

# CS63 Spring 2019

## 'What's Cooking?': Using recipe ingredients to categorize the region of a dish

Bayliss Wagner and Akshay Srinivasan

May 2019

## 1 Introduction

For this project, we hoped to see if we could use machine learning techniques to teach a program how to recognize the region from which a dish originated, based on its ingredients. These regions or 'cuisine types,' of which there were twenty in total, included types such as 'Mexican', 'Indian', or 'Thai.'

We chose to use a deep neural network with one hidden layer to solve this problem. Neural networks mimic the brain's structure to learn by transforming input data in multiple layers, including a hidden layer. Each layer has units which have an activation value, and all of these units are connected to each of the units on the neighboring layer by a weight. The activation of the a unit is equal to the output of the activation function, which for multi-layer nets is a differentiable function that sums up the activation\*weight of each incoming connection and outputs 0 or 1. Once the input data has reached the output layer, the neural network checks each output against the target output for that input, output pair, which is how the learning is 'supervised.' Then, for deep neural networks, error is backpropagated through the network in order to correct the connections that were most responsible for causing it. This is how learning occurs. We specifically used the Keras API sequential neural net for this project, which runs in Python and uses TensorFlow as a backend. Because it is based on linear algebra, as with other NNs, all data must be converted into numeric values, which is why we used preprocessing techniques (outlined below) to do.

We only used dense layers because the data does not lend itself to a visual or spatial representation, which would require the use of a convolution net.

## 2 Method and Details

### 2.1 Data Set

For this project, we worked on an Kaggle competition called 'What's Cooking?' Kaggle provided a training dataset of 39,774 recipes, each of which had a list of around five to twenty ingredients each in the form of strings, and a target output of that recipe's cuisine type (also a string). The competition also provided a test dataset of 9,944 recipes that did not have outputs. We submitted our network's prediction of the cuisine types of these recipes to Kaggle in the form of a csv file. A smart cooking app called Yummly provided the datasets.

The features in this dataset would be combinations of ingredients that are commonly used in some cuisine types (take, for example, green tomatoes and cornmeal in Southern cooking).

While testing our code, we split the provided training dataset by putting the first 30,000 recipes in the training vectors and the remaining 9,774 recipes in the validation set, but we used the full training set to train the network when we were preparing to submit our predictions for the Kaggle test dataset. Keras uses a lot of memory, so we used a lab computer with a good GPU so that we didn't have to wait too long for the data to process.

To preprocess the data, we first determined the total amount of ingredients and cuisine types in addition to the amount of unique ones and their frequencies. We found that there were 7,137 unique ingredients between the training and test datasets and 20 cuisine types. For ingredients, we created a multiple-hot vector for each recipe that converted a recipe's ingredients to distinct numbers between 0 and 7,136 and then set the activation to 1 to represent each ingredient in the recipe. For cuisine type (the target output), we gave each cuisine a numeric label and converted those to a one-hot vector of length 20 that each recipe had as its output. We created a vector of the unique ingredients and the unique cuisine types, and we used those to create dictionaries that mapped the strings to numbers.

To create the output, we created a dictionary that mapped the cuisine numbers to machines, and used the csv library to generate an output file with a column for the recipe id and a column for the model's prediction (from the Keras predict method).

## 2.2 Neural Net Parameters

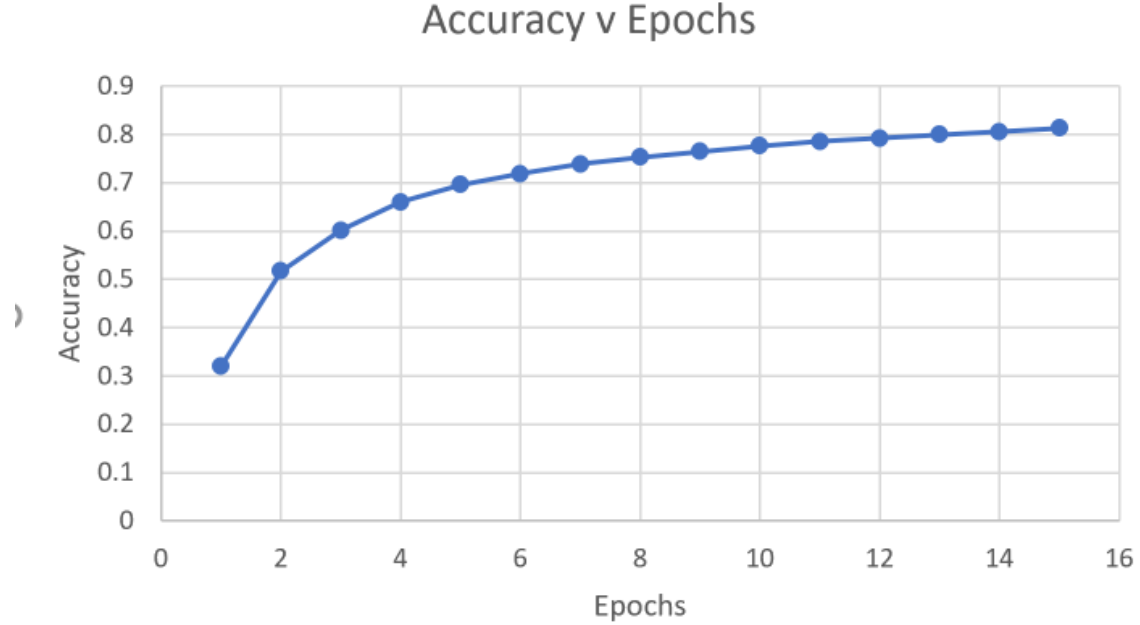
Our network had three layers with 3000, 500, and 20 units each, respectively. The network had 6,459,270 trainable parameters in total. We used 15 epochs because the program's accuracy leveled out after that, which would leave the model vulnerable to overtraining.

The first dense layer takes the input shape of the 7137 unit long multiple-hot vector and converts it to significantly less units. This is due to the ingredients in the multiple hot vector being largely not clustered which means immediately decreasing the input complexity doesn't have a significant impact on accuracy. The second dense layer allows the network to have better pattern recognition and gives it the opportunity to assign different values to groups instead of mapping directly to an output.

Each input has a relatively small amount of possible features, so we did not feel we needed more than three layers, with only one hidden layer. We do not know how much of each ingredient the recipe calls for, which could give us more of a clue about which cuisine type the recipe is. (Southern dishes, for example, might use a relatively higher quantity of butter.) This would introduce the need for more complicated preprocessing and more layers. Another feature of the recipe data that we did not have is cooking time, or number of servings, or preparation instructions, all of which could have had interesting implications for cuisine type. Our accuracy also leveled out with three layers, and the program was slower when we used four.

We used Stochastic Gradient Descent as our optimizer to train the network. Error can become infinitesimally small by the time it is propagated back to weights in early dense layers, so we chose to use Rectified Linear Units, which have no vanishing gradient of error signals, as our activation function for all dense layers except for the output layer. We chose softmax as our activation function for the output layer because the last layer needs only return a 20 character vector instead of a function with an easily differentiable slope.

Figure 1: Figure 1



We were able to use categorical crossentropy as our loss function because we converted our data into a categorical format in pre-processing.

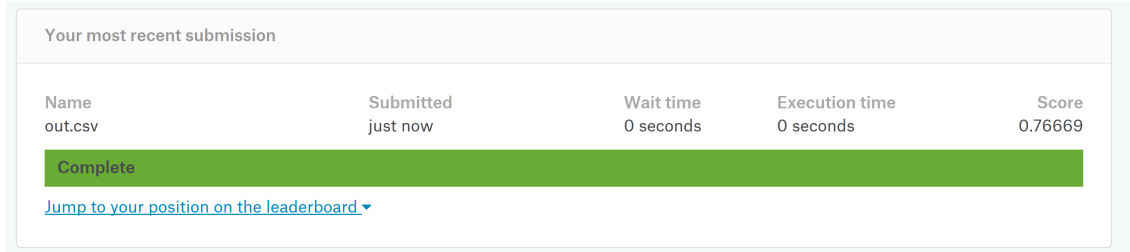
### 3 Results

The final model achieved a 76.669 percent accuracy in the Kaggle submission of the test data, roughly 3 correct for every 4 recipes. Because there are 20 possible cuisine types, a random guess would provide the program with an accuracy of 5 percent, so our program has shown that it has actually learned and gained the ability to differentiate among these recipes. We consider this program very effective because the program of the highest scorer on the Kaggle competition's leaderboard had an accuracy of 83 percent, which is not far from our accuracy (which places us in around 825th place, comparing to those who competed three years ago). We did not aim for a 100 percent accuracy rate because there are no set rules for what ingredients go in a dish belonging to a certain culture, and ingredients do not provide the full picture; a recipe with instructions for preparation would do that.

The first implementation we tried was a basic neural net from the work we did in lab7. Our current model outperformed that model by over 70 percent and also ran faster.

At a baseline, the model with one epoch achieved a 42.5 percent success rate. We then tried varying the number of epochs on which we trained the data. (See figure 1.) The Figure shows the accuracy per epoch for our final model. Our model submission was one of the only ones that required only 1 try to get over 70 percent.

Figure 2: Submission Record

A screenshot of a submission record interface. At the top, it says "Your most recent submission". Below this is a table with five columns: Name, Submitted, Wait time, Execution time, and Score. The row shows "out.csv", "just now", "0 seconds", "0 seconds", and "0.76669". Below the table is a green bar with the word "Complete" in white. At the bottom, there is a blue link that says "Jump to your position on the leaderboard" followed by a downward arrow.

Name	Submitted	Wait time	Execution time	Score
out.csv	just now	0 seconds	0 seconds	0.76669

Complete

[Jump to your position on the leaderboard](#) ▼

## 4 Conclusions

Without much more effort on the part of the programmer than converting data from strings to vectors, we were able to create a program that sorts recipes based on cuisine type at a rate of 76.67 percent accuracy on the test set. . Though the program is costly for both time and memory – it took about 2 minutes to train the network – once it was trained, it could make predictions very quickly, much more quickly than a human could. The program’s success is thanks to Keras, especially because it is a free software, and other recent advancements in the field of deep neural networks. Because we are in the field of artificial intelligence, we can also compare the intelligence of our program to human intelligence. I was not able to guess 3 correct out of 4, especially because I have never eaten Filipino food and am unfamiliar with the categories, but I feel that if I had studied and learned, I would have been able to have at least a 90 percent rate of accuracy. Thus our program in particular does not provide confirmation for the theory that AI can supersede human intelligence. However, it does show that AI can likely learn to predict faster than humans with little prior knowledge of a data set or situation.

It is likely that the missing accuracy came from the model being unable to properly identify cuisines that tend to have similar ingredients such as british and the US. Solving this issue would require significantly more model complexity because it would need to be able to assess smaller differences, possibly at a single-ingredient level.

While this initial experiment was successful, we can still consider ways in which we could increase accuracy. With more teammates and a little more time, we could have sorted all of the unique ingredients into groups such as spices, proteins, veggies, fruits, dairy products, grains, and oils, then made a matrix in which each row represented one of these groups, and each column was an ingredient in that group. It would be interesting to see if this would increase the accuracy of the program, because it likely would have helped the neural net distinguish between the types of grains that Indian dishes usually use, for example, as compared to those that American dishes usually use. However, it is possible that the neural net made some of these associations on its own during training.

This same network setup can be used in future projects that try to classify groups by characteristics that share the same format.