Recipe Generator using RNN

Adarsh Salapaka, Harsh Tandon,
Jaskaran Kohli, Vidhi Gandhi

- Problem Statement
- Dataset information
- Model description
- Training results
- Demo
- Key Takeaways and Future Work

Problem Statement

Problem: Bored of eating same thing during this pandemic? People want to try new recipes.

Solution: Our project helps in generating new recipes along with cooking instructions.

How:

- **Trained** ~100k recipes dataset using TensorFlow, LSTMs and RNNs.
- **Build** model which suggests an entire recipe along with cooking instructions and ingredients based on ingredients input by user.

Dataset contains **125,000 recipes** from various food websites such as Foodnetwork.com, Epicurious.com, Allrecipes.com.

A typical recipe looks something like this:

***Title**: Guacamole*

***Ingredients**: 3 Haas avocados, halved, seeded and peeled, 1 lime, juiced, 1/2 teaspoon kosher salt*

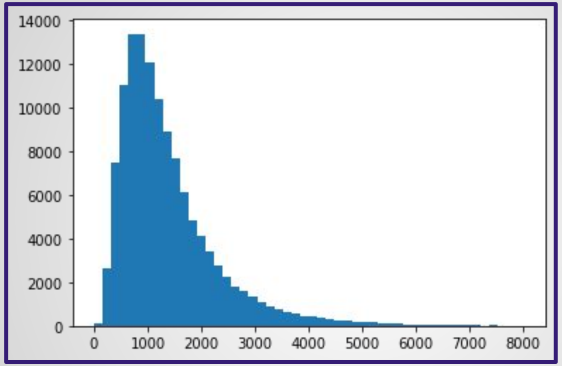
***Instructions**: In a large bowl place the scooped avocado pulp and lime juice, toss to coat. Drain, and reserve the lime juice, after all of the avocados have been coated. Using a potato masher add the salt, cumin, and cayenne and mash.*

***Source**: http://www.foodnetwork.com/recipes/alton-brown/guacamole-recipe*

Picture:

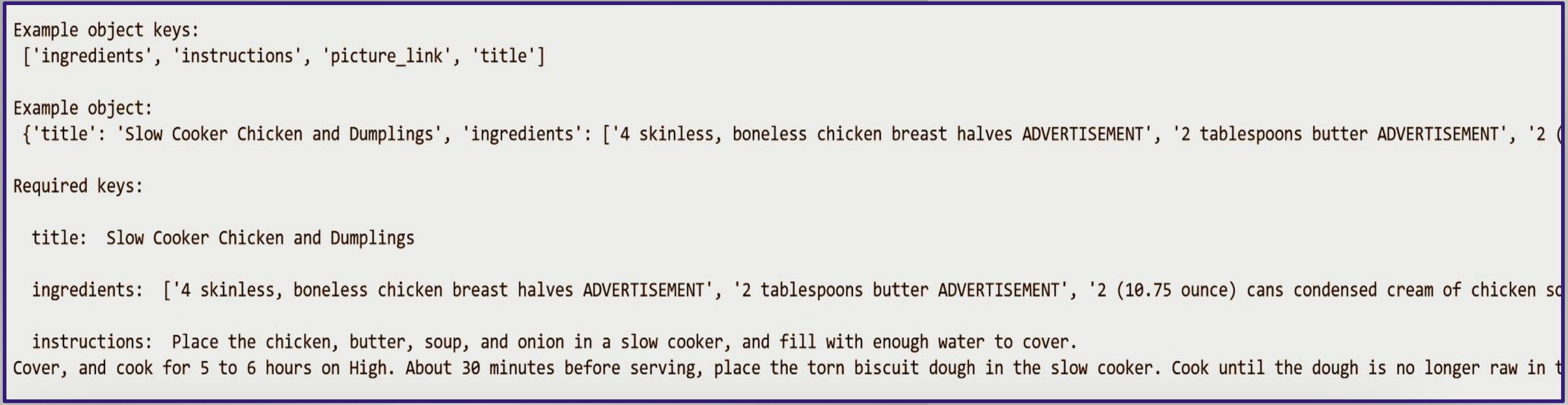
Text Preprocessing :

- Filtering out incomplete examples
- Convert recipe objects to strings
- Filtering out large recipes



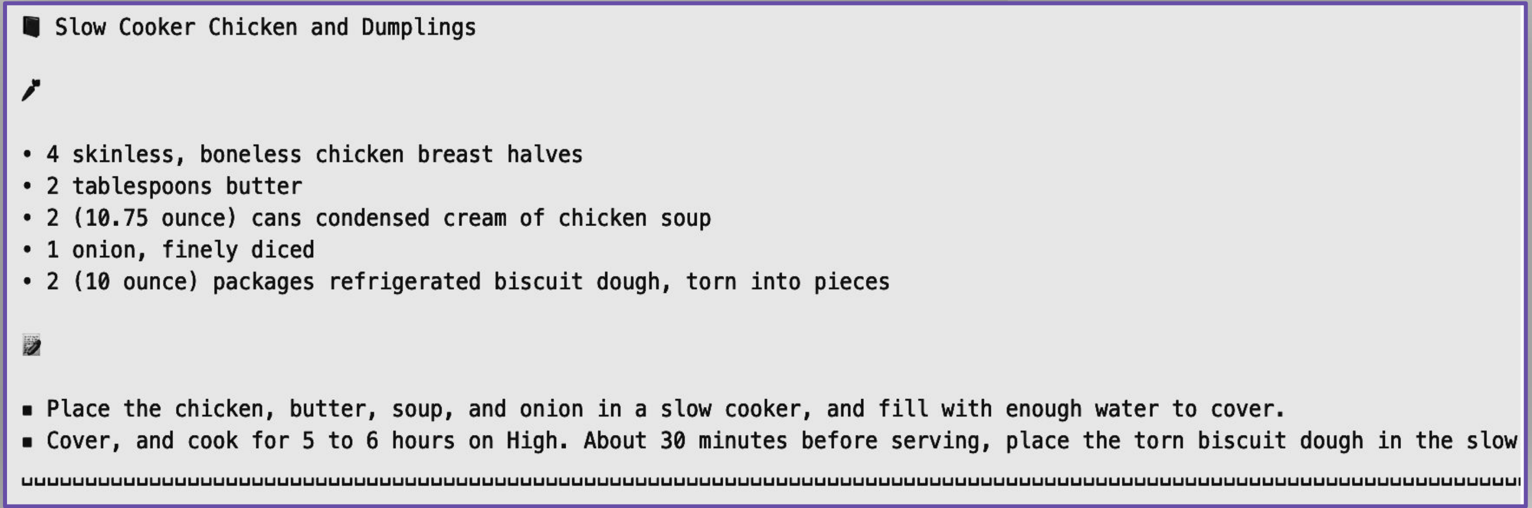
- Creating vocabulary
- Vectorize the dataset
- Adding padding to ensure all recipes have same length

Before Text Preprocessing:



```
Example object keys:
 ['ingredients', 'instructions', 'picture_link', 'title']

Example object:
 {'title': 'Slow Cooker Chicken and Dumplings', 'ingredients': ['4 skinless, boneless chicken breast halves ADVERTISEMENT', '2 tablespoons butter ADVERTISEMENT', '2 (

Required keys:

  title:  Slow Cooker Chicken and Dumplings

  ingredients:  ['4 skinless, boneless chicken breast halves ADVERTISEMENT', '2 tablespoons butter ADVERTISEMENT', '2 (10.75 ounce) cans condensed cream of chicken so

  instructions:  Place the chicken, butter, soup, and onion in a slow cooker, and fill with enough water to cover.
Cover, and cook for 5 to 6 hours on High. About 30 minutes before serving, place the torn biscuit dough in the slow cooker. Cook until the dough is no longer raw in t
```

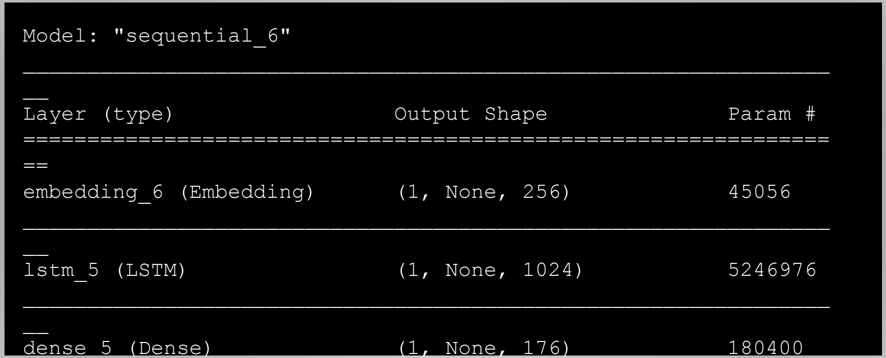
After Text Preprocessing:



```
📕 Slow Cooker Chicken and Dumplings

🔪

• 4 skinless, boneless chicken breast halves
• 2 tablespoons butter
• 2 (10.75 ounce) cans condensed cream of chicken soup
• 1 onion, finely diced
• 2 (10 ounce) packages refrigerated biscuit dough, torn into pieces

📝

▪ Place the chicken, butter, soup, and onion in a slow cooker, and fill with enough water to cover.
▪ Cover, and cook for 5 to 6 hours on High. About 30 minutes before serving, place the torn biscuit dough in the slow
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

Model: LSTM

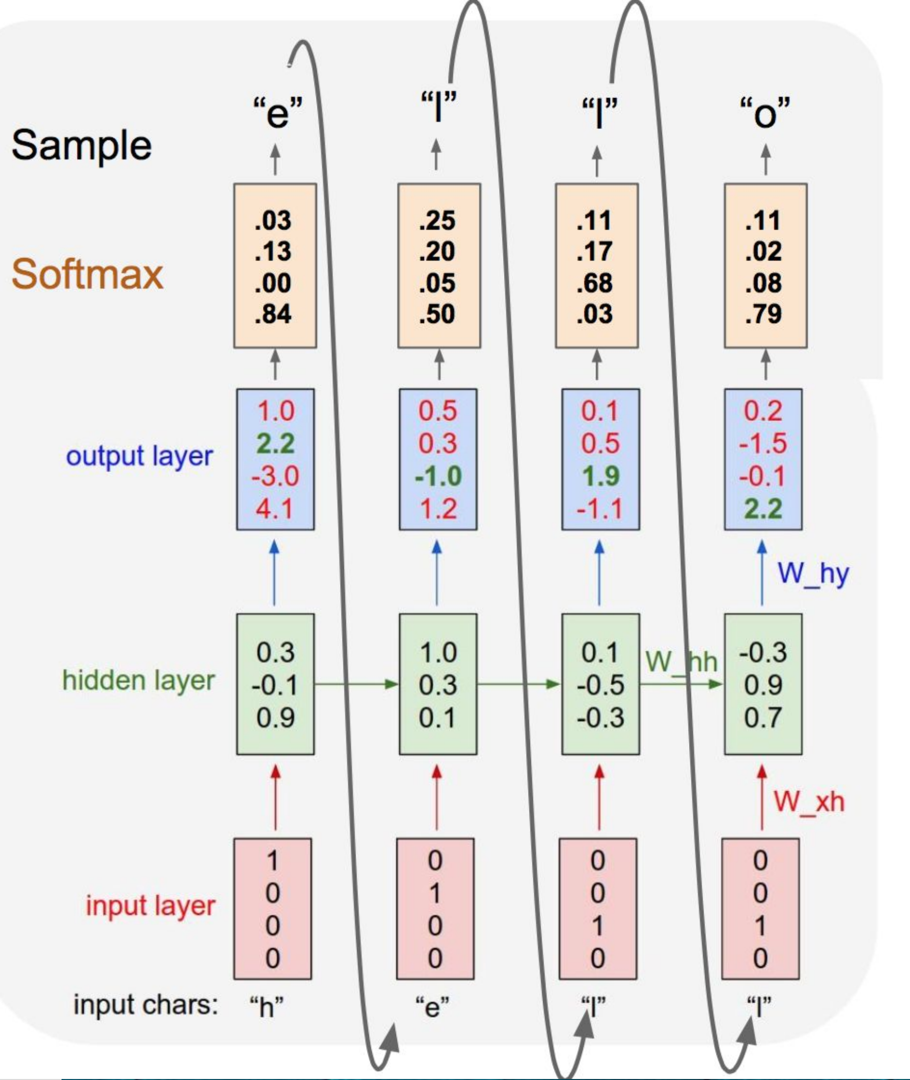
- **Input :** recipe text (recipe[:-1])
- **Target:** recipe text (recipe[1:])

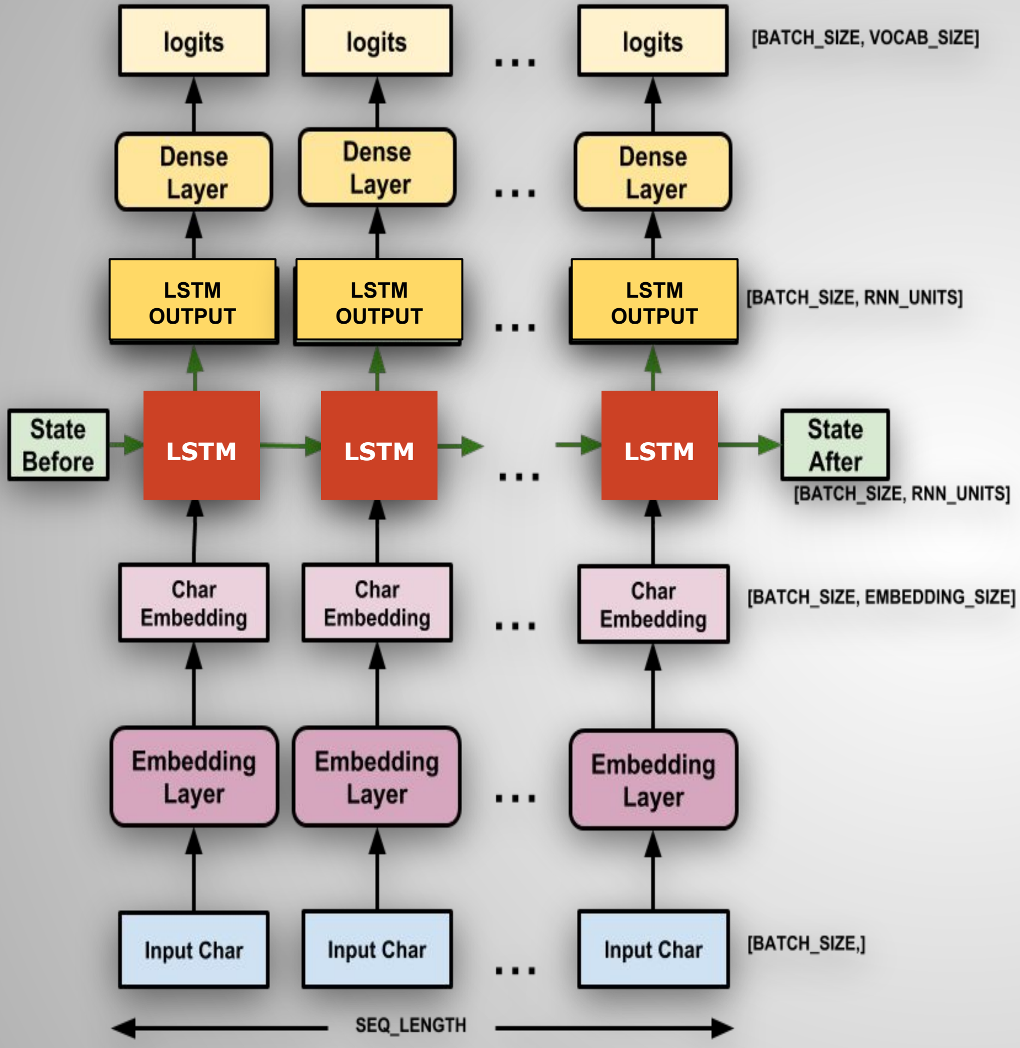


```
Model: "sequential_6"
_________________________________________________________________
__
Layer (type)                 Output Shape              Param #
=================================================================
==
embedding_6 (Embedding)      (1, None, 256)            45056
_________________________________________________________________
__
lstm_5 (LSTM)                (1, None, 1024)           5246976
_________________________________________________________________
__
dense_5 (Dense)              (1, None, 176)            180400
=================================================================
==
Total params: 5,472,432
Trainable params: 5,472,432
```

- **Optimizer:** Adam
- **Loss Function:**

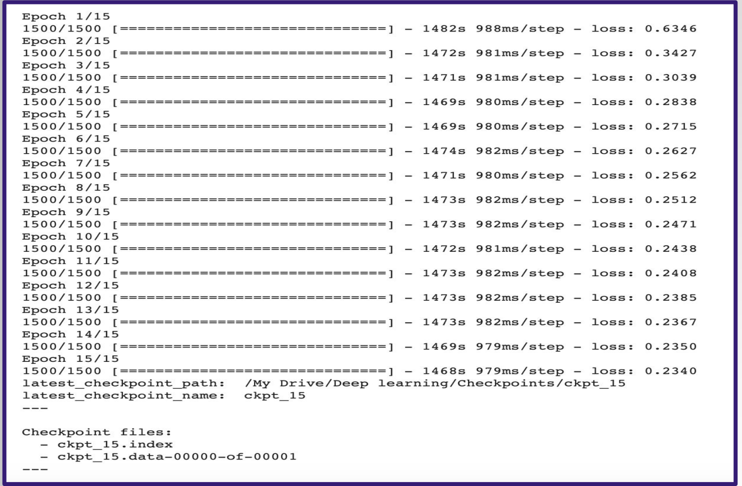
  Sparse_categorical_crossentropy()



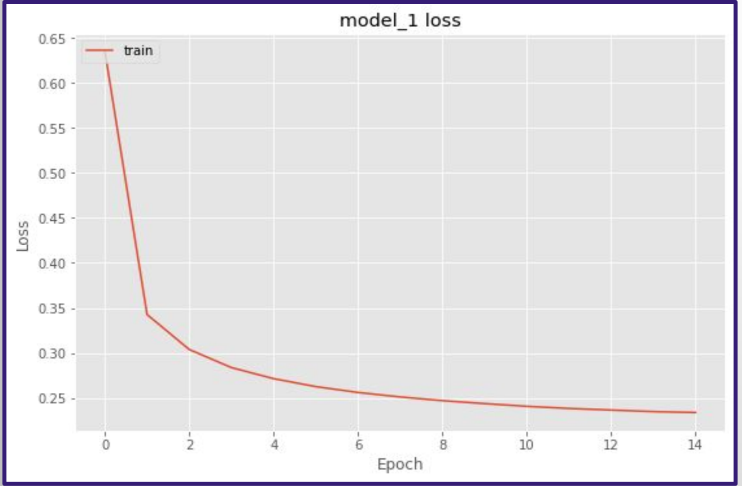


Model Description

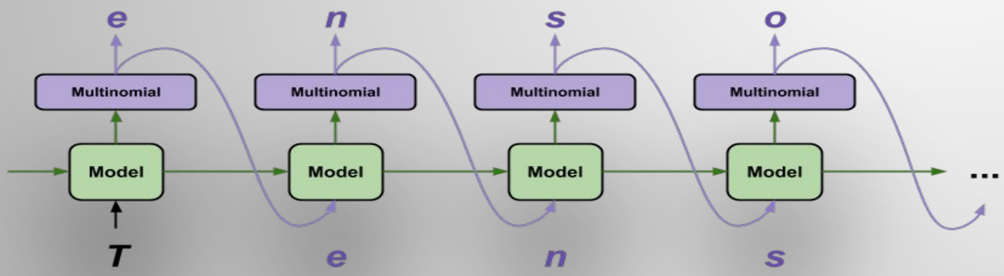
Training Results



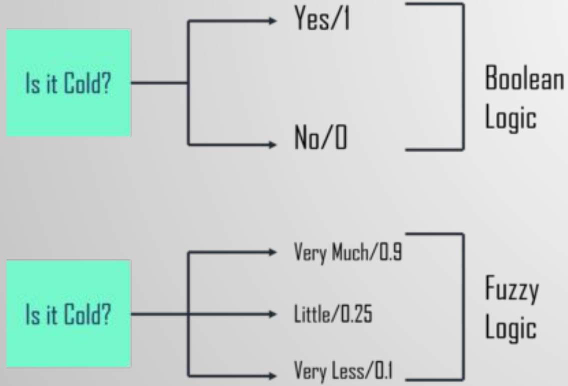
```
Epoch 1/15
1500/1500 [==============================] - 1482s 988ms/step - loss: 0.6346
Epoch 2/15
1500/1500 [==============================] - 1472s 981ms/step - loss: 0.3427
Epoch 3/15
1500/1500 [==============================] - 1471s 981ms/step - loss: 0.3039
Epoch 4/15
1500/1500 [==============================] - 1469s 980ms/step - loss: 0.2838
Epoch 5/15
1500/1500 [==============================] - 1469s 980ms/step - loss: 0.2715
Epoch 6/15
1500/1500 [==============================] - 1474s 982ms/step - loss: 0.2627
Epoch 7/15
1500/1500 [==============================] - 1471s 980ms/step - loss: 0.2562
Epoch 8/15
1500/1500 [==============================] - 1473s 982ms/step - loss: 0.2512
Epoch 9/15
1500/1500 [==============================] - 1473s 982ms/step - loss: 0.2471
Epoch 10/15
1500/1500 [==============================] - 1472s 981ms/step - loss: 0.2438
Epoch 11/15
1500/1500 [==============================] - 1473s 982ms/step - loss: 0.2408
Epoch 12/15
1500/1500 [==============================] - 1473s 982ms/step - loss: 0.2385
Epoch 13/15
1500/1500 [==============================] - 1473s 982ms/step - loss: 0.2367
Epoch 14/15
1500/1500 [==============================] - 1469s 979ms/step - loss: 0.2350
Epoch 15/15
1500/1500 [==============================] - 1468s 979ms/step - loss: 0.2340
latest_checkpoint_path:  /My Drive/Deep learning/Checkpoints/ckpt_15
latest_checkpoint_name:  ckpt_15
---

Checkpoint files:
  - ckpt_15.index
  - ckpt_15.data-00000-of-00001
---
```



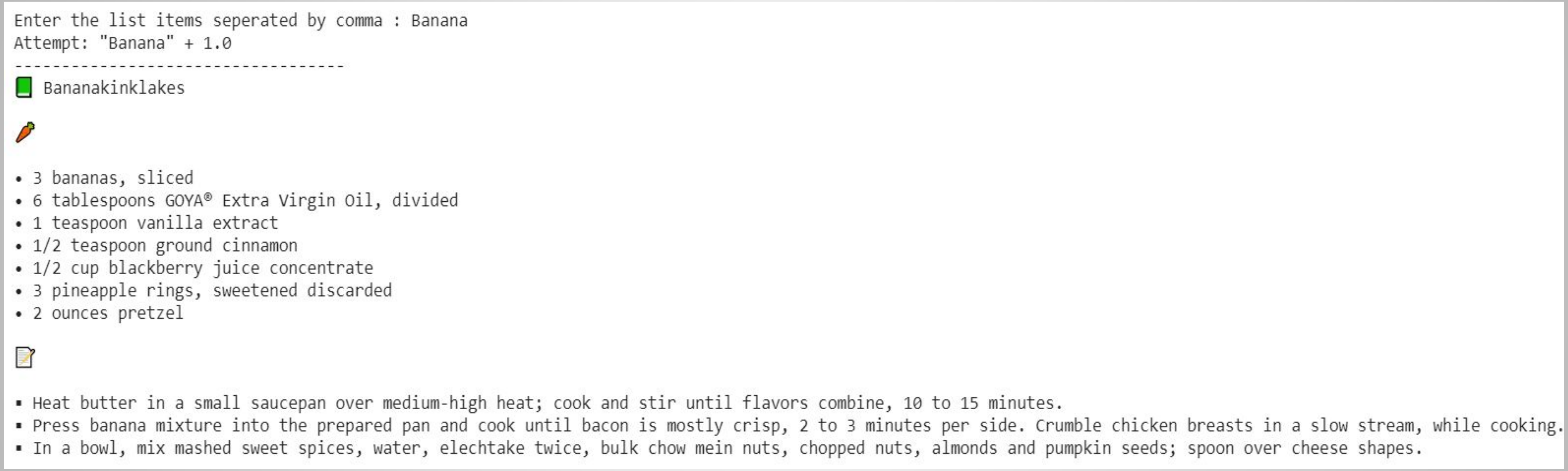
1. **Choose** a start string, initialize the RNN state and set the number of characters to generate.
2. **Obtain** the prediction distribution of the next character using the start string, and the RNN state.
3. **Calculate** the index of the predicted character. This predicted character is used as the next input to the model.
4. **Feed** the RNN state returned by the model back into the model so that it now has more context, instead of only one character.
5. **Modified** RNN states are again fed back into the model, which is how it learns as it gets more context from the previously predicted characters.



- Prediction array contains probabilities of what the next character might be.
- Usually, we pick the character with highest probability.
- Introduce Fuzziness / Randomness to the model.



Final Output & Demo



```
Enter the list items seperated by comma : Banana
Attempt: "Banana" + 1.0
-----------------------------------
📗 Bananakinklakes

🥕

• 3 bananas, sliced
• 6 tablespoons GOYA® Extra Virgin Oil, divided
• 1 teaspoon vanilla extract
• 1/2 teaspoon ground cinnamon
• 1/2 cup blackberry juice concentrate
• 3 pineapple rings, sweetened discarded
• 2 ounces pretzel

📝

▪ Heat butter in a small saucepan over medium-high heat; cook and stir until flavors combine, 10 to 15 minutes.
▪ Press banana mixture into the prepared pan and cook until bacon is mostly crisp, 2 to 3 minutes per side. Crumble chicken breasts in a slow stream, while cooking.
▪ In a bowl, mix mashed sweet spices, water, elechtake twice, bulk chow mein nuts, chopped nuts, almonds and pumpkin seeds; spoon over cheese shapes.
```

Learnings opportunities:

- Text generation
- Text preprocessing techniques like
  - Vectorizing Text
  - Tokenizing Text
  - Padding Text
- RNNs, LSTMs
- Fuzzy Logic

Future Work:

- Using a more sophisticated network structure (more LSTM, Dense Layers)
- Training for more epochs
- Playing around with the batch_size

Key Takeaways & Future Work

Thank You !!!