Word-Based Reccurent Neural Network Cooking Recipe Generator

TDDE16 - Text Mining

Emil Bråkenhielm (emibr678)

August 28, 2021

This report implements and investigates how a word-based RNN (Recurrent Neural Network) could be used to generate cooking recipes. The idea is since it is a word-based RNN, and not character-based, the model should be able to produce meaningful recipes with contextual sentences. A model was trained using a dataset of approximately 90 000 recipes containing three sections: "Title", "Ingredients" and "Instructions".

An evaluation metric was developed to measure the quality of the generated recipes. A score was given between 0-1. The recipes were evaluated on three main areas: if they contain all three sections, if ingredients found in the Title exists in the Ingredients, and lastly if ingredients found in the Instructions exist in the Ingredients section. A parameter was used when generating the recipes, which would dictate how predictable a generated recipe should be. In the final evaluation, 4 different parameter values were compared to each other. The generated recipes produced a score between 0.48-0.68 (depending on the parameter). In conclusion, the model was able to produce some good recipes, while most of them probably would not be suitable for actual cooking.

Contents

1 Introduction

Recurrent Neural Networks (RNN) with its subclasses is widely used to make sequential predictions character by character. The technique could for example be used to generate new sequences of text, by training a language model on a given dataset. The problem with character-based RNNs is that, despite the need for having to learn to spell, text generated from such a model tends to be a bit contextual flawed. While character-based RNNs focuses on what the next character is, word-based RNNs will learn to predict the next word in a sequence, thus providing better prerequisites to for generated text to make more sense.

1.1 Aim

The aim of this project is to implement a word-based RNN to generate new cooking recipes, by training an RNN on a dataset of approximately 90 000 recipes. Also a suitable metric should be developed to measure the quality of the generated recipes.

1.2 Problem description

1. Generate recipes using word-based RNN
2. Evaluate performance

2 Theory

2.1 Preprocess

Preprocessing could be defined as a preliminary processing of data into an understandable format. Working with collected data usually requires cleaning of faulty data, and preparing the data for the training model.

2.1.1 Tokenization

Tokenization is the process of seperating meaningful data into smaller parts called tokens (5). In Natural Language Processing (NLP), tokenization is used to seperate texts to word, characters or subwords by a predifined delimiter (*Figure 1*). These tokens forms blocks of Natural Language.

Text: "This is a sentence."

Tokens: "This", "is", "a", "sentence", "."

Figure 1: Tokenization of text into words.

The open-source python library Gensim provides an in-built tokenization function specifically for NLP[1].

2.1.2 Text Vectorization

Text vectorization is the process of transforming text into numeric representatives. After applying tokenization to a text, each word are mapped to an integer as illustrated in *Figure 2*. Most deep learning algorithms requires numeric feature vectors as input, so called Word Embeddings (7).

Tokenized Text:	[This,	is,	a,	sentence,	.]
Vectorized Text:	[20,	35,	16,	5,	10]

Figure 2: Vectorization of tokens into numeric representatives.

2.1.3 Padding

Padding is a technique used to pre-process sequential data of different length, into a user-defined size. Considering the vectorized sentence in *Figure 3*, which contains 5 tokens. Padding could be used to increase the size of the vector. As an example *Figure 3* illustrates the text being padded to size 10, where '¤' in this case is a special symbol corresponding to the vectorized value 0 in the vocabulary.

[1] https://tedboy.github.io/nlps/generated/generated/gensim.utils.tokenize.html

Tokenized Text:	[This ,	is,	a,	sentence,	.,
	¤,	¤,	¤,	¤,	¤]
Vectorized Text:	[20,	35,	16,	5,	10,
	0,	0,	0,	0,	0]

Figure 3: Tokenization of text into words.

Texts larger than a defined maximum length could also be truncated, meaning that elements are instead removed to meet the size requirements.

2.2 Neural Network

A Neural Network (NN) consists of nodes, and connections between these nodes. It is modeled on the human brain to mimic the system of organic neurons and synapses (6). In simplified terms, neurons can be refereed to as nodes and synapses as the connections between the nodes. Each connection is also asserted with a certain weight in the hidden layer (see *Figure 4*), which represents the influce an input has on the output.

An example of a simple NN is the Feed-Forward NN illustrated in *Figure 4*. The information in a feed-forward NN only moves in one direction- from the input layer, through the hidden layer and finally to the output layer (2).

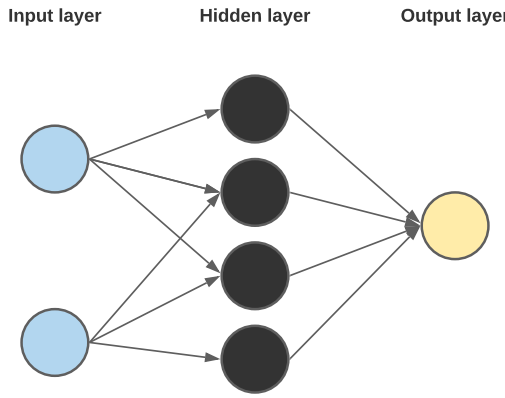

Figure 4: A Fast-forward neural network with 1 hidden layer.

2.2.1 Recurrent Neural Network

Recurrent Neural Network (RNN) are a powerful neural network used in modeling of sequence data. Due to an internal memory, RNN's is able to remember input received to make precise sequential predictions. Sequential data could be for example text, where the order of the words is important for an appropriate sentence construction.

The internal memory is what characterizes an RNN from other NN's. For example, the information in the Feed-forward NN is unidirectional due to its lack of memory. This means that the information will move through the whole network without considering any previous output. *Figure 5* illustrates the difference between the RNN and Feed-forward NN, where the RNN in addition to the hidden layer uses a feedback loop to store information from previous input (2).

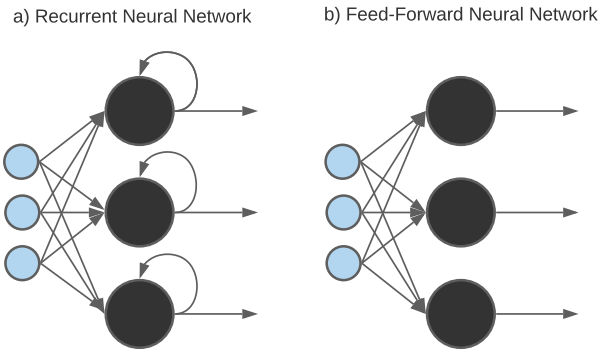

Figure 5: A Fast-forward neural network with 1 hidden layer.

2.2.2 Long short-term Memory

Theoretically RNN is able to handle long-term dependencies due to its internal memory, but it struggles when trying to make predictions that are contextually more complex. For example, consider a language model that predicts the next word based on a given sequence of words. A simple prediction could be "the money is in the *bank*", where no further context is needed to predict that the word is (most likely) going to be "bank".

If a language model were to predict the last word in "Today is thursday... Tomorrow is *friday*", the model needs more context to make a qualified prediction. The sentence "Tomorrow is..." suggests that the next word might be a weekday. However the model needs the context from further back, which in this case is that "today is thursday". If the distance from the context of "thursday" is too far away from the word of prediction, it will be too heavy for the RNN to handle. This is where a special type of RNN called Long short-term Memory (LSTM) is useful to cope with this problem.

LSTM is capable of remember information for a long period of time. Instead of a single NN layer to make predictions, LSTM has an additional three layers called gates. The gates are called: Forget gate, Input gate and Output gate, which regulates which information is let through. (8)

2.2.3 Optimizer

Optimizers is something used when training Neural Networks. It is an algorithm that changes attributes such as weight and learning rate to reduce the loss.

There are different types of optimizers, the most common is the Gradient Descent. It is dependent on the first-order derivative of a loss function. The idea is to reach minima by computing how the weights should be changed while training a model.

While classic gradient descent performs computations on the whole dataset, there are other more efficient methods. For example, the Stochastic Gradient Descent (SGD), which only computes on a small subset of the data. This type of optimizer tries to more frequently update the model's parameters. After the loss has been computed on each training example, the parameters are updated. The advantage of this type of optimizer is that due to its frequent parameter update, it helps the training model to converge quicker than the standard Gradient Descent(3).

An alternative optimizer is the widely used Adam Optimizer, which builds on the SGD. The main difference between classic SGD and Adam is that while SGD uses a single learning rate for all weight updates, Adam adapts the learning rate in real-time (4).

2.2.4 Loss Function

When training neural networks using optimizers it is required to choose a loss function in the model configuration. A loss function is used to evaluate a set of weights. It measures the quality of a prediction model, i.e. how good the model is to predict the expected outcome. Easily put, a loss function learns by its mistakes. If predictions deviate too much from the actual results, the loss function would return a very large number. Gradually the loss function learns to reduce the error in prediction (1).

3 Data

Two datasets are used in this work. One that contains recipes and one that contains ingredients. The recipe dataset is the one that will be used to train the RNN model, while the ingredients dataset will be used in the implementation of the evaluation metric. The Below sections gives an overview of the content. Preprocessing of the datasets will be covered in **4. Method**.

3.1 Recipe Dataset

The recipe dataset typically contains 5 sections: *Title*, *Ingredients*, *Instructions*, *Source*, *Photograph* as seen below. Both the dataset and the picture in *Figure 1* are taken from[2]:

- **Title:** Guacamole
- **Ingredients:**
  - 3 Haas avocados, halved, seeded and peeled
  - 1 lime, juiced
  - 1/2 teaspoon kosher salt
  - 1/2 teaspoon ground cumin
  - 1/2 teaspoon cayenne
  - 1/2 medium onion, diced
  - 1/2 jalapeño pepper, seeded and minced
  - 2 Roma tomatoes, seeded and diced
  - 1 tablespoon chopped cilantro
  - 1 clove garlic, minced
- **Instructions:** In a large bowl place the scooped avocado pulp and lime juice, toss to coat. Drain, and reserve the lime juice, after all of the avocados have been coated. Using a potato masher add the salt, cumin, and cayenne and mash. Then, fold in the onions, tomatoes, cilantro, and garlic. Add 1 tablespoon of the reserved lime juice. Let sit at room temperature for 1 hour and then serve.
- **Source**[2]: http://www.foodnetwork.com/recipes/alton-brown/guacamole-recipe
- **Picture:**

Figure 6: A recipe example from the dataset.

3.2 Ingredients Dataset

The ingredients dataset[3] contains 3 500 cleaned ingredients. It is simply structured as a list of ingredients.

Example ingredients:

[“butter”, “cocoa”, “eggs”, “flour”, “sugar”, “whitesugar”]
[“basilleaves”, “focaccia”, “leaves”, “mozzarella”, “pesto”, “plumtomatoes”, “sandwiches”]
[“babyspinach”, “blackpepper”, “cheese”, “eggs”, “fetacheese”, “grapetomatoes”, “kosher”]

Figure 7: Example of ingredients in the dataset. Credit: Dominik Schmidt.

[2]https://eightportions.com/datasets/Recipes/fn:1
[3]https://dominikschmidt.xyz/simplified-recipes-1M/

4 Method

The work will be implemented in steps described in *Figure 8*. The first step is to preprocess the recipe dataset and turn it into a uniform and understandable format, as well as clean it from faulty data. With the preprocessed data, the next step is to train the actual model. The recipes are at this stage ready to be generated and should then be tested in the evaluation.

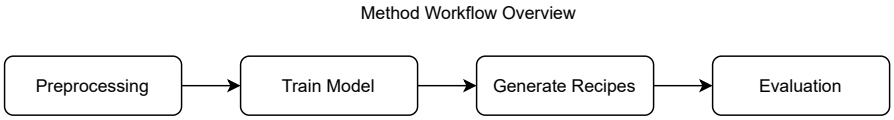

Figure 8: Workflow of the implementation structure.

4.1 Preprocessing - Recipe Dataset

The preprocessing will be divided into three main steps before creating the model (see *Figure 9*).

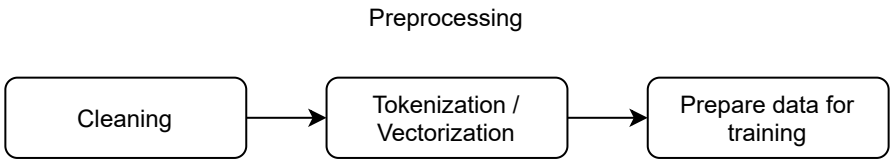

Figure 9: Workflow of the preprocessing.

4.1.1 Cleaning

First of all, the dataset needs to be cleaned from misleading data, and be transformed into a uniform format. Each recipe in the dataset should contain exactly three categories: "title", ingredients" and "instructions". Initially the recipes contain additionally 2 sections: "Source" and "Photo" which should be removed (see chapter (3. Data) for an overview). Some recipes, however, also contains double instructions, some contains NaN values and thus should be initially discarded.

The next step is to look at each category of the recipes and identify things that might affect the training process. For example, the occurrences of specific ingredients brands are often shown in the ingredients list followed by a trademark symbol ("™", "©" and "®"). As the dataset is based on American brands, these types of recipes are removed to make the generated recipes more general.

Another thing that is frequent in the recipes is double units e.g. "*2 (10 ounces) cans of chicken*". This might make it hard for the model to learn the appropriate unit and

measurements for certain ingredients and is thus removed. Lastly, credit to the photographer for the pictures found in the original "picture" category should be removed. This should be done by simply remove all sentences beginning with "Photograph by...".

The length of each recipe varies throughout the dataset, measured in the number of words. To make the data more uniform and to reduce the size of the dataset (due to hardware limitations) outliers should be removed. Outliers in this case would be recipes with an infrequent number of words, e.g. very short or very long recipes. By plotting the frequency versus the number of words in each recipe, a minimum and maximum length could be identified to filter the dataset.

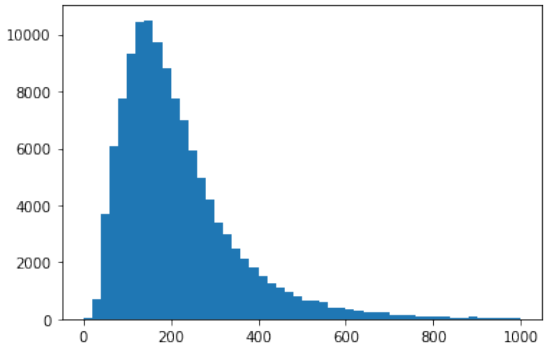

Figure 10: Illustrates the frequency (y-axis) of recipes with given lenght (x-axis)

Since RNN requires a fixed batch size when training a model, the dataset should be shrunken so that the selected batch size is a proper division of the total number of recipes in the dataset.

4.1.2 Tokenization and Vectorization

When the dataset is completely cleaned the recipes should be converted into a vectorized format. Each word in the data set should firstly be tokenized. Then each word should be mapped to its corresponding numeric representatives. A method to translate vectorized data back into a uniform sting formatting should also be implemented and tested at this stage.

As the length of the recipes differs between the minimum and maximum values defined in the cleaning process, padding should be applied to each recipe. The padding will be mapped to the number 0, which in the translation back into words should be ignored.

4.1.3 Prepare data for training

At this stage the dataset should be converted into a Tensor Dataset, which is compatible with the RNN model. To help the model predict words in a sequence, input and target texts should be defined and saved as the final training set. Lastly, the training set should be shuffled, and divided into chunks defined by the batch size.

4.2 Model Training

With the dataset cleaned and preprocessed it is time to define some parameters before and build the model before the actual training. The following parameters will be used to define the model:

- Model type: Sequential
- Layers: Embedding, GRU, Dense
- Batch Size: 64
- Embedding Dimension: 256
- RNN Units: 512

The model should then be compiled using the Categorical Crossentropy loss function, and the Adam Optimizer function with the learning rate set to 0.001.

The model is now ready to be trained and should run 30 epochs. To avoid overfitting an early stopping callback will be used in the training process.

4.3 Generating Recipes

Recipes should be generated using the trained model. Recipes are then generated with a given start sequence e.g. ["Title", "Chicken"], but to limit the scope of the evaluation, recipes will be randomly generated with only "Title" as the start sequence. A temperature parameter (henceforth referred to as the Temp-parameter) will also be used to define how unexpected the generated recipe is going to be. For example, a low value would generate more predictable recipes.

4.4 Evaluation

There exists metrics for evaluating text generation models, but it usually requires both candidate and reference to score the resemblance between the two. Since the idea of this project is to generate new recipes, the aim is not to be as similar to a reference as possible. Instead, a few criterion's are identified which will help to develop a suitable evaluation metric specifically for generating recipes.

Critera 1: A recipe should contain three sections: *Title*, *Ingredients*, *Instructions*. Lacking one of the sections should give a higher penalty than having duplicate or additional sections.

- **Score 1:** For every section that is found, 1 point is given. To penalize duplicate sections, the total points should be at most 3. The total points are then divided by the number of identified sections. If the recipe lacks one of the sections the returned score is instantly set to 0.

Criteria 2: Ingredients found in the *Title* should also exist in the *Ingredients*. For example, a recipe with the title "Slow Cooked Chicken" are assumed to have "Chicken" as one of the ingredients. Titles that do not contain ingredients (e.g. "New Years Dinner"), are a bit too complex for this Evaluation metric and are therefore given a pass.

- **Score 2:** Every ingredient that fulfills Criteria 2 generates 1 point. Total points are then divided by the number of identified ingredients in the title. If no ingredients are identified in the title, the score is set to 1.

Criteria 3: Ingredients mentioned in the *Instructions* should also exist in the *Ingredients*.

- **Score 3:** This score is calculated the same way as Score 2, only difference is that *Instructions* are compared with the *Ingredients*.

Final Score: The final score is the mean value of the three scores above. However, some exceptions are made due to limitations in the evaluation metric. If Score 1 = 0, then Score 2 and Score 3 are discarded and the Final Score is set to 0. If Score 1 ¿ 0, but the number of identified sections is larger than 3, Score 2 and 3 are also discarded.

Calculation Example:

Title: **Blueberry** Pancakes with Melted **Chocolate**

Ingredients:
- 1 cup flour
- 1 egg
- 1 teaspoon salt
- 3 tablespoons butter
- 1 1/2 cups milk
- 1 cup blueberries

Instructions:
- Swift together **flour**, **salt** and **sugar**. Pour **milk**, **egg**, **butter** and **chocolate**.

Criteria 1: Title=1, Ingredients=1, Instruction=1 = 3/3 = 1
Criteria 2: Blueberry=1, Chocolate=0 = 1/2 = 0.5

Criteria 3: Flour=1, Salt=1, Sugar=0, Milk=1, Egg=1, Butter=1, Chocolate=1 = 6/7 = 0.85

Total score: (1 + 0.5 + 0.85) / 3 = 0.78

Recipes will be generated using different temperature parameters, which then will be tested on the evaluation metric presented in the result.

4.4.1 Ingredients Dataset

Though the ingredients dataset was already cleaned, it needs further preprocessing to fit this project.

The first step is to keep only the base ingredients, for example, keep only "chicken" and not "boneless chicken". The dataset also contains a mixture of singular and plural forms of the ingredients, so each ingredient should be converted into its singular form to make it more uniform.

A test run of the evaluation at this stage should be made, identify the most frequent ingredients found in the evaluation. Two types of ingredients should be identified, let us call them Positive and Negative Ingredients:

- Negative Ingredients = Ingredients found in the title but NOT among the ingredients
- Positive Ingredients = Ingredients found in the title and among the ingredients

By finding the most frequent words in both categories, the dataset should be cleaned further. This remaining part of the cleaning will be done manually. The main idea is to remove collective names such as "Pasta", which would give a lower score if the *Ingredients* section only contained a specific type of pasta (e.g. "Spaghetti"). Also, the result of a recipe is sometimes occurring in the title, such as "Hot Soup". A human would have a decent idea of what ingredients to expect in a soup e.g. "water", "broth" etc. But automatically identifying these types of words would require a much more complex evaluation metric, thus the most frequent words of this type are identified manually and removed from the dataset.

5 Results

The result of the project is presented in this section. Initially the evaluation metric was tested on the training set which produced a Final Score of 0.92 (See *figure11*). Score 2 3 was a bit lower at 0.89 respectively 0.86.

Score - Training Data	
	Training Data
Sample Size	90000
Score 1	1.0
Score 2	0.89
Score 3	0.86
Final Score	**0.92**

Figure 11: Frequency on the y-axis and the number of words on the x-axis.

For the generated recipes, the scores are presented in *figure 12*. A lower Temp-parameter seems to produce a higher Final Score, where Temp=0.6 gave a Final Score of 0.68. However, Temp=0.6 only produced the third best with Score 1. The sample size of type of each generated recipe was 1000.

Score - Generated Recipes				
Temp	0.6	1.0	1.2	1.4
Sample Size	1000	1000	1000	1000
Score 1	0.88	0.91	0.89	0.82
Score 2	0.63	0.59	0.52	0.42
Score 3	0.54	0.39	0.3	0.2
Final Score	**0.68**	**0.63**	**0.57**	**0.48**

Figure 12: Frequency on the y-axis and the number of words on the x-axis.

5.1 Example Recipes

Following figures shows example of generated recipes with different Temp-parameter.

Example Recipe - Temp: 0.6

Title spinach artichoke dip

Ingredients
• 1 package frozen artichoke hearts, squabs
• 1 can artichoke hearts, maple-garlic marinated artichoke liquid reserved
• 1 can artichoke hearts, mozerrella over artichokes, 'lady strings
• 1 cup frozen artichokes, easily-and drained
• 1/4 cup fresh lemon juice
• 1 tablespoon chopped fresh parsley leaves

Instructions
▪ Preheat the oven to 375 degrees f.
▪ In a medium sized bowl mix the artichokes, dissolved— the mixture together. In a small bowl combine the artichokes, oro. Place the artichoke leaves on the kale leaves. Dip the artichoke bottoms in the mixture and place on a baking sheet. Surround with the artichoke hearts and artichokes, coconuts, 8x4-inch. Bake for 1 hour or until the spinach is tender. Remove the artichoke slices

Figure 13: Example recipe with temperature 0.6.

Example Recipe - Temp: 1.0

Title cheesy rice breakfast casserole

Ingredients
• 2 cups uncooked short-grain white rice
• 2 cups white rice
• 1/2 cup chicken stock
• 2 cups water
• 1/2 cup vegetable oil
• 1 tablespoon salt
• 1/2 teaspoon cracked black pepper
• 1/4 cup onion

Instructions
▪ Preheat the oven to 350 degrees f.
▪ Heat 2 tablespoons oil in a large skillet over medium heat. Add the onion and cook until lightly browned, florettes, target in. Add the onions, hugs® white bits from the sides of the onion, costeño, granules, china cap, kiawe-grilled, cookery style with poultry seasoning, spongecake and add salt and pepper to taste. Allow the excess onion from the heat for 1 minute. Fold in the ground chuck, cavity and season with

Figure 14: Example recipe with temperature 1.0.

Example Recipe - Temp: 1.2

Title velveeta cheesy cheeseburger dip

Ingredients
• 2 pounds cream cheese, the3 ketchup, drexler in dairy milk
• 3 cups mayonnaise, conversely thicken in milk
• 15 tablespoons hot water
• 1 teaspoon worcestershire sauce
• 1/4 teaspoon black pepper
• 1 slice red pepper

Instructions
▪ Preheat it to 325 degrees f. arrange the rice triangles in one layer in a large bowl. Beat the evaporated milk and cheese with an electric hand mixer until most of the texture is light and uniform. Sprinkle in the flour, 8.75 of cheddar and scrape out any rubber taco mix. Pour the sauce over the onions, pan- slices like shredded season, panna cotta with paprika, manhattan and bowls evenly over.
▪ Top with the other ingredients and coat with the herb mixture

Figure 15: Example recipe with temperature 1.2.

Example Recipe - Temp: 1.4

Title lobster nachos

Ingredients
• 1 stick canned chipotle chile
• 2 large ears yellow corn
• 5 cups caribbean cooked lobster meat, nougat, rum/gelatin red 's hot pad liquid
• 1 pint whipping cream
• 1/4 cup chopped cilantro, opening in a chilled 6 chipotle fritters

Instructions
▪ Goya soup with a small pot of southwestern enchilada or water, sort of new mac and then in a porridge stir as medium-high. Serve 2/3 cup per plate for garnish sprinkle of paprika while lay out the bowls of mush. Divide up so half to go past a 12-ounce container at home in serve., swallow meat, grape-tinted toasted for rolling clotted cream
• Strong espresso or coffee tin preparing the recipe preferred layers spice at room temperature
• Salt
• 1 cup granulated

Figure 16: Example recipe with temperature 1.4.

6 Discussion

The generated recipes produced a Final Score between 0.48-0.68 with a lower Temp-parameter producing higher results. As the training data produced a score of 0.92, the result of the generated recipes was not too bad.

Looking simply at the score can be a bit misleading since there are much more factors that affect how good a recipe is. The example recipe with Temp=0.6 in figure 13 shows one possible reason as to why a lower Temp-parameter produced the best result. The Title contains the ingredient "artichoke", so it should also occur in the ingredients. The problem here is that the Ingredients section contains 5 occurrences of the word "artichoke", and the Instructions contains 7 occurrences. Though a higher number of occurrences does not affect the metric (either an ingredient exists or not), the chance of fulfilling Criteria 2 & 3 increases if an ingredient is overproduced. Thus, a higher chance of getting a full Score 2 & 3.

With a higher Temp-parameter, the recipes seem to be less repetitive and look more like regular recipes. For example Temp=1.2 in figure 15 almost resembles a real recipe. The title works with the ingredients, and though the instructions are a bit off, it contains some instructions and ingredients that somewhat follows the same theme.

It is hard to say that something is a pattern with only a few examples, which means that the above arguments might not be completely valid. But looking through the full datasets of generated recipes gave me a similar perception. It is of course much easier for a human brain to quickly say whether a recipe is good or not. We can easily see the common thread, which ingredients should be included in a Mediterranean recipe, but it is hard to translate this into an evaluation metric.

6.1 Sources of Error

As seen in figure 12, the sample size of each set of generated recipes was only at 1000. Using a training data of 90 000 recipes, a test data of 1000 would typically be seen as way too small. A too-small dataset is prone to randomness, and may not give fair scoring between the different Temp values. I was planning on producing at least 10 000 recipes of each Temp-parameter, but generating the recipes was much slower than I had anticipated. 1000 recipes took nearly 2 hours to produce. Though I tried to run the evaluation on random samples of 1000, 10 000 and 50 000 from the training data, which still gave a Final Score of 0.90-0.92, so perhaps the error is not too high.

Recipes could have been generated with a start sequence containing a fixed ingredient. This could perhaps produce more accurate recipes in terms of the different scores. Currently, the data in the results are produced completely random using only "Title" as the start word.

To give a more fair evaluation score, the training data could have been cleaned from recipes that initially did not fulfil any of the evaluation criteria. Also, recipes that had no identified ingredient in the title could have been discarded before training the model.

7 Conclusion

Most recipes might not be suitable for an actual dinner, but in some cases, the generated recipes were good. The evaluation metric used, showed that the generated recipes had some troubles following a common thread of the Ingredients section containing ingredients from the Title, and so on. Though a majority of all generated recipes contained the correct structure. Despite that many instructions does not make sense, or are out of context, they often contain full sentences. Only the fact that it produces a structure, and sentences, and sometimes even a common thread is an achievement in itself.

8 Future Work

Due to time constraints, I did not have the time to tune the parameters for the RNN model. I am sure that some tweaking could increase the quality of the generated recipes.

Some recipes still contain trademark and copyright symbols, and sometimes weird symbols occur which proves that the cleaning of the training set was not completely successful.

Lastly, the evaluation metric could be improved by designing a way to deal with ingredients with collective names. For example, if "Salad" were found in the Title, the algorithm could look through a specific list of ingredients that falls under the collective name salad (e.g. arugula, kale, lettuce etc.).

References

[1] Jason Brownlee. A gentle introduction to the rectified linear unit (relu). *Machine Learning Mastery*, 2021. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[2] N. Chowdhury and M. A. kashem. A comparative analysis of feed-forward neural network recurrent neural network to detect intrusion. In *2008 International Conference on Electrical and Computer Engineering*, pages 488–492, 2008. https://doi.org/10.1109/ICECE.2008.4769258 doi:10.1109/ICECE.2008.4769258.

[3] Sanket Doshi. Various optimization algorithms for training neural network. *Towards data science*. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Aravind Pai. What is tokenization — tokenization in nlp. *Analytics Vidhya*, 2020. URL: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/.

[6] Xiongwen Pang, Yanqiang Zhou, Pan Wang, Weiwei Lin, and Victor Chang. An innovative neural network approach for stock market prediction. *The Journal of Supercomputing*, 76, 03 2020. https://doi.org/10.1007/s11227-017-2228-y doi:10.1007/s11227-017-2228-y.

[7] Paritosh Pantola. Natural language processing: Text data vectorization. *Medium*.

[8] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation. *Towards Data Science*, 2018. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.