Recipe for Disaster: A Seq2Seq Model for Recipe Generation

Dev Bhargava
Department of Computer Science
Stanford University
Stanford, CA 94305
devb@stanford.edu

Thomas Teisberg
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
teisberg@stanford.edu

Abstract

Text generative recurrent neural networks (RNNs) have proven to be highly effective in producing locally coherent text but struggle with tasks that require either global coherence or a deep semantic knowledge of previously appearing tokens. In this paper, we present a model that takes a set of recipe ingredients as an input, and outputs a sequence of instructions for combining the ingredients in a realistic manner. Our model is a sequence to sequence network made up of an encoder and a decoder, each of which leverage LSTM cells, dropout, and attention. Our network successfully demonstrates the ability to generate novel series of instructions for sets of ingredients both seen and unseen.

1 Introduction

Over the last couple of years, there has been a tremendous amount of progress in the domain of deep learning with applications in natural language processing in emulating human writing structure [1], question answering [2], and machine translation [3]. While models in these fields continue to improve at rapid rates, there is still a relative void in the subspace of natural language processing that deals with long sequences of data that are highly interrelated. Recipes are an interesting case study for this: specifically, recipes can be interpreted as a pair of ingredients and instructions that govern how to mix the aforementioned ingredients. Predicting a list of instructions for a set of given ingredients is a problem that has been partially explored but remains unsolved. The problem itself comes with two major challenges. First, it requires relational coherence: that is, the output instructions must specifically contain the tokens found in the ingredients. Secondarily, recipes can be very long but must maintain global coherence.

At a high level, our goal is to solve exactly this problem. In particular, we present an end to end generative model that takes as input a set of ingredients and outputs a list of instructions that specify how to combine the ingredients to create a put together dish. In addition to exploring an unsolved and difficult space of computational problems, producing such a model has commercial applications - indeed, a solution to this problem would yield both theoretic and pragmatic benefit.

2 Related Work and Background

We split the previous work in accomplishing the task of recipe generation into two distinct categories: a technological side in which the mathematical foundations of our methods were developed and the practical side, in which other technologies were used to generate preliminary results on the task.

We begin with a technical exploration of previous work. Recurrent Neural Networks were first used to model language by Mikolov et. al in 2010 [4], who demonstrated the applications of RNNs to

speech recognition. This work kickstarted the use of RNNs in modeling the English language and demonstrated their effectiveness in accomplishing such a task. Sutskever et. al (2014) [5] were the first to publish work specifically using a sequence to sequence model (in which both the inputs and the outputs of the model are sequences), successfully performing machine translation on long sentences. Luong et. al (2015) [6] showed the efficacy of using attention for seq2seq models when dealing with large inputs or outputs, in their case focusing again on the task of machine translation.

The task of recipe generation dates back to 1986 when Hammond [7] first used machine learning in a recipe planning context. Notably, Hammond's work divided the recipe generation task into two separate subtasks: first, determining the content itself, and then converting content to a coherent string of words. Lapata and Barizalay (2015)[8] focused on the second subproblem and presented a model and framework from which to evaluate the coherence of specific words. Mori et. al (2014) [9] attempted an end to end recipe generative system using a flow graph and filling in words based on recipe patterns. Most notably, Kiddon et. al (2016)[10] used a neural checklist attention model to efficiently ensure local coherence and that items in the ingredients list were found in the instructions list. This is the current gold standard for coherent recipe generative deep learning models, but is convoluted and heavy handed. We seek to achieve similar results with a more lightweight architecture that is faster to train and evaluate.

3 Dataset

3.1 Overview

We use the MIT Recipe1M Dataset [11]. This dataset contains over 1 million recipes with titles, ingredients, instructions, and pictures. Both the ingredients and the instructions were formatted as lists. These recipes were scraped from a variety of cooking websites, many of which are user-contributed content. This means that typos and highly unusual recipes are common occurrences.

3.2 Pruning

To reduce the size of the output vocabulary and decrease the overall computational power required to train on the dataset, we perform thresholding to prune the number of examples considered by the dataset. In figure 3 (see appendix), we show the correspondence between the number of tokens found in each of either the set of instructions or the set of ingredients and how such numbers relate to the percentage of examples from the dataset the model can cover. In the majority of our experiments, we set the maximum number of tokens in the ingredients list to be 10 and the number of tokens in the instructions list to be 50. As per figure 3, with these limits, the dataset is pruned to cover roughly 10% of all possible examples. In other words, the modified dataset contains about 100,000 examples. For efficient training and testing of the model, we further reduced this to a dataset containing 25,000 examples.

3.3 Preprocessing

In our first step of preprocessing, we convert the ingredients and directions each into a single text field (they are broken up into individual ingredients and directions in the original dataset). We also use a unit-parsing Python package to detect and remove all identifiable quantities. Finally, we remove anything inside of parentheses (which are often notes on substitutions, and subsequently add no value to the model itself) and strip the examples of any characters that are not alphanumeric or whitespace.

Next, we must apply padding. As noted in the previous section, we make the choice to limit the number of tokens in the instructions list to 10 and the ingredients list to 50. To ensure that all of the inputs and outputs have standardized length, we first apply a special `<END>` token to each of the list of ingredients and list of instructions before applying `<PAD>` tokens to the list of ingredient tokens until its length is 10 and to the list of instructions tokens until its length is 50. Additionally, for the instructions list (which will serve as the output of our model), we also prepend a special `<START>` token.

4 Methods

4.1 Word Embeddings

After the examples have been preprocessed, we are left with a padded set of tokens that represents the ingredients, and a similarly padded set of tokens that represents the instructions for a particular recipe. Next, for each of the tokens for the input ingredients, we look up a 50 dimensional embedding. The choice of embedding (whether to use a pretrained embedding or randomly initialize these embeddings) is a hyperparameter of the model. For a specific example $x^{(i)}$, then, we have $x^{(i)} \in R^{10x15}$, where 10 represents the number of tokens in the input sequence (including padding) and 15 is the dimensionality of the embedding.

4.2 Architecture

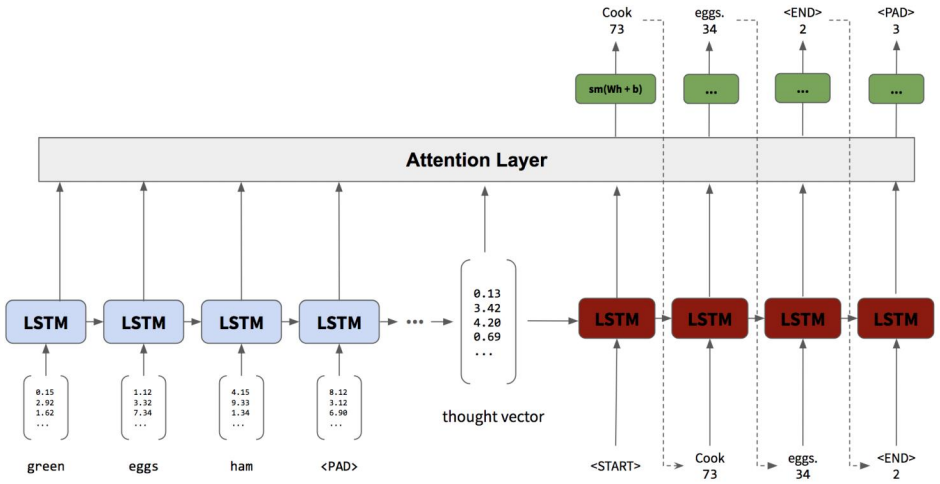

Figure 1: Model Architecture

The architecture of the actual model is a sequence-to-sequence model with Luong attention [6]. The model consists of two interdependent submodules: first, an encoder that takes in a series of ingredient embeddings of length 10 and outputs a thought vector that is a latent, low dimensional representation of the recipe and second, a decoder that takes this thought vector as primary input and outputs a series of tokens that represents the instructions for combining the ingredients together.

Each of the two submodules (the encoder and the decoder) are Long Short-Term Memory networks with attention. We use LSTMs for the encoder and decoder because of the relative length of the input and output sequences. Formally, this means that for an input x, we have:

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W(\begin{bmatrix} h_{t-1} \\ x \end{bmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Note that in this model, we additionally use Luong Attention [6]. For the sake of brevity (and adhering to the 5 page limit), the in depth nitty gritty mathematical details of attention are left out of this paper. Equations 4-7 of [6] describe the mechanism in great detail for the interested reader.

Including attention, however, we are left with some $\hat{y}$ that is a function of h_t as above, and

a handful of other parameters that are used in attention to ensure that specific regions of the input are heavily considered when producing output tokens.

In our model, we have $\hat{y}_{enc} \in R^{50}$ and $\hat{y}_{dec} \in R^{50x|V|}$. To compute our final $\hat{y}$, that is, the logits for the output words, we compute one more softmax over $\hat{y}_{dec}$ to end up with output probabilities. We then use the argmax over $\hat{y}$ to compute the predicted word.

For completeness, we note that the tensorflow seq2seq tutorial [12] and udacity [13] seq2seq model were both very helpful in building our own architecture.

4.3 Optimization

The output logits $\hat{y}$ of the model for a specific example will $\hat{y} \in R^{50x|V|}$, where 50 is the maximal padded length of the instruction tokens. Given some $yinR^{50}$ that are the labels corresponding to the actual words of the recipe, we expand y such that we have $y \in R^{50x|V|}$ and then compute the cross entropy loss L as follows:

$$L = -\sum_{i=1}^{T} y \log \hat{y}$$

5 Discussion

5.1 Experiments

5.1.1 Attention

Attention mechanisms allow stages of the output RNN to learn how to refer back to specific hidden states of the input RNN. In the context of recipe prediction, we expected (and found) that attention will help significantly as recipe instructions tend to reference ingredients in the order they are mentioned. We implemented Luong attention[6] and show its improvement in the results section below.

5.1.2 Pretrained GloVe Embeddings

Initially, we though that using pretrained word embeddings would help reduce our training time. We experimented with using 50-dimensional pretrained GloVe embeddings [14] and not backpropagating into the embeddings. Unfortunately, we had little success with this approach. We abandoned the GloVe embeddings in favor of lower-dimensional (we use 15) learned embeddings. We suspect that using embeddings trained on general text is suboptimal for a domain-specific task where small differences between cooking-related words matters a great deal.

5.1.3 Dropout Regularization

Dropout [15] is a mechanism of avoiding overfitting to the training set by randomly removing some percentage of the neurons from the model while training on each batch of data. We experimented with both dropout and L2 normalization but found that our results were highly dependent on our choice of the L2 penalty weight whereas dropout performed well without much hyperparameter tuning. We show the results of adding dropout to our model in the results below.

5.2 Results

The hyperparameters of our model included hidden size, which we set to 50, learning rate, which we set to 10^{-3}, embedding size, which we set to 15 and dropout keep probability, which we set to 20%. Some of these hyperparameters (dropout rate, learning rate) were selected by reviewing relevant literature for networks with architecture similar to our own. Others (embedding size, hidden size), we set doing a random hyperparameter search and selecting models with the lowest dev loss. We show a sampling of quantitative and qualitative results in figure 2, but for further examples of our model's generated recipes please see appendix figure 4.

5.2.1 Quantitative

	PP (Train)	PP (Dev)	BLEU (Dev)
Vanilla Seq2Seq	3.83	539.84	0.0614
Attention	**3.80**	892.52	**0.0812**
Dropout	4.42	**474.04**	0.0598
Attention + Dropout	4.02	736.35	0.0615

ingredients	model output	reference recipe
water rice	combine all ingredients in a small saucepan bring to a boil reduce heat and simmer for 10 minutes remove from heat and let cool and let cool	bring water to a boil in wok on high setting stir in rice and cover reduce heat to medium and simmer 15 minutes or until all water is absorbed turn off heat and allow rice to sit 5 minutes fluff with a fork and serve

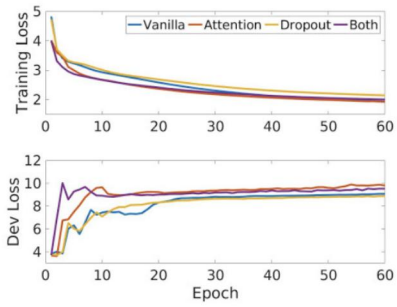

Figure 2: Results

From a quantitative standpoint, we evaluate two of the most common metrics for natural language processing models: perplexity and BLEU score. We compute perplexity as $PP = 2^{-\sum_x p(x) \log_2 p(x)}$. BLEU score is essentially a proportion of how many words that appear in the predicted caption also appear in the true label. We use BLEU-2, which measures the BLEU score of bigrams.

Quantitatively, judging by figure 2 it would appear that the attention model is the best of the four that we tested. Not only does the model converge the most quickly, but it also maintains the lowest scores in terms of perplexity and BLEU score. Though the attention model performs very well in a relative sense, from an absolute standpoint none of the models are particularly strong based on the specified metrics.

5.2.2 Qualitative

Though the model doesn't perform strongly when evaluated by the traditional metrics, from a qualitative standpoint, our model appears to generalize quite well. Importantly, the model demonstrates categorical inference: that is, when the list of ingredients contains any kind of alcohol, the model is able to infer that the recipe must be for a mixed drink and produces items like collins' glasses, champagne flutes, and cocktail shakers. When the list of ingredients contains items used frequently in baking, the model suggests using baking trays, ovens, and other baking related paraphernalia. The model is also good at determining when it needs to specify a time (put it on low heat for x minutes, simmer for y minutes), but performs less effectively at determining a realistic quantity of time to predict (i.e. predicts 600 minutes instead of 6).

6 Conclusion

We present an end to end seq2seq model that takes as input a set of ingredients and produces a series of instructions that dictate how the pre-specified ingredients should be combined. When attention is applied, the model performs very well and is able to successfully achieve categorical inference - that is, it has a semantic understanding of when a particular set of ingredients is to be used in baking or preparing mixed drinks and produces a coherent recipe accordingly.

Further work would include continuing to help the model better generalize. Having already tried dropout, attention, and pretrained word embeddings, the next steps might include trying different RNN cell types (GRU or vanilla RNN) along with potentially using bidirectionality for the encoder which has been shown to improve model performance on other tasks. A more radical approach might be to redesign the architecture altogether to incorporate some kind of dependency parsing to better make use of the relationships between the ingredients and the instructions of particular recipes. Most importantly, however, is to note that given that perplexity and BLEU score were not really indicative of how well our model performs on the data, new metrics for coherence and effectiveness of generated natural language need to be devised.

7 Contributions

We did almost all of the work for this project together. We designed our model architecture together and both worked on parts of the data pre-processing scripts. We each took the lead on implementing one of the two full models we tried but jointly added features to both. We jointly worked on the writeup and poster. Dev did the test cooking. Thomas did the test eating.

8 Link to Code Repository

`https://github.com/thomasteisberg/recipe-disaster`

We have shared this repository with `guillaumegenthial`. Please let us know if we should add others.

9 Acknowledgements

We would like to thank Guillaume and the rest of the CS230 teaching staff for their guidance and Amazon Web Services for computing resources.

10 References

[1] Andrej Karpathy. *Multi-layer Recurrent Neural Networks (LSTM, GRU, RNN) for character-level language models*. 2015. URL: `https://github.com/karpathy/char-rnn` (visited on 04/30/2016).

[2] Caiming Xiong, Victor Zhong, and Richard Socher. "Dynamic Coattention Networks For Question Answering". In: *CoRR* abs/1611.01604 (2016). arXiv: `1611.01604`. URL: `http://arxiv.org/abs/1611.01604`.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2014). arXiv: `1409.0473`. URL: `http://arxiv.org/abs/1409.0473`.

[4] Tomas Mikolov et al. "Recurrent neural network based language model." In: *INTERSPEECH*. Ed. by Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura. ISCA, 2010, pp. 1045–1048. URL: `http://dblp.uni-trier.de/db/conf/interspeech/interspeech2010.html#MikolovKBCK10`.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112. URL: `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

[6] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025 (2015). arXiv: `1508.04025`. URL: `http://arxiv.org/abs/1508.04025`.

[7] Kristian J Hammond. "Learning to anticipate and avoid planning problems through the explanation of failures". In: *Intelligence* 5.7 (1986), p. 8.

[8] Mirella Lapata and Regina Barzilay. "Automatic evaluation of text coherence: Models and representations". In: *IJCAI*. Vol. 5. 2005, pp. 1085–1090.

[9] Shinsuke Mori et al. "FlowGraph2Text: Automatic sentence skeleton compilation for procedural text generation". In: *Proceedings of the 8th International Natural Language Generation Conference (INLG)*. 2014, pp. 118–122.

[10] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. "Globally coherent text generation with neural checklist models". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 329–339.

[11] Amaia Salvador et al. "Learning Cross-modal Embeddings for Cooking Recipes and Food Images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[12] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. “Neural Machine Translation (seq2seq) Tutorial”. In: *https://github.com/tensorflow/nmt* (2017).

[13] Udacity. “Deep Learning Nanodegree Foundation”. In: *https://github.com/udacity/deep-learning/tree/master/seq2seq* (2018).

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[15] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

11 Appendix

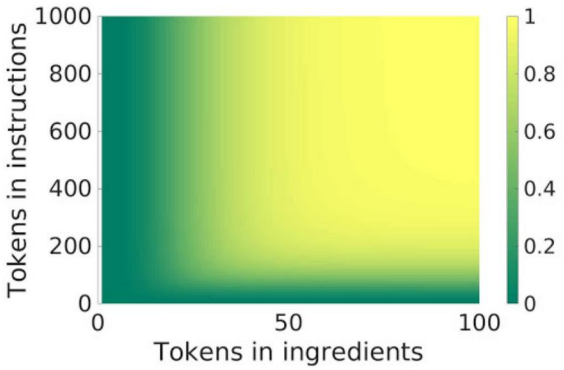

Figure 3: Recipe 1M Coverage

ingredients	model output	reference recipe
self rising flour milk mayonnaise dried tarragon garlic powder	mix all ingredients together and roll into balls and bake at 350 degrees for 20 minutes	preheat oven to 425 degrees coat 12 muffin cups with nonstick cooking spray in a bowl combine all ingredients mix well divide batter equally amongst muffin cups bake 12 to 15 minutes or until golden serve warm
water rice	combine all ingredients in a small saucepan bring to a boil reduce heat and simmer for 10 minutes remove from heat and let cool and let cool	bring water to a boil in wok on high setting stir in rice and cover reduce heat to medium and simmer 15 minutes or until all water is absorbed turn off heat and allow rice to sit 5 minutes fluff with a fork and serve
raspberry lime juice sparkling ginger ale ice crushed	pour all ingredients in a blender and blend until smooth	blend all ingredients on high until frothy
vanilla vodka sour apple liqueur	pour the vodka and vodka in a cocktail shaker filled with ice shake vigorously strain into a chilled cocktail glass garnish with a lime wedge	shake with ice and strain into a chilled cocktail glass
pomegranate juice chilled prosecco sparkling wine chilled	pour the grapefruit juice into a champagne flute and stir well	pour pomegranate juice into champagne flutes dividing equally and add chilled wine over the fruit juice
butter all-purpose flour cheddar cheese cayenne pepper	mix all ingredients together and chill for 1 hour or until golden brown	cheese and make sure all 3 ingrediants are blended well together shape into marble sized balls place on ungreased baking sheet bake at 350 degrees for 10 minutes or until done

Figure 4: Model Generated Recipes