MLP Coursework 3: 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. We take cocktail recipe generation as a concrete example of such task and investigate the performance of recurrent neural networks as a language model. We first provide an overview of related work and describe common methods used for Natural Language Generation. Two baseline models are proposed, operating on character-level and word-level input. These models are used for subsequent experiments, which aim to establish the impact that different hyperparameters have on the performance of the model. It is shown that both models suffer from overfitting, but the word-level model produces output which is more grammatically and semantically correct. While we fail to provide a comprehensive approach to model evaluation at this stage, we propose several directions for future work. Our contributions also involve creating an original dataset of cocktail recipe descriptions, which is used to train our neural network models.

1. Introduction

The problem of Natural Language Generation has been widely researched by machine learning experts and there is a large body of literature addressing this problem. Nevertheless, many existing approaches are domain-specific and cannot be easily adapted to solve a different text generation task.

In this report, we present 'Deep Drinks' - a neural network model that can be used to generate new cocktail recipes based on existing ones. We start by introducing our research objectives in Section 1. Section 2 presents a review of related work in natural language generation. We discuss our data collection procedure and preprocessing methods in Section 3. Our methodology and theory behind relevant neural network models are presented in Section 4. We provide an overview of our experiments in Section 5. Lastly, we finish our report with a conclusion in Section 6 and directions for further work in Section 7.

1.1. Motivation

Natural Language Generation has been used in various useful 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).

We were interested in exploring and applying state of the art models in Natural Language Generation including the use of Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Generating cocktail recipes is an example of a novel and difficult natural language generation task.

1.2. Research Questions

The research question explored in this project is whether neural networks can learn the dependencies between words/characters and the conditional probabilities of words/characters in sequences so that we can, generate wholly 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

Upon completion of the project, we hope to achieve the following:

- Create a text corpus of cocktail recipes including their names, ingredients, and steps by scraping websites containing such data.
- Train both a character-level and word-level LSTM language model on our cocktail dataset.
- Investigate the effect of varying model hyperparameters, such as the sequence length, vocabulary size, number of epochs, hidden unit count, depth of the network and regularization.
- Discover and apply relevant evaluation methods for Natural Language Generation.
- Understand and apply state of the art RNN Natural Language Generation models to create novel but useful cocktail recipes.

2. 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 natural language generation 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 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. However, they aimed to recognize a picture of a dish whose recipe the algorithm has seen during training, and output that recipe to the user.

3. Data

Natural Language Generation 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.

3.1. Web 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. In practice, a single web crawler is used to scrape data from multiple websites by repeatedly following URL links contained on a webpage.

There are several popular libraries that are used to create web crawlers. For this project, we decided to use Scrapy, an open-source Python framework, which supports developing web crawlers, running the scraper and saving the output. Scrapy also allows web crawlers to follow URL links, which enabled us to start scraping on the main page and follow links to individual recipe web pages.

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.

In order to train our baseline model, we used data collected from "Social and Cocktail" (Social+Cocktail Events) and "Liquor" (Liquor) websites, which contain 938 and 215 unique recipes, respectively. Both of these websites provide descriptions that are semantically and stylistically similar, allowing us to combine data scraped from these websites into a single dataset.

3.2. Data Preprocessing

We designed two neural networks that operate on different types of input. One model takes sequences of characters as input, while the other one accepts sequences of words. Both models attempt to predict the next item in the sequence.

As a first step, we concatenated all descriptions into one string, separating them with a special symbol '|'. We made all characters lowercase and removed punctuation, apart from periods ('.'), which are used to denote sentence boundaries. By introducing '|' and '.' symbols we aimed to teach the model to predict sentence and description endings. Afterwards, we separated the string into characters for the first model and into words for the second model. We obtained 176250 characters and 31638 words. We leave 5% of our data out to be used as the validation set.

In order to use our data as input to neural networks, we had to perform data encoding. We first mapped each unique element in our dataset to a number from 0 to n, where n is the vocabulary size. For characters, n was equal to 63, while for words n could be any number between 1 and 1266. After that, we transformed these integers into one-hot encoding, such that each number corresponds to a binary vector with only one '1' and '0's in all other positions. Thus, our input became a matrix, where every row corresponds to an element in one-hot encoding.

Finally, we transformed the list of items into a list of sequences of items, represented as a 3D matrix. Every row of this matrix is a sequence of items in one-hot encoding. The task of our model then is to predict the next element of each sequence.

In order to illustrate our data preprocessing procedure, consider the following recipe:

"Add all ingredients to mixing tin and fill with ice."

We perform text preprocessing, remove punctuation (apart from periods) and add description boundary delimiters. If we consider the network that accepts sequences of words as an input, then we transform the string into a list of words:

["add", "all", "ingredient", "to", "mixing", "tin", "and", "fill", "with", "ice", ".", "|"]

We transform our data into one-hot encoding, where each element corresponds to an array with '1' in one position and '0' on every other position. Thus, each row in the resulting matrix has a size equal to the number of unique items in the

training dataset (i.e. vocabulary size):

```
[[1, 0, 0, ..., 0], [0, 1, 0, ..., 0], [0, 0, 1, ..., 0] ..., [0, ..., 1, 0, 0], [0, ..., 0, 1, 0], [0, ..., 0, 0, 1]]

    “add”          “all”          “ingredient”          “ice”          “.”          “|”
```

We map our data to an input matrix, where each row corresponds to a sequence of several elements. If sequence length is 2, then:

```
[
    [[1, 0, 0, ..., 0], [0, 1, 0, ..., 0]],          # [“add”, “all”]
    ...,
    [[0, ..., 1, 0, 0], [0, ..., 0, 1, 0]]           # [“ice”, “.”]
]
```

The output corresponds to the next element in the sequence. Thus, for the input matrix above, our output would be:

```
[
    [0, 0, 1, ..., 0],      # “ingredient”
    ...,
    [0, ..., 0, 0, 1]       # “|”
]
```

4. Methodology

We have experimented training both a character-level and word-level RNN language model on the 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/character in the sequence given a sequence of previous words/characters. This allows us to generate one word or character at a time.

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

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

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 and characters sequences like `add vodka to glass` but low probabilities to sequences like `ice to to ice ice`.

To make a prediction with our RNN model, 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.

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.

4.1. Long Short Term Memory

Recurrent neural networks (RNNs) 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 - or often just LSTMs - is a type of RNN architecture first introduced in 1997 (Hochreiter & Schmidhuber, 1997). The network architecture is modified such that the vanishing gradient problem is explicitly avoided, whereas the training algorithm is left unchanged.

In particular, when derivatives values are back-propagated from the output, derivatives are able to propagate arbitrarily far back without decaying significantly. This will allow the model to learn longer dependencies than a traditional RNN.

A standard neural network unit will only consist of the input and output activations, related by an activation function (e.g. tanh, ReLU...). The LSTM unit adds several intermediate steps.

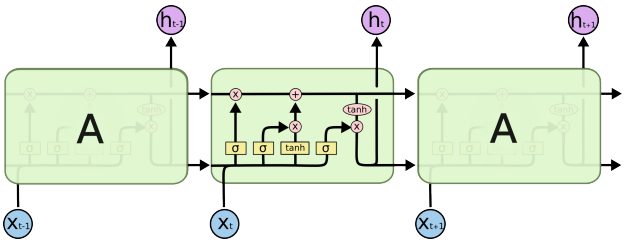

Figure 1. Diagram of a LSTM and its interacting layers. Source: (Olah, 2015)

A LSTM is described by the following equations (Olah, 2015):

- **Input Gate Layer:** it controls how much of the current input x_t and the previous output h_{t-1} will enter into the new cell:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

- **Forget Gate Layer:** decides which components we will erase or keep of the cell state. It outputs a number between 0 ("completely erase") and 1 ("completely keep") for each number in the cell state:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

- **Cell Update Transformation Layer:** transforms the input and previous state to be taken into account into

the new current state:

$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g) \tag{4}$$

- **Cell State Update:** computes the next timestep's state using the gated previous state and the gated input:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{5}$$

- **Output Gate:** scales the output from the cell:

$$o_t = \sigma(W_0 \cdot [h_{t-1}, x_t] + b_o) \tag{6}$$

- **Final LSTM Output:** output of the LSTM scaled by a tanh transformation of the current state:

$$h_t = o_t \odot \tanh(c_t) \tag{7}$$

We will now move on to a variation of Long Short Term Memory Units, GRUs, which have recently gained a lot of attention, and which we have experimented with in the next section.

4.2. Gated Recurrent Unit

Gated Recurrent Units (GRU) introduced by (Cho et al., 2014) in 2014, are closely related to LSTMs. Both types of networks address the vanishing gradient problem by gating the information. GRUs, however, use two gates: update gate and reset gate. The LSTMs forget and input gates are combined into the update gate, while the cell state and hidden state are also merged (Olah, 2015).

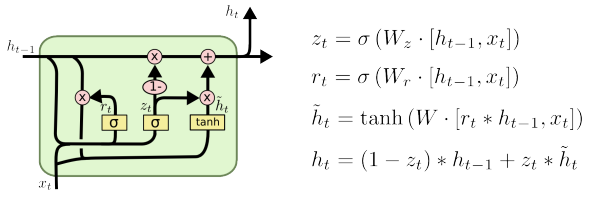

Figure 2. Diagram of a Gated Recurrent Unit.
Source: (Olah, 2015)

The resulting model above is less complex than LSTMs. However, due to the simpler architecture, GRUs are more computationally efficient, while displaying similar performance.

5. Experiments

We introduce a set of baseline experiments for studying cocktail recipe generation: a character-level and a word-level LSTM language model.

Unless otherwise stated, in the following experiments the batch size used is 32 and the activation function is ReLU. The output layer is a fully-connected dense layer with a softmax activation function, having one node per word in the vocabulary, where the value of each node is the likelihood of that being the next word in the sequence.

In order to choose the network architecture and hyperparameters, we experimented with sequence length, vocabulary size, number of epochs, hidden unit count, depth of the network and regularization.

For the first part of this investigation, we assessed the performance of a model by deciding whether the generated text is grammatically and semantically correct. Furthermore, the model should generate novel recipes, rather than outputting sentences or paragraphs that are part of the training data, which we consider overfitting.

The subsections below describe the common network architecture used in our character and word-level models, and the respective hyperparameters that have been explored.

5.1. Architecture

- **RNN Units:** we first designed a network consisting of a single hidden layer with 512 LSTM units and a fully-connected output layer with a number of neurons equal to the size of vocabulary. The number of weights in the model, however, was significantly larger than the number of our training points, which resulted in overfitting and slow training speed. In order to address this problem, we switched LSTM units for GRU neurons. This led to a decrease in the number of parameters that had to be learned, which made the model less complex. This improved the training speed and made the network less prone to overfitting. Model which used LSTMs had 2,843,324 parameters and took 438 seconds per epoch while using GRUs resulted in 2,222,268 parameters and took only 291 seconds per epoch.

- **Network Depth:** we also considered the effect of increasing the depth of the model. In a single-layer GRU model, the single hidden layer did not capture enough features to represent the temporal structure of the sequence. For this reason, we decided to separate the single hidden layer into two layers with 256 and 128 neurons, respectively.

- **Regularization:** we also explored the effects of regularization and more specifically dropout. At each training stage, individual nodes were either deactivated with probability 1-p or kept with probability p, so that a reduced network was left. Incoming and outgoing edges to a deactivated node were also removed. This technique reduces overfitting by preventing complex co-adaptations on training data. We considered different probabilities p ranging from 0 to 0.5 and found that the best performing one was 0.2.

5.2. Character-level model

The first baseline model we developed was inspired by the work of Karpathy et al. (Karpathy et al., 2015). In this

setting, the input to the network is a sequence of characters and the network is trained to predict the next character in that sequence with a Softmax classifier at each time step.

In the character-level model, the vocabulary size is a constant of 63 characters including the end of recipe symbol ("|").

Hyperparameters

- **Sequence length:** we have investigated the effect that varying the length of the sequence has on the network output. This is done during the preprocessing of the data, where we transform the list of input sequences into the form `[samples, time steps, features]` expected by the LSTM network.

  The sequence lengths expected for the character-level model are expected to be relatively long, encompassing at least 10-15 words from the recipe. We started with a sequence length of 10 and experimented with lengths of up to 50 characters. Going beyond that required too much memory and significantly slowed down the training of the model. However, longer sequence lengths always performed better.

- **Epochs:** Training the network with a large amount of data was challenging. Each epoch took from 5 to 30 minutes to run depending on the sequence length and the network overfitted quickly, producing results already present in the training set. A smaller number of epochs yielded the best results for the character-level model.

The character-level model often produced repetitive words and sentences, and more importantly was not semantically correct. We decided to try an alternative approach to improve the results and implemented a word-level model,as described below.

5.3. Word-level model

Our second baseline model was designed to take sequences of words as input and produce the next word in the sequence as an output.

Hyperparameters

- **Vocabulary size:** we investigated how constraining the vocabulary size affects the network output. We first converted all of the words to lower case, to reduce the vocabulary the network must learn. To further reduce the vocabulary and improve the modeling process, we removed the words "a", "an", and "the". They do not carry additional information, but they complicate training, as we observed during our early experiments. As these words tend to be the most probable predictions, the model becomes biased and the text generator would often get into a loop and predict sequences such as `add the the the the`.

  We found our dataset contained a total of 1266 unique words. Due to our one-hot-encoding data representation, and hence memory constraints, we were unable to use the entire vocabulary to train the model. To overcome this, we calculated the frequency of each occurring word in our dataset and ranked them from highest to lowest. We then used a varying number of the top words during training. We started with a small vocabulary size of 300, which we found insufficient for our network to generate semantical and syntactically correct sentences. As we increased the vocabulary size, we observed an increase in the quality of the output generated. Within our constraints, we found 700 to be the best size for the vocabulary that we can use for training.

- **Sequence length:** we have investigated the effect of varying the length of the word sequence on the network output. This is done by preprocessing the data similarly to the character-level model. The average length of a cocktail recipe description in our dataset is around 20 words. We started by looking at short sequences of length 3-6. This did not give good results; the text generated was semantically incorrect, and often repetitive. Due to the shortness of the sequences, the network architecture seems unable to remember the context of the words. As we increase the sequence length, we see considerable improvement in the generated output, as the network seems to become more aware of the context of words. Due to memory constraints, we found 15 to be the maximum sequence length we could experiment with, as the input matrix increases as a function of the vocabulary size and the number of observations as we increase the sequence length.

- **Epochs:** we trained our network architecture for 5, 10 and 15 epochs. When the network was trained for 5 epochs, it presented signs of underfitting, producing incomprehensible text. When trained for a longer number of epochs, 15 or over, the model would overfit the data, and copy entire phrases or even lines from the input text. We were able to reach an acceptable balance between generating mostly comprehensible and original text training the network for 8-12 epochs.

5.4. Results

Character-level model

As mentioned earlier, the character-level model did not produce the results we were hoping for. Many grammatical errors such as "pour the ingredients **to** a shaker" were present. Often, there were semantical errors such as "rub the lemon wedge on the outer except the cocktail glass". Finally, using smaller sequence lengths resulted in the model getting stuck in a loop - "...garnish with a chilled cocktail glass. garnish with a chilled cocktail glass. garnish with a chilled cocktail glass...". The best performing model with 2 GRU layers of 256 and 128 units, dropout and training for 5 epochs generated the following text:

seed = "s e r v e , a d d t h e i c e b l o c k a n d g a r n i s h l i b e r a l l y w i"

prediction = "t h a s t r a w b e r r y s l i c e . | p o u r t h e i n g r e d i e n t s i n t o a c o c k t a i l s h a k e r w i t h i c e c u b e s . s h a k e w e l l . s t r a i"

Word-level model

For all the models, it was the case that a vocabulary size of 700, and sequence length of 15 yielded the best results, which are presented in the table below. As mentioned earlier, a longer sequence length resulted in the model to be more context aware. Furthermore, a bigger vocabulary size allowed the network to generate a wider variety of sentences. In fact, text generated with low vocabulary size (e.g. 300) was often incomplete or incomprehensible.

Model	Units	Train Acc	Val Acc	Epoch
LSTM	512	0.7320	0.4210	6
GRU	512	0.8204	0.4684	6
GRU	256, 128	0.6934	0.4494	10
GRU (Drop)	256, 128	0.6190	0.4633	10

Table 1. Maximum achieved training and validation accuracies for variations of our baseline model, and epoch number in which the validation accuracy was achieved.

Our best model, 2 Layer GRU with Dropout, generated the text below:

seed = "tabasco salt and pepper into highball glass then pour all ingredients into highball with ice"

prediction = "cubes . stir gently . garnish with celery stalk . | pour ingredients into cocktail shaker with ice cubes . shake well for 10 - 15 seconds . strain into chilled cocktail glass . garnish with lemon twist . |"

6. Conclusions

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

We concluded that a word-level model has a few distinct advantages over a character-level: misspellings and non-words do not appear (unless they appear in the dataset) and the generated text is often more comprehensible due to the guarantee that each generated word follows its preceding word under some existing grammatical structure that is contained in the dataset. Furthermore, the word-level model was significantly easier to train and leaves more opportunities for further development.

Nevertheless, the character model is a more standard approach for natural language generation tasks. It does not require a large vocabulary size and, given a large enough dataset and longer training times, it could produce much better results.

Overall, the model that produced the best outputs was a word-level model consisting 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. The outputted recipes made grammatical sense and followed the expected structure.

7. Plan

A big limitation of our current work is a lack of a comprehensive dataset. While we already scraped two websites to collect the training data, we believe we could improve the quality of our model's output by scraping more websites and increasing the size of our dataset. We are aiming to scrape at least three more websites, which would increase the number of training points by at least a half.

Another limitation of our current model is data representation. We transform our input data to one-hot encoding, which restricts the vocabulary size and sequence length that we can use, due to memory limitations. We would like to consider improving our model by using a more space-efficient representation, such as word2vec (Mikolov et al., 2013).

We are also planning to improve our current baseline model. The outputs produced by our current network are deterministic, meaning that given the same input sequence the output will always be the same (i.e. the class with the highest probability). This model structure has a negative side effect of getting stuck in loops, where a certain seed will always result in the same text generation. We propose an extension to our model, such that the output of the next element is sampled from a probability distribution defined by the output layer of the network. Because softmax function guarantees that the output values add up to 1, we can assign a probabilistic meaning to different class outcomes. This will allow us to develop a more complicated language model that the one used so far.

At the moment, we also fail to provide a comprehensive approach to model evaluation. Proper result validation is complicated by the fact that the quality of generated natural language text can be subjective, and existing evaluation methods cannot capture both grammatical and semantic components of natural language. We propose using user studies as a way to evaluate our models, by asking participants to assess the quality of produced sentences.

References

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.

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/.

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.

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.

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.