# Forage: Optimizing Food Use With Machine Learning Generated Recipes

**Angelica Willis** ARWILLIS@STANFORD.EDU
**Elbert Lin** EL168@STANFORD.EDU
**Brian Zhang** YZHNG@STANFORD.EDU

## 1. Introduction

Food waste is a major issue in the United States. For an American family of four, the average value of discarded produce alone is nearly $1,600 annually (EPA, 2015). Our project, Forage, is a machine learning algorithm that considers what you have in the fridge or pantry, to generate an innovative recipe that utilizes those available ingredients. Specifically, Forage takes in a string of available ingredients words separated by comma, converts them to a vector representation and uses a recurrent neural network to generate a full recipe including instructions, the category and the title. Our goal would be for Forage to help minimize food waste while helping to create your next delicious, never seen before meal.

## 2. Related Work

Our topic of recipe generation using machine learning algorithm is quite novel and, as a result, not too many relevant previous works were found. MIT's recent project, Pic2Recipe (Salvador et al., 2017), incorporated a recipe (ingredient and cooking instruction) module behind the image recognition module. Similar to our project, their recipe module utilized word2vec representation of ingredients words and their core learning algorithm was also recurrent neural network, or LSTM to be specific. But we had fundamentally different goals: Pic2Recipe aimed to recognize a picture of a dish whose recipe the algorithm has seen during training, and output the particular recipe to the users. That is why we had different network model structures. Pic2Recipe used two separate LSTM models on ingredients and instructions, so there was no particular connections learned between the two. In the two-stage LSTM model for instructions, their representation was on the sentence level where each sentence encoding served as a context for decoding or predicting the previous and the next sentences, allowing accurate recalls of trained instruction sentences.

## 3. Dataset and Features

Because deep learning algorithms require a large amount of training data for the model to perform well, we chose to leverage Meal-Master (Meal-Master, 2013), the largest, free recipe database available. From this flat file dataset, we have filtered out 60k recipes and extracted title, categories, ingredients and instructions of each recipe example, and use word2vec to convert word tokens in training data to a vector represented feature for the LSTM model.

### 3.1. Raw Data Preprocessing

We prefer a more concise representation of the ingredients, as features of the model, to control the feature space size. Using the `nltk` package, we were able to tokenize and lemmatize each word to stem it without losing its natural form. Then, we tagged the sequence of the word with speech tags and pick out "NN", referring to noun, singular or mass words. To be even more selective, we searched key words (cook, food, fruit, etc.) in the meaning of the word from WordNet and made sure it is actually some type of food.

To train the specific "recipe language" and the cooking instruction relation between words and sentences in the LSTM model, we think there is a lot of valuable information within the raw text complexity itself. To preserve this, we tokenized the instructions by sentence. Indeed, noises in recipe instructions, like irrelevant notes or citations, are still present at this point. One effective way to remove them, after we surveyed the noisy examples, was to search for named entities and discard the sentence. Admittedly, the Stanford Named Entity Recognition is much more accurate than the tagger from `nltk` in our implementation, yet at a computing cost we could not afford for 60k recipe examples. Now, the natural language preprocessing is complete and we are ready to pack the title, categories, ingredients and instructions into a JSON-like structure, as in the recipe example shown in Figure 2.

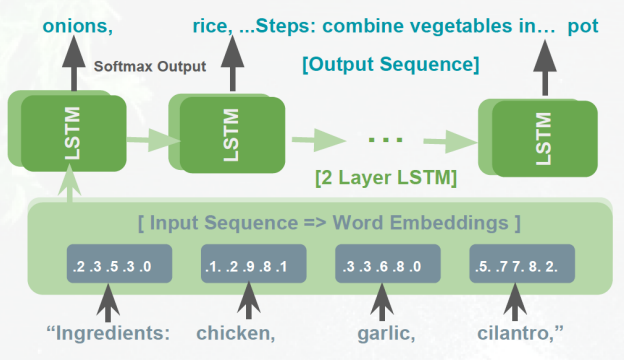


*Figure 1.* Structure of the LSTM Model

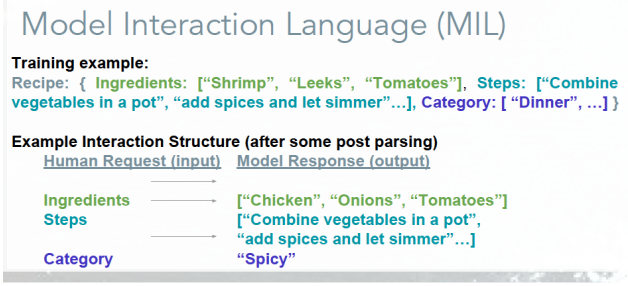


*Figure 2.* Model Interaction Language allows for dynamic model output. For example, a request for ingredients will return output data beginning with ingredients.

### 3.2. Vector Representation of Words

One traditional method that feeds word strings into a model is one-hot encoding, where each feature vector would take the size of the model vocabulary. Word2vec (Mikolov et al., 2013), in contrast, allows for a variable sized (often shorter than vocabulary size) feature vector and mapping semantically similar words close by each other. This representation of the words was extremely helpful for the LSTM model to learn the meaning of words and produce sensible results. Within the Word2vec models, we chose skip-gram over continuous bag-of-words (CBOW) model, because skip-gram predicts source context-words from the target words, while CBOW does the opposite (TensorFlow, 2017). From the training perspective, skip-gram views each word and the expected context words as multiple examples and usually works better for a large dataset. In terms of the project goal, skip-gram is also more suitable for creating new recipes from given ingredients, because it is able to build a context from words, not to fill in a blank in an existing context.

## 4. Method

We have developed a recurrent neural network (RNN) to create each recipe, given an input sequence of initial ingredients. The expected output of the model is a set of instructions. The generative nature of our project means that traditional evaluation metrics do not reveal much useful information. To aid evaluation, we used $k$-means clustering to gain a sense of how similar recipes should look, and could determine whether our Forage recipes were reasonable, or outliers that did not correlate with any of the trusted clusters, or categories of recipes from our dataset.

### 4.1. LSTM

We have chosen the LSTM RNN because of its ability to "remember" sequences of data, capturing remote dependencies, to generate new data. We modeled a word-based LSTM to consecutively generate words based on the previous words sampled. With an initial feeder sequence of ingredients, the model generated complete recipes, word by word. The "long term" memory was achieved by stored in a vector of memory cells. During training, the LSTM could decide to overwrite a memory cell, use it for the current prediction, or leave as is until the next time step.

We designed our model to have a softmax output with two stacked LSTM hidden layers, and compared several structural differences, including 256 hidden cells versus 512 hidden cells, instance sequence lengths of 25 and 50. This structure can be seen in Figure 1. Each LSTM cell unit has the form given in (Zaremba et al., 2014). For each time step $t$, our cell states are given by:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

where $i$, $f$, $o$ and $c$ are respectively the input gate, forget gate, output gate, cell and cell input activation function vectors, $\sigma$ is the logistic sigmoid function, and $h$ is the overall deterministic state transition function. We chose a common loss function which minimizes the average negative log probability of the target words: $loss = -1/N \sum_{i=1}^{N} \ln p_{targets}$.

We trained the model with the dataset represented as one large corpus of recipes, where one training instance is a text sequence of fixed length (25 or 50 words). This allowed the input to have a fixed size so that we can

create our training batch as a single tensor of shape `batch size` × `sequence length` × `features`.

#### 4.1.1. Model Interaction Language (MIL)

Out of the necessity for our model to understand the complex nature of our data (i.e. ingredients, steps, recipe titles, and categories) we designed and implemented a new approach to facilitate genuine and dynamic interaction with a trained model, which we call Model Interaction Language, or MIL (see Figure 2). MIL is similar to the structure of JSON, and not only allows us to request different types of model output, the additional structure acts as scaffolding which enables faster learning.

#### 4.1.2. Recipe Sampling

In our models, which generated the next most probable word in a the sequence based on previous words, we explored two methods of sampling recipes from the trained model, beam search and greedy search. Greedy search predicts the next word based the single most probable word in the probability distribution at time $t$, beam search differs in that it expands the $x$ most promising nodes and keeps track of probabilities for each beam. We found beam search with a width of more than 1 to produce lower quality recipes, therefore the results focus on greedy sampling only.

### 4.2. $K$-means Clustering

We used minibatch $k$-means to separate our recipe dataset into 88 clusters. The number of clusters was chosen using silhouette analysis, choosing $k$ that gave the largest average silhouette score and created clusters with similar sizes (Pedregosa et al., 2011). See Figure 3. For speed and accuracy, we used a sparse term frequencyinverse document frequency matrix of 10000 features, and found a low rank approximation using latent semantic analysis with 1000 components. The number of components was chosen such that the explained variance of the SVD step would be around 70%. We wish to use these clusters as a means of classifying our generated recipes, and we can compare against recipes in the dataset, using a metric such as distance with cosine similarity. Though it may not be as accurate as actual human evaluation, comparing against clusters can give us a general idea of how well our main method is doing.

## 5. Experiments

### 5.1. $K$-means Experiments

We were able to obtain our clusters with the feature extraction and $k$-means algorithm (finishing in a total of 17.46 minutes even on a weak laptop). Due to the size of the dataset and number of clusters, however, it is infeasible to plot/view the clusters, though upon a quick survey of the top terms of each cluster, it appears that they are more so separated by ingredients, than by a cooking method (e.g. bake, boil). For example, the top terms of one cluster are "brisket beef meat" and another "rice water heat".

### 5.2. LSTM Experiments

Given the computation resources available to us (3 virtual machines with 8 CPUs each) we were able to train between 3-6 epochs for each of the models tested with 50 training examples per minibatch of gradient descent. We have seen an average reduction in training loss from 12.82 to 3.23 over these iterations. Though the exact value of the training loss gives little precise information about the usefulness of the recipes, we can determine when the model has "converged." We reduced our per-batch training time by 52% by training on a vocabulary size of 150,000 words instead of the original vocabulary size of the data (over 300,000 words and symbols) with virtually no impact on model results. (See Table 2.)

#### 5.2.1. Sample Recipes

Table 1 shows examples of output from our test models. These recipes were produced by querying our model using MIL "Ingredients: `INGR_LIST`", where `INGR_LIST` contains the first 1/2 ingredients from the original recipe. Our models have proven to be able to produce seemingly consistent recipes that are "never before seen" (or at least do not show up in Google searches). In general, results indicate that when the number of words sampled increases, the quality of recipe output decreases. (See Table 1.)

## 6. Results and Discussion

### 6.1. Metric Evaluation

With the centroids we obtained using $k$-means clustering, we assigned our generated recipes to the closest centroid. For initial testing, we used three metric evaluations: Euclidean and cosine distance, and Language Tool (LT) score. The LT score uses the following for-

mula as seen in (Napoles et al., 2016):

$$1 - \frac{\#\text{ errors}}{\#\text{ tokens}}$$

where the errors are those detected by Language Tool (Myint, 2017) and tokens are those found by `nltk`. The results are tabulated in Table 2. We see that our models generally produced weaker scores than the original, as expected, but are close enough that we believe that our models are at least working as intended, though not as effectively as we would have liked.

### 6.2. Human Evaluation

To more accurately evaluate the success of our models, we asked other people to read and rate the grammar and logic of our output recipes. 20 output recipes were chosen, shuffled with 10 random recipes from the dataset. We received 100 responses, where each response matched a single recipe. Readers were tasked with rating the grammar, steps logic (i.e. are steps in logical order), ingredients logic (i.e. do ingredients match the steps), and title logic (i.e. if it has a title, does it match the steps) of the recipes on a scale of 1-5. Readers were also asked if they were able to make the food, and finally given a long-answer box where they could fill in additional comments.

The average ratings for the recipes are shown in Table 3. We note that some of the output recipes were simple one-line recipes, which would skew the results, and so we removed the scores on these recipes to better capture the intent of our models. For comments, on original recipes we saw such observations as "The instructions were really good and easy to understand." while for our output recipes, we sometimes received "Some of the steps made sense but all together I wouldn't really be able to make this."

We see that, although the grammar rating for our output recipes were of an average score, the ingredients, steps, and title logic did not perform particularly well, and readers were not able to make the food. The scores marginally increased when removing one-line recipes, though they were still below what we desired. It appears that logic is rather difficult to enforce in a machine model, and we would like to explore more methods as to how that might be achieved.

### 6.3. LSTM model

Thanks to MIL, our generated recipes overall made semantic sense and had an expected structure. The output from raw string input without the MIL structure in Table 2 seemed to achieve good scores all around, but most of the recipe outputs lacked consistent structure and are not practically usable. For example, most recipes had multiple ingredients sections.

However, the correlation of ingredients in the "ingredients" and "instruction" sections still needs improvement, as reflected in the evaluation. One possible improvement for the ingredients logic would be to characterize this logic in the cost function. Specifically, we could penalize each ingredient appearing in the "instruction" sections but is not part of the "ingredients" section.

Although the LSTM model was computationally expensive to train, we found by controlling the number of most frequent words as vocabulary, the model could run faster and avoid overfitting. By providing a pretrained word2vec model to TensorFlow, we also sped up training, but care must be taken to ensure the consistency between the word2vec model training data and the data input of the LSTM model.

## 7. Conclusion and Future Work

In this project, we took on the challenge of using word2vec and an LSTM to generate new recipes given available ingredients. Our biggest challenge was mass evaluation of this generative model, but we leveraged creative ways to understand output through $k$-means clustering and human survey. Among our experiments, we found that the MIL structure, reduced vocabulary size of 150,000, larger number of hidden cells at 512, and smaller sequence length at 25 all help generate recipes that have the expected recipe structure, are close to existing recipe clusters and make grammatical sense. In the future, we would like to explore other new frontiers in evaluation.

Additionally, we hope to focus on a more customized training cost function that penalizes the model for not consistently using the ingredients (from the ingredient list it produces) in the accompanying steps.

## 8. Contributions

- Angelica Willis: LSTM model
- Elbert Lin: $K$-means clustering and evaluation
- Brian Zhang: Training data processing

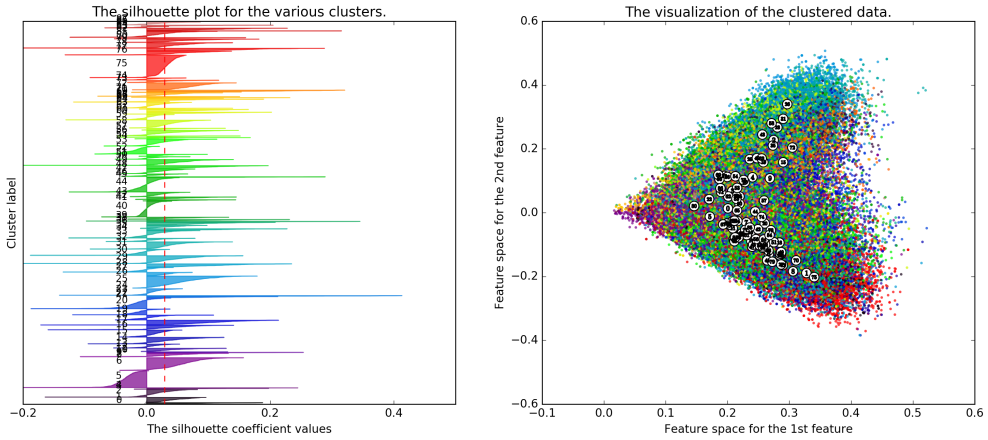


*Figure 3.* Silhouette analysis for $k$-means clustering on sample data with 88 clusters

*Table 1.* Example Recipes Produced

| Quality | Recipe Steps |
|---|---|
| Poor | Title: HERSHEY'S WILD PEA RED BEANS WITH CHIP CAKE, "Ingredients": egg, milk, wine, salt, celery, egg, catsup, orange tomato and turmeric on night.", "1976."], "Categories": ["Cookies", "Bundt"], |
| Average | Title: RICE & BEAN SPROUT SALAD "Ingredients": bean, pepper, sauce, lemon juice, cream, salt Steps: "Cover and shake until thick and opaque.", "Add the hard-boiled egg whites, a little at a time, turning them so they won't brown and bright green.", "Increase to low heat until constant all the ingredients are smooth.", "Add them to the pasta.", "Add the remaining ingredients and cook briefly, into a sauce.", "Serve with marinade to low-heat point are served the room-temperature with your favorite sticks packet or for a tossed salad, a few zucchini... |
| Good | Title: CATFISH SOFRIT, Categories: "Seafood", "Ingredients": spinach, egg, salt, mustard, turmeric, Steps: "Combine soy sauce, sugar, curry powder and oregano in large saucepan.", "Bring to boiling, stirring occasionally.", "Add soy sauce and wine; simmer, covered, 2-3 minutes.", "Add snow peas and cook until well-browned and slightly thickened. Beat 1/2 cup at a time and browned, stirring after 45 seconds before serving.", "Serve over rice" |

*Table 2.* Average distances and scores for different models

| Model type | Euclidean distance | Cosine distance | LT score |
|---|---|---|---|
| Original recipes | 0.875 | 0.527 | 0.913 |
| Vanilla model (VM), 4 epochs | 0.943 | 0.660 | 0.782 |
| VM with reduced vocab. size: 150,000, 3 epochs | 0.931 | 0.628 | 0.792 |
| VM with reduced vocab. size: 150,000, 6 epochs | 0.947 | 0.672 | 0.778 |
| Model with pretrained word2vec, 3 epochs | 0.910 | 0.583 | 0.750 |
| Model with String input (not MIL) 3 epochs | 0.910 | 0.576 | 0.885 |
| VM with 512 hidden cells, vocab. size: 150,000, 4 epochs | 0.939 | 0.645 | 0.771 |
| VM with 512 hidden cells, vocab. size: 150,000, 6 epochs | 0.948 | 0.672 | 0.763 |
| VM with sequence len.: 50, vocab. size: 150,000, 3 epochs | 0.943 | 0.658 | 0.748 |

*Table 3.* Average ratings for each category on different model types

| Model type | Grammar | Ingredients | Steps | Title | % made food |
|---|---|---|---|---|---|
| Original recipes | 4.538 | 3.462 | 4.154 | 4.152 | 0.654 |
| 256 hidden cells, 4 epochs | 3.194 | 1.871 | 2.226 | 1.545 | 0.097 |
| 512 hidden cells, 4 epochs | 2.304 | 1.435 | 1.348 | 1.850 | 0 |

## References

EPA. Americans throw away $1,600 of wasted food per year. reduce food waste and save money http://www.epa.gov/recycle #nowastedfood #frscharleston, 2015. URL https://twitter.com/epaland/status/666340964064186368.

Meal-Master. Now you're cooking! recipe software, 2013. URL http://www.ffts.com/.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL http://arxiv.org/abs/1310.4546.

Myint, Steven. language-check 1.1, 2017. URL https://pypi.python.org/pypi/language-check.

Napoles, Courtney, Sakaguchi, Keisuke, and Tetreault, Joel R. There's no comparison: Reference-less evaluation metrics in grammatical error correction. *CoRR*, abs/1610.02124, 2016. URL http://arxiv.org/abs/1610.02124.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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.

TensorFlow. Vector representations of words, 2017. URL https://www.tensorflow.org/tutorials/word2vec.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL http://arxiv.org/abs/1409.2329.