decided to use pre-trained GloVe embeddings: https://nlp.stanford.edu/pubs/glove.pdf. I utilized the "glove-wiki-gigaword-100" embedding model, which is trained on a combination of the Wikipedia 2014 and Gigaword 5 dataset. The pre-trained embeddings capture rich semantic information by representing words as dense vectors in a high-dimensional space. After initial experimentation with lower dimensionality embeddings, I found that using higher values consistently led to consistently better results. Given the preponderance of highly specific food and cooking vocabulary in my data, I allowed the embeddings to be trainable to capture the nuances of the cooking vocabulary.

The GloVe embeddings contain a vocabulary of roughly 400k unique tokens. A corpus of this size was beyond the memory capacity available, so I devised a strategy to prune and then selectively expand the vocabulary using cosine similarity. I began with the GloVe vocabulary found only in my training data—approximately 18k words—and then looped through these words to iteratively add additional tokens in the GloVe vocabulary that had similar embeddings to those in my training data (i.e., the cosine similarity within a defined threshold). This added an additional 35k unique word tokens in the vocabulary corpus. The selective expansion of the initial pruned vocabulary was critical because it led to a lower percentage of unknown tokens when performing inference on out-of-sample data.

3 Model

3.1 Model Architecture

The sequence-to-sequence models I built leverage a multi-layered LSTM network that captures temporal dependencies within input and output sequences. I explored two similar yet functionally distinct models. Both consist of two fundamental components: an encoder and a decoder. The encoder stage processes the input sequences, specifically recipe titles and ingredient lists. Embedding layers initialized with the pre-trained GloVe embeddings transform words into dense vectors, which are passed through LSTM layers that encode the inputs into fixed-length context vectors. At the decoder stage, another LSTM network generates recipe instruction sequences iteratively, one token at a time.

In both models, I implement teacher forcing to aid the training process. During training, the true value of the output at each time step is passed to the decoder as input, rather than using the previously generated token. This technique accelerates convergence by providing the decoder with the correct token at each step.

The base model serves as the foundational architecture, wherein the final hidden states of the title and instructions LSTM layers are concatenated and transmitted to the decoder. The concatenated hidden states serve as the initial inputs for the first timestep in the decoder LSTM. The refined model (hereafter: "attention model") integrates an attention mechanism that dynamically weighs the significance of different segments within the input sequences during the decoding phase. This enhances the model's ability to hone in on salient information, refining the quality of generated instructions. Notably, the attention model diverges from the base model in that it foregoes the use of the final hidden state of the encoder to initialize the first timestep of the decoder. Both architectures are depicted in Figure 1 using notional tokens.

I train the models with early stopping for a maximum of 30 epochs with a batch size of 128 on a NVIDIA A100 GPU, using a learning rate of 0.005 and a dropout of 0.05. (Hyperparameters were tuned but kept consistent across models).

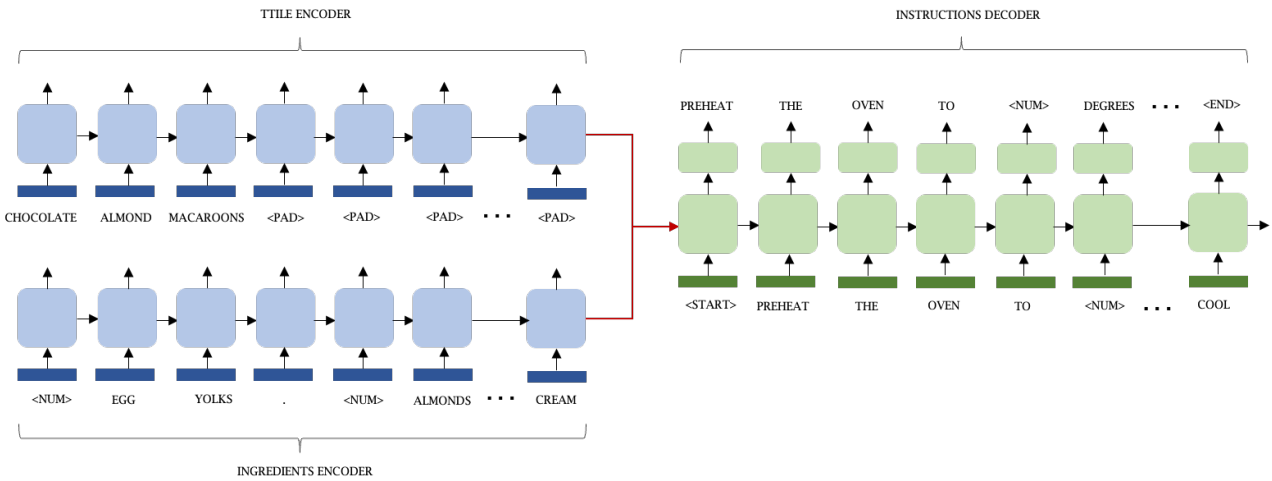
3.2 Inference

During inference, the ground-truth sequences are not available. Rather, at each step in the sequence, the previously generated token is passed as input at the next step in an auto-regressive manner. This process is repeated until an <END> token is produced or a specified maximum sequence length is reached. I implement both greedy and beam search as described in the results section.

3.3 Evaluation Metrics

I evaluate the quality of generated recipe instructions using two standard metrics: ROUGE (Recall-Oriented Understudy for Gisting Evaluation) and METEOR (Metric for Evaluation of Translation with Explicit Ordering). ROUGE measures the overlap of n-grams between the generated text and the reference text. METEOR is similar, computing the harmonic mean of precision and recall, but incorporates stemming and other methods that result in scores that have shown improved correlation with human evaluations over the more popular BLEU. Consistent with best practices, I remove special tokens prior to evaluation. Although ROUGE and METEOR are typically applied at the sentence level, I apply them to each generated recipe as one block, consistent with related work.

(a) Model architecture with concatenated hidden state vectors and no attention mechanism:



(b) Model architecture with attention mechanism:

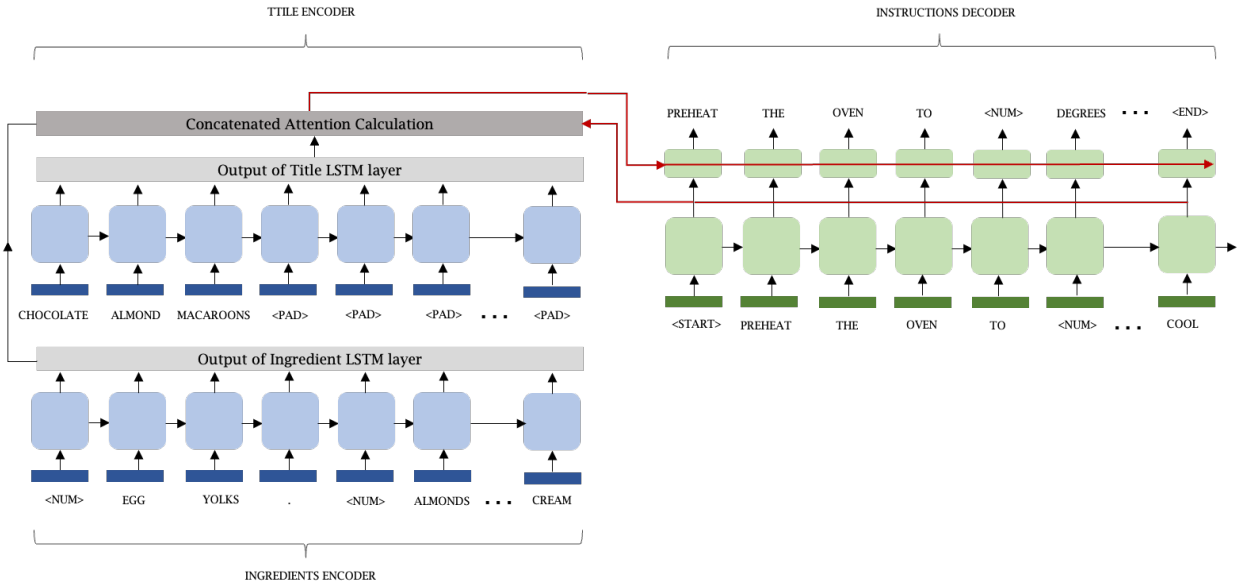


Figure 1: The two Seq2Seq model architectures, stylized. Blue and green are used to distinguish the encoder and decoder, respectively. Dark rectangles represent an embedding layer, squares represent LSTM cells at each step in the sequence, and black arrows are cell outputs. Red arrows represent connections between the encoder and decoder. In case (a), this is the concatenated final hidden states of the two encoder blocks. In case (b) these are the query and context vectors used in the attention calculation.

4 Results and Discussion

4.1 Generated Recipes and Beam Search

Validation loss from the base model and attention model over training is shown in Figure 2. The attention model outperformed the base model on both ROUGE and METEOR F1 scores by about 75%, with a ROUGE-1 F1 score of 0.37 as compared to a score of 0.21 in the base model (Table 1). These differences were particularly large for n-gram recall, with the attention model score more than double that of the base model.

For each model, inference was performed using beam search with widths of 1 (i.e., greed search) 5 and 10. Surprisingly, I found no significant differences in evaluation scores between these configurations. (Full results are not shown here for sake of brevity but available in code). Beam search aims to find the most probable sequence within a finite set of a search tree. While this may improve the fluency and coherence of the generated sequences, it does not necessarily increase overlap with the reference text and thus may not result in improved scores. Future extensions of this model could look to refine the beam search approach and its influence on generated output quality.

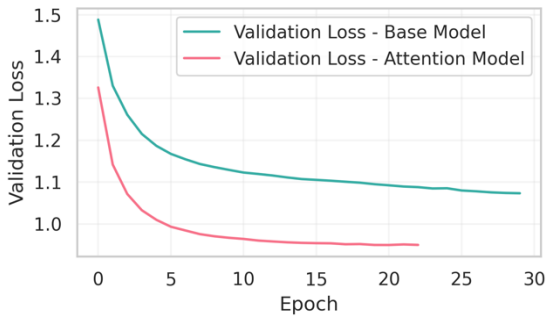


Figure 2: Validation Loss Across Training Epochs

	Base Model	Attention Model
ROUGE-1 Precision	0.23	0.34
ROUGE-1 Recall	0.23	0.48
ROUGE-1 F1	0.21	0.37
ROUGE-L Precision	0.17	0.37
ROUGE-L Recall	0.17	0.26
ROUGE-L F1	0.15	0.29
METEOR	0.15	0.26

Table 1: Evaluation Metrics on Generated Output, Beam Width of 1

4.2 Qualitative Analysis

Qualitatively, the generated instructions—particularly using the attention model—are generally coherent, grammatically correct, and logically structured. An example of the generated output is shown in table X, with others available in code. The results show that the model is successful in capturing the implicit relationship between certain types of ingredients and the typical preparation methods. For instance, cookies are baked and require preheating an oven to a specific temperature, pasta involves boiling a pot of water, and cocktails are often shaken or stirred before being poured into a glass.

That said, the model has clear limitations. First, some results contain long sequences of repeating n-grams. Although not considered here, adjusting the randomness of the token generation process (i.e., through a temperature hyperparameter) might address this issue. Second, some generated recipes also omit key ingredients or steps in the preparation process. Future enhancements could incorporate mechanisms for explicitly representing all key ingredients within the instructions. Incorporating a mechanism that tracks which input items have been mentioned in output and which haven't has shown promise in other work [5].

Title	hot chicken wing dip
Ingredients	<NUM> pound skinless boneless chicken breasts . <NUM> bottle ranch dressing . <NUM> packages cream cheese softened . <NUM> bottle hot pepper sauce . <NUM> package shredded cheddar cheese .
Actual Instructions	preheat oven to <NUM> degrees f . heat a non stick skillet over medium heat . add chicken breasts and cook until juices run clear about <NUM> minutes per side . remove to a cutting board to cool completely . use a fork to shred the meat into strands . pmy the ranch dressing cream cheese and wing sauce into a bowl and mix until combined . add the shredded chicken and mix well . spread the mixture into a <UNK> inch baking dish . sprinkle the cheddar cheese over top of the chicken mixture . bake in the preheated oven until the cheese is bubbling about <NUM> minutes .
Generated Instructions with Attention (beam width = 5)	watch how to make this recipe . preheat the oven to <NUM> degrees f . place the chicken on a baking sheet and bake for <NUM> to <NUM> minutes or until the chicken is no longer pink in the center and the juices run clear . an instant read thermometer inserted into the thickest part of the chicken breast should read at least <NUM> degrees f . remove the chicken from the oven and allow to cool . in a large bowl combine the ranch dressing cream cheese hot pepper sauce and cheddar cheese . pour the mixture into the prepared baking dish . bake for <NUM> to <NUM> minutes in the preheated oven or until cheese is melted .
Generated Instructions without Attention (beam width = 1)	place chicken breasts in a large bowl . mix chicken and chicken together in a bowl . pour chicken mixture into the prepared baking dish . bake in the preheated oven until chicken is no longer pink in the center and the juices run clear about <NUM> minutes .

Table 2: Example of generated and actual recipe instructions.

4.2 Recipe Length

I also investigated how the quality of generated recipes depends on length and whether the relationship between length and quality differed between models. While recipes with longer ingredients often have longer instructions (Pearson's correlation = 0.48), this is not always true. Therefore, for each recipe in the test set I computed the average number of tokens across the ingredients and instructions, thus considering the length of not just the ingredient sequences but also the expected outputs. Recipes are then sorted into deciles by this length and generated instruction quality is evaluated for each decile. Results from these experiments are shown in Figure 3.

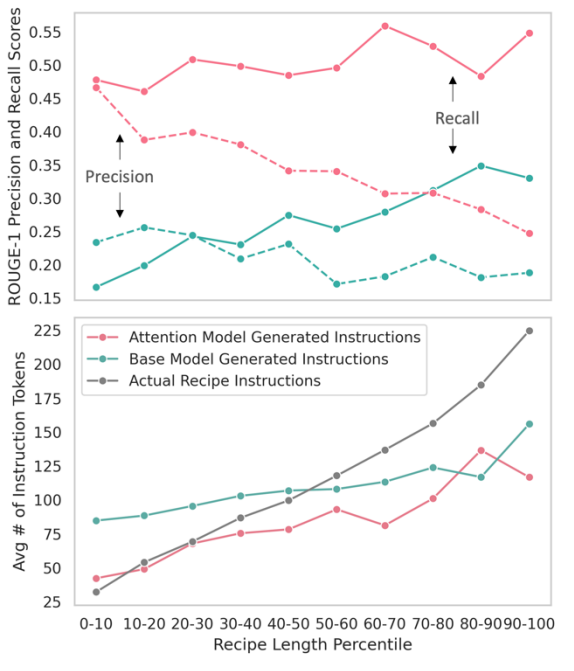


Figure 3: Evaluation metrics and generated length by recipe length percentile. Recipe length is a simple average of the number of tokens in the ingredients and ground-truth instructions of each recipe in the test set.

Both models generate longer instructions for recipes with longer ground-truth ingredients and instruction sequences. While both tend to overshoot length on the shortest recipes and undershoot length on the longest recipes, the attention model better correlates with ground-truth value length, particularly for shorter recipes. Interestingly, the base model generates longer recipe instructions for nearly all the deciles examined. Indeed, qualitative evaluation of this phenomenon shows that the output from the base model tends to be less concise, including more repetition and the omission of key details.

Model recall rises and precision falls for longer recipes. Successful recipe generation arguably depends most on the ability of the model to capture all essential ingredients and preparation methods that appear in the reference output, which is measured by recall. Particularly for the attention model, recall is higher than precision across all lengths.

The attention model ROUGE-1 scores are higher than that of the base model across all recipe lengths. While I expected the attention model to outperform relatively more on longer recipes because of its ability to capture long-term dependencies, the differential was actually greater for shorter recipes. This may be in part to the challenge the base model had in modulating its length appropriately. Additionally, shorter recipes contain fewer elements, which may make it easier for the attention mechanism to attend to the most important details. More investigation is needed to fully explore this question.

Conclusion

In this paper, I present a pair seq2seq models using LSTMs to generate recipe instructions from title and ingredient inputs. I demonstrate that these models can be used to generate plausible recipe instructions and that the attention model significantly outperforms the baseline model across all evaluation metrics, with results varying by sequence length.

Future extensions of this model could address some of its limitations, such as modifying the inference model to help avoid repetition in generated sequences, or using reinforcement learning and other approaches to ensure key ingredients are not omitted from generated instructions. New or modified model architectures could also be considered, such as a bi-directional LSTM to better capture long-term dependencies over long sequences.

References

[1] Fujita, Jumpei, Masahiro Sato, and Hajime Nobuhara. “Model for cooking recipe generation using reinforcement learning.” 2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW). IEEE, 2021

[2] Galanis, Nikolaos-Ioannis, and George A. Papakostas. “An update on cooking recipe generation with Machine Learning and Natural Language Processing.” 2022 IEEE World Conference on Applied Intelligence and Computing (AIC). IEEE, 2022.

[3] Gona, Sai Nikhil Rao, and Himamsu Marellapudi. “Suggestion and invention of recipes using bi-directional LSTMs-based frameworks.” SN Applied Sciences 3 (2021): 1-17.

[4] H. Lee, Helena, et al. “RecipeGPT: Generative pre-training based cooking recipe generation and evaluation system.” Companion Proceedings of the Web Conference 2020. 2020.

[5] Kiddon, Chloé, Luke Zettlemoyer, and Yejin Choi. “Globally coherent text generation with neural checklist models.” Proceedings of the 2016 conference on empirical methods in natural language processing. 2016.

[6] Salvador, Amaia, et al. “Inverse cooking: Recipe generation from food images.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[7] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. “Sequence to sequence learning with neural networks.” Advances in neural information processing systems 27 (2014)