MLP Coursework 4: Deep Drinks

Group 36: s1407243, s1457693, s1432492

Abstract

This research investigates how deep neural networks can be applied to solve the problem of Natural Language Generation in the context of cocktail recipe generation. We build upon our previous work on the topic and introduce a new probabilistic recipe generation method. Our contributions also involve expanding our original dataset of cocktail recipes, and incorporating an ingredients list into our model predictions. Moreover, we experiment with GloVe as a word embedding to avoid existing data sparsity problems and improve the training process. Finally, we introduce a wholly new model for recipe name generation and an appropriate evaluation method for all our models using human input.

1. Introduction

The problem of Natural Language Generation (NLG) has been widely researched by machine learning experts and there is a large body of literature addressing the problem. It has proven useful in a diverse range of applications, such as natural language understanding (Graves, 2013), response generation in dialogue systems (Wen et al., 2015; Sordoni et al., 2015), text summarization (Rush et al., 2015), machine translation (Bahdanau et al., 2014) and image caption (Xu et al., 2015). Nevertheless, many of the existing approaches are domain-specific and cannot be easily adapted to solve a different text generation task.

In Coursework 3, we commenced the project 'Deep Drinks', in which we aimed to explore and apply state of the art models in NLG including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) to a novel and complex language generation task. Namely, cocktail recipe generation.

The aim of Coursework 4 is to investigate more advanced approaches for cocktail recipe generation associated with the original cocktail dataset we created. The baseline system built in Coursework 3 will be used as a reference.

We start by summarizing our baseline system results, along with our research questions and objectives in Section 1. Section 2 briefly discusses our data collection and preprocessing methods. Our methodology and theory behind relevant neural network models are presented in Section 3. We present and discuss our experiments in Section 4. In Section 5 we describe our model evaluation methods. Lastly, we finish the report with a review of related work in NLG in Section 6 and our conclusions in Section 7.

1.1. Baseline Model

In Coursework 3 we investigated the performance of RNNs as a language model for the task of cocktail recipe generation. We created an original dataset of cocktail recipe descriptions by scrapping websites, which we used to train our neural network models. We proposed two baseline models operating on character-level and word-level input, and we investigated the impact that different hyperparameters had on the performance of the model: vocabulary size, sequence length and number of epochs. Ultimately, we showed that both the character-level and the word-level model suffered from overfitting. However, the character-level model did not produce the results we were hoping for. Many grammatical errors were present in the output. Often, there were semantical errors, and the model got stuck in loops. The word-level model generated language that was more grammatically and semantically correct, and we decided to focus on improving it for this coursework.

The best performing word-level model consisted of a 2 Layer GRU network architecture with 256 and 128 hidden units, respectively, and 2 Dropout layers with p = 0.2. It achieved an accuracy of 0.4633 on the validation set, which seemed as a satisfactory trade-off between fitting to the training data and model generalization in our task. We will use this as baseline to compare our more advanced models for the rest of this coursework.

1.2. Research Questions

Similarly to Coursework 3, the research question we are exploring is whether neural networks can learn the dependencies between words and the conditional probabilities of words in sequences so that we can generate new and original cocktail recipes. A more general problem we are trying to address is whether RNNs can learn high-quality language models.

1.3. Objectives

In this project, we aim to achieve the following:

- Introduce ingredients and cocktail names to our existing recipe dataset.
- Increase the size of our dataset by scrapping 3 more websites, thus increasing the number of training datapoints by at least half.
- Extend the current model to a probabilistic one, where

the output of the next element is sampled from a probability distribution defined by the output layer of the network.

- Use word embedding to represent each data point more efficiently, enabling us to increase the vocabulary size and sequence length.
- Build a non-recurrent name generating model, which generates a recipe name given the ingredients.
- Provide a comprehensive model evaluation using human input.

2. Data

NLG requires a large text corpus in order to generate sentences that are coherent and semantically correct. One of the main challenges that we encountered was a lack of such training datasets. We decided to create our own text corpus of cocktail recipes by scraping data from external websites.

2.1. Data Scraping

Web scraping refers to an automatic process of data extraction from websites using bots, also known as web crawlers. It involves accessing the HTML document of a webpage, which is represented internally as a tree of HTML elements, known as the Document Object Model (DOM). Web crawlers can extract data contained within individual HTML tags by traversing the DOM tree. We designed several web crawlers in order to collect the data.

For our problem, we found several websites that are focused on preparing cocktails. We chose websites which present cocktail recipes as free-text descriptions, possibly consisting of multiple sentences, but without any explicit enumeration of the steps. We also ensured that a list of ingredients was present alongside every recipe instruction.

In our previous work, we created a dataset of 1153 recipe instructions that were scraped from two websites (Social+Cocktail Events) (Liquor). This time, we decided to increase our text corpus by scraping an additional 2194 recipes from three more websites (All Recipes), (Live in Style), (Serious Eats) which brought our total sample count to 3347. Moreover, we also scraped ingredients and cocktail names from all five websites and included that information alongside every recipe description.

2.2. Data Preprocessing

Our models take sequences of predefined length as input and attempt to predict the next word in the sequence.

As a first step, for every cocktail recipe, we combined the list of ingredients and recipe instructions into one string, placing '#' as a delimiter. We then concatenated all resulting strings together, separating them with the symbol '|'. We made all characters lowercase and removed punctuation, apart from periods ('.'), which are used to denote sentence boundaries. Finally, we separated the string into distinct words by splitting on whitespace. In the end, we obtained 88425 words and we leave 5% of our data out to be used as the validation set. The number of unique words, i.e. the vocabulary size, was equal to 4465. Since our project involves generating natural language, normal model evaluation such as train/test accuracies provide little information about the performance of the model. Therefore we only create a validation set to control overfitting.

In order to use our data as input to neural networks, we had to perform data encoding. We first mapped each token in our dataset to a number from 0 to n, where n is the vocabulary size. In our previous work, we then applied one-hot encoding to each of the resulting integers. This approach was not scalable, however, as it required operating with massive sparse matrices, leading to out-of-memory errors. As we increased the number of datapoints and the vocabulary size, we decided to encode our input using word embeddings.

2.3. Word Embeddings

Word embedding refers to a set of language modeling and feature learning techniques in natural language processing. The goal of word embedding is to map words from a vocabulary to corresponding vectors of real numbers. For instance, one-hot encoding is a word embedding that maps each word to a binary vector with only one '1' and '0's in all other positions. This means that the dimension of each vector equals the vocabulary size, which often results in huge sparse vectors that are infeasible to fit in memory. One way to counteract this problem is by limiting the vocabulary size, for instance, by taking only N most common tokens.

Another approach is to learn a dense vector representation of words, with a much lower dimension than one-hot encoded vectors. There are several models that can be used to learn continuous vector representations. One of them is the word2vec model by (Mikolov et al., 2013), which has gained significant popularity in the recent years. In this project, we decided to use the GloVe algorithm (Pennington et al., 2014), which can be trained on aggregated global word-word co-occurrence statistics. GloVe learns vector representations that account for semantic similarity between words, so that words with similar meaning end up laying close to each other in vector space. As a consequence, vector arithmetics are defined, for instance king - man + woman = queen.

Instead of using GloVe algorithm to learn our own word embeddings, we decided to use publicly available pre-trained word vectors. In this work, we used vector representation of words obtained by training on the Common Crawl (Elbaz) dataset containing 42 billion tokens and a vocabulary size of 1.9 million. This embedding maps each word into a 300-dimensional continuous vector space and is applied as the first layer of our neural network models.

3. Methodology

In language modeling, the task is to model the probability of sequences of words (Sundermeyer et al., 2012). More precisely, for a sequence T of words w_1,...,w_T we define its probability as

$$P(w_1, ..., w_T) = \prod_{t=1}^{T} P(w_t \mid w_{1:(t-1)}) \quad (1)$$

where 1:$(t-1)$ denotes the sequences of indexes 1 up to $t-1$. Hence we hope our model will assign high probabilities to words sequences like `add vodka to glass` but low probabilities to sequences like `ice to to ice ice`.

We have trained three different word-level RNN language models on the expanded cocktails dataset described above. That is, we feed the RNN a large text containing cocktail recipes, and it models the distribution of the next word in the sequence given a sequence of previous words. This allows us to generate one word at a time as shown in Figure 1.

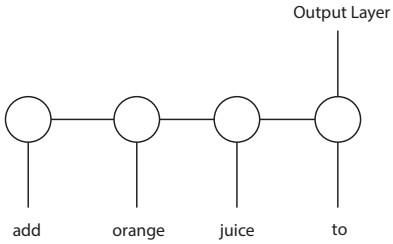


Figure 1. RNN model for generating new words from a sequence.

The output layer of this model is a dense layer with a softmax activation function. The number of neurons on this layer is equal to the size of the vocabulary. The output is a vector with a probability for each class (i.e. word) present in the vocabulary as shown in Figure 2.

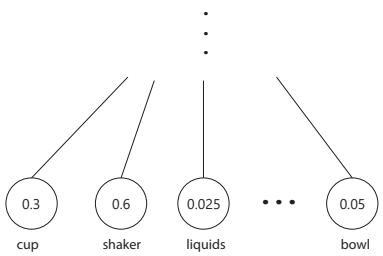


Figure 2. Output Layer of RNN model.

To make a prediction, we start with a random seed sequence as input. This is a sequence of words from the dataset of the selected sequence length. We generate the next word and then update the seed sequence to add the generated word on the end and trim off the first word. Then we repeat this process for as long as we want to predict new words.

3.1. Probabilistic prediction model

Often times our previous model got stuck in a loop and produced output such as "1 ounce freshly squeezed lime juice from 1 lemon 1 ounce freshly squeezed lime juice from 1 lemon". To tackle this issue we created a probabilistic prediction model. Each time we predict a new word, instead of choosing the most probable one, we choose one at random with the probability distribution given by the softmax function of the output layer. For instance, when predicting the next word of the sequence `add orange juice to`, the output could look something like:

[(shaker , 0.6) , (cup , 0.3) , (bowl , 0.05) , (ingredients , 0.025) , (liquids , 0.025)]

We choose one of these words with its respective probability. This strategy allows the model to avoid repetition loops, offer a more varied output and avoid overfitting.

Unfortunately, this strategy also entails that sometimes the model will output words that have very low probability and do not make sense in a certain context. For this reason, we introduce the Creativity hyperparameter - **c**. It represents the number of words we want to consider when making a prediction. Hence, we only take into account the c words with the highest likelihood and normalize their probabilities using the following formula:

$$f(x_i) = \frac{x_i}{\sum_j x_j}$$

where x is the probability of a word in the reduced set. In the example above, we can set c to any integer between 1 and 5, where $c = 1$ would represent a completely deterministic output and 5 would cover the entirety of the vocabulary. For instance, after setting $c = 3$ we get:

[(shaker , 0.631) , (cup , 0.315) , (bowl , 0.054)]

3.2. Name Generation model

In addition to the recurrent recipe generating model we created a wholly new language model to generate cocktail names from their ingredients. For this purpose, we built a fully connected feed-forward network architecture which takes a list of ingredients as an input and outputs the title of the recipe.

The first issue we encountered with this model is the size of the input and output layers. After applying label and one-hot encoding to each training example, we were left with an architecture that has 100,000 neurons in the input layer and 30,000 in the output. This made learning impractical, considering the size of our dataset.

To address this issue, firstly we create two separate label and one-hot encoders for the input and output data, and limit the vocabulary size to 1000 for each of them. We then remove numbers, brackets and measurement words (e.g. 'tablespoon', 'ml', 'ounces'...) from our vocabulary. Lastly, we restrict the input to 20 ingredients and the output to 5 title words. We pad shorter recipe titles with the special word 'null', which otherwise will not exist in the vocabulary. Due to these constraints, we remove around 500 recipe examples from our dataset.

Since we want to optimize multiple one-hot encoded vectors, we cannot use a softmax activation function for the

output layer in this model. Instead, we split the output as shown in Figure (3), and apply softmax to each of them.

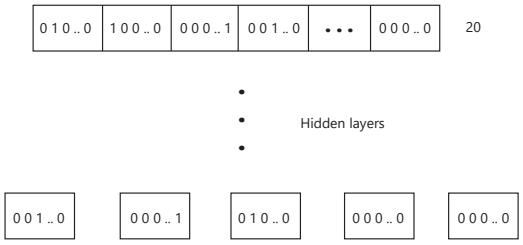


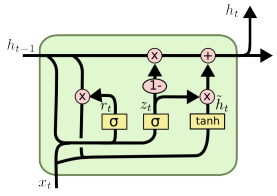
Figure 3. Name generation model.

3.3. Gated Recurrent Unit

Recurrent neural networks are a powerful model for sequential data but suffer from the exploding and vanishing gradient problem, which makes them difficult to train in practice. More precisely, when the gradient of the error function of the neural network is back-propagated through a unit, it gets scaled by a factor. This is often either greater or smaller than one. As a result, a sequence of derivatives less than 1 will vanish exponentially quickly with the length of the time lag, while a sequence of derivatives greater than 1 will cause the resultant gradient to explode.

Long Short Term Memory is a type of RNN architecture first introduced in 1997 (Hochreiter & Schmidhuber, 1997) to explicitly avoid the gradient problem. It is well established that the LSTM unit works well on sequence-based tasks with long-term dependencies (Chung et al., 2014), and they were explained in detail in Coursework 3.

We now put our focus on Gated Recurrent Unit, a variant of LSTM introduced by (Cho et al., 2014). Similar to LSTMs, GRUs retain the resistance to the vanishing gradient problem, however their internal structure is less complex. This makes them more computationally efficient, since fewer computations are needed to make updates to its hidden state. The gates for a GRU cell, and equations are illustrated in the following diagram:



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 4. Diagram of a Gated Recurrent Unit.
Source: (Olah, 2015)

Similar to LSTM, GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell. Nevertheless, the performance of the two units is comparable to each other (Chung et al., 2014).

3.4. Technical details

Our implementation uses only NumPy and Keras, a model-level library that relies on a tensor manipulation library on the background to deal with low-level operations. In this case, we have used it on top of a Tensorflow backend.

4. Experiments

We introduce a set of experiments for cocktail recipe generation on our improved models with a word-level GRU language model and a fully connected name generation model.

We have chosen the network architecture and hyperparameters based on our results from the baseline system. Unless otherwise stated, the values below describe the common network architecture and hyperparameters used in our models.

- **Batch size**: 100
- **Epochs**: 10
- **Activation function**: ReLU
- **Architecture**: 2 GRU layers with 256 and 128 neurons, respectively, and Dropout ($p = 0.2$)
- **Optimizer**: RMSProp

Each section below describes an improved model with respect to our baseline system, each addressing issues found in our previous work.

4.1. Probabilistic model without ingredients

The first improvement that we introduced to the baseline model was to use the output of the network as a probability distribution, rather than outputting the word with the highest numerical value. Different values of the c parameter were explored in order to find satisfactory trade-off between generating deterministic predictions and completely random ones.

Description
Our model architecture described above was trained for 10 epochs on the reduced dataset (only recipe descriptions). The vocabulary size was limited to 500 and sequence length set to 10.

After each epoch, training and validation accuracies were recorded and the model was used to generate recipes with c equal to 1, 10, 100, 350 and 500 (vocabulary size).

Results
At the end of epoch 10, the model achieved a training accuracy of 0.5294 and a validation accuracy of 0.5011. The model was still improving the validation accuracy throughout the last epoch.

Here is an example of a recipe generated by this model:

```
----------------------
1. - epoch: 10, c: 350
----------------------

Pour gin creme de cacao and orange juice
into cocktail shaker over ice. Cover and
shake until outside of shaker has frosted.
Strain into rocks glass filled with soda.
Garnish with ginger.
```

```
----------------------
```

Discussion
The inclusion of the c parameter in the baseline model, although minor, added a considerable improvement to the quality of generated recipes. The c parameter can be tuned to find the right balance between deterministic ($c = 1$) and overly random ($c = 700$) predictions.

Furthermore, validation accuracy was improving up to the last epoch and was insignificantly less that training accuracy. This meant that the model was not overfitting and could be used to generate original recipes that are not in the training data.

4.2. Probabilistic model with ingredients

The probabilistic model using the creativity parameter performed well and generated semantically and grammatically correct novel recipe descriptions. Unfortunately, often times descriptions such as "pour ingredients into mixing glass with ice cubes" require the knowledge of the recipe's ingredients.

Furthermore, we wanted to see the limitations of our model when dealing with more complicated, structured data. This is why we introduced a probabilistic model which incorporates ingredients into the cocktail generation.

Description
The model itself works similarly to the original probabilistic model. One of the differences comes at the preprocessing stage. We combined the list of ingredients separated by the word ';' and the description by placing '#' as a delimiter. From then on training continued as described in the previous section.

We also vary the creativity parameter c and test how it performs with values of 1, 10, 100, 350 and 700 (vocabulary). We expanded the vocabulary size to 700 to reflect the increase of words due to the introduction of ingredients. Furthermore, we increased the sequence length to 15 as we believe that it would help correlate the descriptions with the ingredients listed before them. Due to memory constraints, these were the maximum values we could work with.

Results
Figure 5 describes the training and validation accuracies during learning. The training and validation accuracy increase with each epoch until epoch number 7 where the validation accuracy levels off. Although we could achieve better accuracy, we know from the baseline experiments that these numbers tend to work well for the task at hand.

Shown below are two recipes generated using this model. These recipes were also part of the evaluation survey at the end.

```
----------------------
1. - epoch: 10, c: 700
----------------------
```

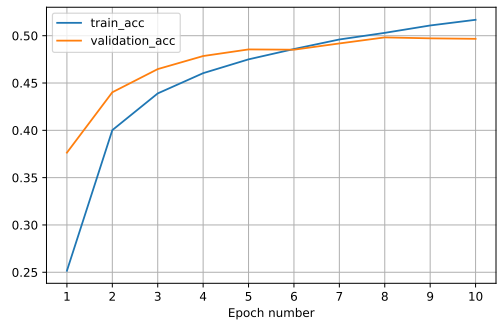


Figure 5. Training and validation accuracies for the probabilistic model with ingredients.

```
Vodka
White rum
Banana liqueur
Banana

Pour ingredients into cocktail shaker with
ice cubes. Shake well. Strain into chilled
cocktail glass. Serve.
----------------------
2. - epoch: 4, c: 10
----------------------

Vodka
Lime juice
Orange juice
Cranberry juice

Pour vodka and lime juice over ice in
highball glass. Add gin and bitters.
Stir to combine.

----------------------
```

Discussion
The generated recipes from this experiment were structurally, semantically and grammatically correct. As expected, due to the short sequence length and vocabulary size, many of the recipe descriptions did not match the ingredients listed before them. The second example given above illustrates this well. In the first sentence of the description, vodka and lime are mentioned and they are present in the ingredient list but then the non-existent gin and bitters are referenced.

While some training was necessary, good recipes were generated at various epochs, further supporting the hypothesis that training and validation accuracies are not necessarily the metric for evaluation when it comes to language model generation.

As demonstrated by the examples above, good recipes can be generated with different values of c. One exception is when $c = 1$ and the model reverts to our baseline model which, in turn, tends to get stuck in loops and repeats the same words and sentences.

4.3. Probabilistic model with embedding and ingredients

Adding ingredients alongside recipe descriptions significantly increased the number of unique words. As a result, one-hot encoded data became too large to fit in memory, forcing us to significantly limit the vocabulary size and the sequence length.

In order to improve the performance of the network, we decided to reduce the dimensionality of the input data by using dense vector representation of words. We also switched our loss function to sparse categorical crossentropy, which allows our labels to be stored as integers, instead of one-hot encoded vectors.

Description

We decided to replace binary word vectors with real-valued dense vectors. First we tranformed each word in the vocabulary to an index from 0 to $n-1$, where n is the vocabulary size. We then imported the GloVe pre-trained word embedding matrix, where row i corresponds to a 300-dimensional continuous vector representation of a word with index i. This dimensionality reduction is sufficient enough to allow all input data to fit in memory.

We could not apply the GloVe word embedding to the output labels, as our model would have to do regression on real-valued target labels, instead of classification. As a compromise, we changed our loss function to sparse categorical cross-entropy provided by TensorFlow and Keras. Sparse categorical cross-entropy accepts integer targets and performs one-hot encoding internally. This allowed us to store our word labels as integer indexes, improving computational speed and memory consumption.

As a consequence, we did not have to limit the vocabulary size of our training data and we performed experiments with sequence lengths of 30, 60 and 90. The models with these sequence lengths were trained for 10, 7 and 5 epochs, respectively, with longer-sequence models being trained less in order to avoid overfitting. After each epoch, the model was used to generate recipes with c parameter being equal to 1, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000 and 4466.

Results

The results for the three networks with different sequence lengths are presented in Table 1:

Length	Epochs	Train Accuracy	Valid Accuracy
30	10	0.4715	0.4719
60	7	0.4602	0.4697
90	5	0.4490	0.4577

Table 1. Different sequence lengths used for model input, number of epochs, training and validation accuracies achieved after the final epoch

For instance, these are two recipes produced by this model with different sequence lengths:

```
----------------------------------------------

1. - epoch: 9, sequence_length: 30, c: 500
----------------------------------------------

Tequila
Lemon juice
Grenadine syrup

Gently shake first three ingredients together
in mixing tin with ice and pour into chilled
cocktail glass.
----------------------------------------------
2. - epoch: 5, sequence_length: 60, c: 4000
----------------------------------------------

Gin
Fresh lemon juice
Simple syrup
Angostura bitters

Combine ingredients in mixing glass and fill
with ice. Stir well for 20 seconds and strain
into chilled cocktail glass. Garnish with
lemon twist.

----------------------
```

Figure 6 demonstrates the learning curve for validation set for three different models, plotted as a function of the number of epochs.

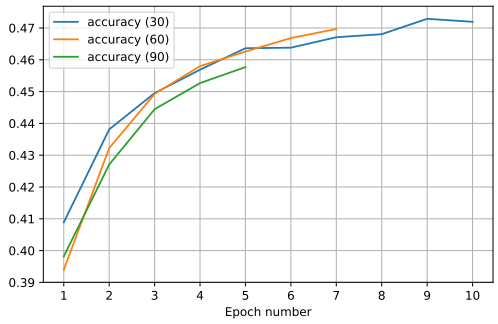


Figure 6. Learning curve for validation accuracies for models using sequence lengths of 30, 60 and 90.

Discussion

The recipes generated by the three models were, in general, grammatically and semantically correct. Longer sequence lengths also meant that the description for the recipe more likely contained the aforementioned ingredients, although there were still many cases when generated descriptions did not match given ingredients. In general, after training the models, the variability in the output between the three different models was not significant and they produced comparable results.

There was influence produced by the c parameter. By setting c to 1, the model essentially degraded to the baseline model that was deterministic and often stuck in loops. By setting c to 4466 (vocabulary size), the model occasionally

generated words that were out of context. Best results were achieved with intermediary values of c, between 500 and 4000.

4.4. Name Model

To complete our investigation of NLG, we decided to create a wholly new model that generates a cocktail's name given its ingredients. While we could have applied the same techniques and models as before, we decided to explore a different type of language generation model that might fit this task better.

Description
As described in Section 3.2, we use a fully connected feed-forward network with an input of up to 20 ingredients and output of up to 5 words. For this set of experiments, we kept the batch size equal to 100, activation function ReLU and Adam optimizer. We experimented with different number of epochs - 10, 50, 100, and depths of the network - 1 Layer of 128 neurons, 2 Layers with 128 and 64, 3 Layers with 128, 64 and 64 neurons.

Results and Discussion
Table 2 outlines the average validation accuracy over all separate outputs. We understand that averaging can be misleading because the model easily learns that there are rarely cocktail names longer than 2-3 words, and the 5th word would almost always be 'null', thus achieving high average accuracy.

EPOCHS	GRU 128	GRU 128,64	GRU 128,64,64
10	0.55	0.50	0.55
50	0.54	0.50	0.49
100	0.53	0.51	0.49

Table 2. Average validation accuracies for all depths and epochs.

As in previous experiments, we believe that accuracy is not a relevant metric in evaluating a language generation model. Results from this experiment further support this claim. Training with 10 epochs obtains the overall highest validation accuracy, however, the names it generates are usually a word long (e.g. `the` or `recipe`), and nonsensical.

The models trained for 50 epochs performed the best and produced names such as `tequila margarita sunrise` and `pinky lemon cocktail`, both of which do not exist in the training data. Furthermore, at times the names were related to the recipe ingredients, as it is the case for these two examples.

The models trained for 100 epochs generated very impressive names as `wild river bloody mary mix` and `creamy hazelnut martini`. Nonetheless, further inspection exposed that these are existing names of recipes in the dataset, and perhaps the model was overfitting.

The depth of the network had negligible effects on the output. The names generated for evaluation were created by training the GRU 128,64 model for 50 epochs.

5. Evaluation

In the previous coursework we failed to provide a comprehensive approach to model evaluation. This was complicated by the fact that the quality of generated natural language can be subjective, and we evaluated the outputted recipes using our own judgment.

In this coursework, in order to evaluate our proposed models, we have asked other people to read a series of recipes and rate the grammar and logic of them. We chose a total of 18 recipes, where 9 were randomly selected from the dataset (6 including ingredients, and 3 only including a description) and 9 were generated using the 3 models described above: probabilistic model without ingredients (3 recipes), probabilistic model with ingredients (3 recipes) and probabilistic model with both embedding and ingredients (3 recipes).

We received a total of 41 responses, each matching a random 6 recipe questionnaire. In each questionnaire, half of the recipes were real and half generated. The participants had no previous knowledge of the source of the recipes. We asked them to read each the recipe, and choose the number that best reflected their response to each statement, on a scale from 1 to 5, where 1 represented 'Strongly disagree' and 5 'Strongly agree'. For statement 6, we gave participants two options - "Yes" and "No". The statements were the following:

	Statement
1	The recipe is grammatically correct
2	The steps in the recipe appear in a logical order
3	The ingredients match the recipe steps
4	The title matches the recipe content
5	I would be able to replicate this recipe
6	I would drink this recipe

Table 3. Statements that participants were asked to rate on a 1 to 5 scale for each recipe in our evaluation survey

Finally participants were given a long-answer box where they could fill in additional comments. The results obtained are summarized in Table 4. They represent the average rating obtained by each type of sampled recipe in each of the statements that apply in Table 3.

Models used for sampling the recipes:
(A) Probabilistic model without ingredients
(B) Probabilistic model with ingredients
(C) Probabilistic model with embedding and ingredients
(D) Real recipes sampled from the dataset

Statement	A	B	C	D
1	3.56	4.39	4.15	4.27
2	3.93	4.71	4.36	4.33
3	na	3.49	4.29	4.30
4	na	3.37	3.09	4.29
5	3.41	4.02	4.32	4.08
6	53.66%	68.29%	82.93%	69.92%

Table 4. Average rating obtained by the recipes generated by each model, and percentage of people who agreed to statement 6.

We note that the average scores for Model (A) were lower than those for the rest of the models. The grammar rating and ability to replicate the recipe were particularly poor, possibly due to the lack of ingredients. Some participants observed that in the comments - 'Replication is complicated by the lack of information on proportions.'

Grammar and logic of steps performed well in models (B) and (C). In model (C), we see a great improvement in rating of statement 3 (i.e. ingredients match the recipe steps), perhaps due too the longer sequence length used in this model. Furthermore, we see an increase in agreement in the ability to replicate the recipes generated by this model.

For comments, on original recipes we saw observations such as as 'Nicely and logically written.'. Nevertheless, recipes generated using model (C) received similar comments: 'Easy to understand and follow'.

We observe that, altogether, both models (B) and (C) receive similar ratings to real recipes when it comes to grammar correctness, step logic, ingredient matching and ability of replication. However, title generation did not perform particularly well in any of these models. It appears that the title-ingredients logic is rather difficult to enforce using our name generation model?

6. Related Work

Natural Language Generation models can be roughly put into two categories. The classical one, based on rules or templates, and the recent approaches utilizing recurrent neural networks. In the classical version of the problem (Cheyer & Guzzoni, 2007), the rules and templates are designed by humans for specific tasks and cannot easily generalize to other domains. Furthermore, the human factor leaves room for a variety of errors.

With the development of Recurrent Neural Networks (RNN), new NLG models have proved very effective (Graves, 2013; Sutskever et al., 2011; Brarda et al., 2017). This approach allows for working with arbitrary text sequences and automatically generating sensible natural languages, learned from large amounts of data. The use of RNNs have shown some promising results when generating Wikipedia articles (Graves, 2013), linux source codes, scientific papers or NSF abstracts (Karpathy et al., 2015).

The topic of cocktail recipe generation using RNNs, however, is quite novel, and not many relevant previous works have been found. A few projects in the past have dealt with food recipes generation, but for completely different purposes. The Cocktail-Ingredient project on Kaggle (Mader) aimed to retrieve cocktail ingredients just from the name. However, their core network architecture consisted of Convolutional Neural Networks.

Another project, Recipe1M (Salvador et al., 2017) trained a neural network which yielded impressive results on image-recipe retrieval tasks. The project used word2vec (Mikolov et al., 2013) representation of ingredients, and their core network architecture was an LSTM. They aimed to recognize a picture of a dish whose recipe the algorithm has seen during training, and output that recipe to the user.

7. Conclusions

In this project we investigated various advanced recurrent neural network models in order to generate new and original cocktail recipes. Our results demonstrate the feasibility of this network architectures to create novel, grammatically and semantically correct cocktail recipes.

We increased the data of our original datataset to contain 3347 recipes, each of which included a name, ingredients list and description. We concluded that our probabilistic model with both embedding and ingredients has a few distinct advantages over the simpler ones. Firstly, the word embedding allowed us to have a more efficient representation of our data, hence reducing memory requirements. We increased the sequence length, and were able to generate recipes with ingredients, where these matched the description.

Finally, we experimented with a wholly new name generation model and devised an appropriate evaluation method for our natural language generation task using human input. The evaluation results from our best performing model yielded ratings comparable to those of real recipes, further exemplifying the strength of the networks.

Future Work

It is worth mentioning the limitations in our experiments and provide direction for further work. Although we achieved satisfiable results with our probabilistic model with embeddings, we relied on the baseline hyperparameters and tweaking these could result in further improvement.

Our new name generating model didn't seem to perform very well on our generated recipes as illustrated in our evaluation survey. Applying our ingredients and description recurrent model to the name generating problem could potentially yield better results.

Our evaluation method could also be improved by increasing the number of sampled recipes and participants. Lastly, wholly new language generation evaluation methods could be explored.

References

All Recipes, Ltd. All recipes. URL https://www.allrecipes.com/recipes/133/drinks/cocktails.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Brarda, Sebastian, Yeres, Philip, and Bowman, Samuel R. Sequential attention: A context-aware alignment function for machine reading. In *Proceedings of the 2nd Workshop on Representation Learning for NLP,*

Rep4NLP@ACL 2017, Vancouver, Canada, August 3, 2017, pp. 75–80, 2017. URL https://aclanthology.info/papers/W17-2610/w17-2610.

Cheyer, A. and Guzzoni, D. Method and apparatus for building an intelligent automated assistant, May 3 2007. URL https://www.google.com/patents/US20070100790. US Patent App. 11/518,292.

Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Elbaz, Gil. Common crawl. URL http://commoncrawl.org/.

Graves, Alex. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Karpathy, Andrej, Johnson, Justin, and Li, Fei-Fei. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015. URL http://arxiv.org/abs/1506.02078.

Liquor, Inc. Cocktail recipes. URL https://www.liquor.com/.

Live in Style, Ltd. Live in style. URL https://www.liveinstyle.com/cocktails.

Mader, Kevin. Cocktail ingredients. URL https://www.kaggle.com/kmader/gan-drinks-model.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Olah, Christopher. Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015, 2015.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Rush, Alexander M., Chopra, Sumit, and Weston, Jason. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015. URL http://arxiv.org/abs/1509.00685.

Salvador, Amaia, Hynes, Nicholas, Aytar, Yusuf, Marin, Javier, Ofli, Ferda, Weber, Ingmar, and Torralba, Antonio. Learning cross-modal embeddings for cooking recipes and food images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Serious Eats, Ltd. Serious eats. URL https://www.seriouseats.com/recipes/topics/meal/drinks/cocktails.

Social+Cocktail Events, Ltd. Cocktail recipes. URL https://www.socialandcocktail.co.uk/.

Sordoni, Alessandro, Bengio, Yoshua, Vahabi, Hossein, Lioma, Christina, Simonsen, Jakob Grue, and Nie, Jian-Yun. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. *CoRR*, abs/1507.02221, 2015. URL http://arxiv.org/abs/1507.02221.

Sundermeyer, Martin, Schlüter, Ralf, and Ney, Hermann. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Sutskever, Ilya, Martens, James, and Hinton, Geoffrey. Generating text with recurrent neural networks. In Getoor, Lise and Scheffer, Tobias (eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pp. 1017–1024, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Wen, Tsung-Hsien, Gasic, Milica, Mrksic, Nikola, Su, Pei-hao, Vandyke, David, and Young, Steve J. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *CoRR*, abs/1508.01745, 2015. URL http://arxiv.org/abs/1508.01745.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron C., Salakhutdinov, Ruslan, Zemel, Richard S., and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015. URL http://arxiv.org/abs/1502.03044.