Identifying and Combining Ingredients from Natural Instructions

Allen Park
apark93@mit.edu

Abstract

We extract a set of simple actions from a list of natural instructions in an unsupervised fashion. To do that, we solve four sub-problems: classifying a set of instructions as a recipe or not, classifying a step as an ingredient list or not, classifying a word as an ingredient or not, and extracting actions from natural instructions with labeled ingredients. To solve these problems, we use an unsupervised EM-like algorithm to train an ingredient value for every word. This ingredient value indicates how likely the word is to be an ingredient. For each problem, we were able to demonstrate an improvement upon baselines and comparable performance with a supervised alternative.

https://github.mit.edu/apark93/6864Project

1 Introduction

To truly reach artificial intelligence, a lot of work is required from the field of natural language processing. One important step to a helpful artificial intelligence is the ability to produce results from a list of instructions. Normally for a computer, this list of instructions is given in code. However, code is a difficult way to give instructions that requires training. Code itself is slowly becoming more naturally readable, but the ultimate goal is to give instructions for any task in natural language and have artificial intelligence interpret those instructions.

Action	**Object**	**Modifier**
cut	onions	into small pieces
put	oil	into the pot
take	pot	from the stove

Table 1: Examples of parsed actions.

A few examples of actions that can be parsed from natural language are given in table 1. In order to be most useful, we tackle the problem of parsing actions from completely unannotated instructions. Specifically, this means that we parse instructions into actions from how the instructions were presented to other humans to interpret. These instructions should be written by humans for other humans to interpret and act upon.

In addition to parsing actions from unannotated instructions, we train our models with unannotated data. In other words, we approach this problem with an unsupervised approach, attempting to find a solution that does not require us to annotate large amounts of data.

In section 2, we give you an overview of our general approach with the data we use and the sub-problems we focus on. In section 3, we present the core of our unsupervised approach in assigning ingredient values to words. In sections 4, 5, 6, and 7, we present our results in the different sub-problems we tackle. Finally, in section 8, we reflect on our progress and conclude.

1.1 Definitions

Throughout this paper, we will use several terms that we would like to define explicitly here.

First, an *instruction set* is a list of instructions. These instructions are in natural language and were written for a human by another human.

This instruction set is made up of a series of *steps*. Steps are a subdivision of the instruction set into vaguely related components. The writer of the instruction set decides how to split the instruction set into steps.

Each step can contain any number of *actions*. Note that a step can possibly have 0 actions, if the writer of the instruction set decided to use a step for background information or other information. These actions are easily read by a computer. Examples are given in table 1.

In an instruction set, *ingredients* are used. Ingredients are any component that are used in the instructions. In table 1, the ingredients are 'onions', 'pieces', 'oil', 'pot', and 'stove'.

1.2 Previous Work

There is a large host of research on deriving a set of actions from a set of text instructions, some of which are listed below. Three highly relevant papers are Mise en Place: Unsupervised Interpretation of Instructional Recipes (Kiddon et al., 2015), Reinforcement Learning for Mapping Instructions to Actions (Branavan et al., 2009), and Learning High-Level Planning from Text (Branavan et al., 2012).

(Kiddon et al., 2015) attempted to map instructional recipes to action graphs using an unsupervised approach. They were able to achieve a F1 score of 66.8, with a precision of 68.7 and a recall of 65.0. (Branavan et al., 2009) trained a model for action selection based on textual commands with minimum or no training. With a partially supervised model, they were able to achieve an action-level accuracy of 72.3%. (Branavan et al., 2012) used relations derived from text to improve on a non-relationship-aware model, achieving a plan percentage accuracy of 80%.

The other papers below learn other aspects of the learning instruction problem. These other aspects include learning states and actions (Narasimhan et al., 2015), a dependency tree data structure for representing recipes and corresponding parser (Jermsurawong and Habash, 2015), a simple model for parsing (Babes-Vroman et al., 2012), and an environment model for filling in missing data (Branavan et al., 2010).

2 Approach Overview

Overall, we present a viable unsupervised approach to parsing actions from instructions. In this section, we first present the data sets that we used. Then, we talk about our approach to the problem and how we manage the different technical challenges.

2.1 Data

We used two kinds of data sets in this work. First, we needed instruction sets to parse into actions. Second, for this to be unsupervised learning, we needed lists of ingredients that could be used in recipes. We detail here where we obtained these data sources and the limitations from both of those data sets.

2.1.1 Instruction Sets

First, we obtained instruction sets from a website. This website can be found at www.instructables.com (Instructables,). On this website, users can submit their own do-it-yourself (DIY) plans for other users to browse. We simply scraped the website for the text of the DIY plans, which were separated into several steps by the users who wrote the plans. In total, we scraped 7005 instruction sets.

Since these DIY plans were written by casual users for other casual users, the quality of these plans varied widely. Although some plans were very professional and clearly written, other plans were not well written. There are typos, unclear instructions, non sequiturs, general statements, and even plans written in other languages. Furthermore, there are identifiable actions that do not have to do with the instruction set, e.g. "Please upvote my instructable!" which were fairly common. Because of these conditions, we consider these instructions significantly harder to parse than simple recipes commonly found in previous research.

2.1.2 Ingredient Lists

Second, we downloaded ingredient lists from two sources. The first source was from ESHA (ESHA,), a food labeling company. This source produced 8207 ingredients. The second source was from the Food Standards Agency in the UK (FSA,). This source produced 3424 ingredients. Examples of these ingredients include "Ground Flaxseed", "Whole Eggs Raw-Lrg", and "Seasoned Salt No MSG UNI-LA".

After combining the list of ingredients, separating by word, and removing duplicates, we obtained a list of 2152 words. From now on, we refer to these words as ingredients.

However, this preliminary list of ingredients included words that weren't necessarily ingredients, like "Ground", and words that weren't words at all, like "Lrg". We address this problem by completing an unsupervised training on natural text. The details are in section 3.

2.2 Sub-Problems

In order to make this problem approachable, we split the problem into a few different manageable sub-problems. We train and test each of these sub-

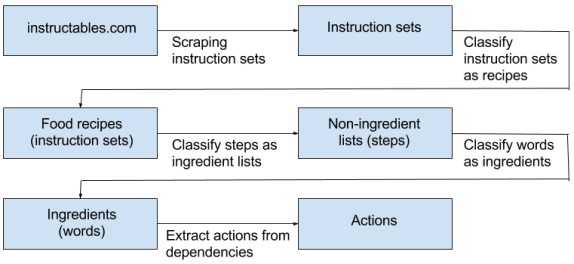

Figure 1: A diagram showing the flow between the different sub-problems. The sub-problems are shown as the links between states.

problems separately, and, as such, we present results for each sub-problem in their own section below. In this section, we will talk briefly about each sub-problem. Figure 1 shows the different sub-problems all together.

1. The first sub-problem is to classify instruction sets as food recipes or not. Since our data is based on food ingredients, we expect it to work only on recipes. However, many instruction sets that we have are not related to food. We will need to filter those out.

2. The second sub-problem is to classify steps as containing lists of ingredients or not. Parsing actions from steps will not work well on steps with lists of ingredients, since those lists will not have any sentence structure. Therefore, we will want to avoid those lists.

3. The third sub-problem is to classify words as ingredients or not. If we can classify words as ingredients, then parsing actions from steps will be significantly easier.

4. The fourth and final sub-problem is to parse actions from instructions given the ingredients in the instructions. Since we are given the ingredients to focus on, this will be a fairly simple problem of parsing the dependency tree.

The third and fourth sub-problems are arguably the more important of the four. The first and second are only for filtering for better results, but the latter two get the actual actions. Therefore, we focused on improving the last two sub-problems more.

To solve each of these problems, we first attempt to find a value for each word that reflects whether that word is an ingredient or not. This process is described in section 3.

3 Unsupervised Training of Ingredient Values

From straightforward analysis detailed in section 2.1.2, we have a preliminary list of ingredients. However, this preliminary list of ingredients includes words that are not always used as ingredients, like ”Ground” from ”Ground Salt” and ”Light” from ”Light Whipped Cream”.

In order to both refine the list of ingredients and to assign a numerical value to each ingredient, we do some unsupervised training to produce *ingredient values*. Ingredient values measure how likely a word is to be an ingredient or not. For example, ”Cheesecakes” will have a high ingredient value while ”Inductive” will have a low ingredient value.

For this problem, we used both the ingredient lists and the instruction sets. The ingredient lists were used as a initial estimate of whether a word was an ingredient or not. The instruction sets were used as examples of natural text.

3.1 Algorithm

The basic intuition behind the unsupervised training is that ingredients tend to co-occur only with other ingredients while non-ingredients also occur with non-ingredients as well as sometimes ingredients. We take advantage of this fact with an algorithm similar to an EM algorithm. The algorithm is listed below.

1. Initialize each word with an ingredient value of 0.5 if the word is in the ingredient list or 0 otherwise.

2. For each step, compute an average ingredient value over all words in the step. Call this the step value.

3. For each word, compute the average step value across all steps that the word appears in.

4. Update the ingredient values for each word with exponential smoothing.

5. Iterate.

Since ingredient values for ingredients would be higher than ingredient values for non-ingredients, this would mean that ingredient values for ingredients would decrease slower than ingredient values

for non-ingredients. That therefore makes ingredient values for ingredients even more higher than ingredient values for non-ingredients. The iteration emphasizes this difference.

In our experiments, we used a smoothing factor of 0.2 with 10 iterations. Since $0.8^{10} \approx 0.1$, that meant that our initial guess could only contribute to 0.1 of the final ingredient value. We also found that only accepting nouns and ignoring stop words as ingredients in our initialization improved our results.

3.2 Bigram Attempt

We also attempted this algorithm with bigrams instead of unigrams. This did not work very well due to data sparsity. The algorithm overfitted on the data we had since there were not enough bigrams in our data set to train with. We suspect that bigrams will not work well unless we have an extremely large dataset.

3.3 Qualitative Results

This algorithm worked well to separate ingredients from non-ingredients. We will detail in the next few sections how the ingredient values work quantitatively. For qualitative evaluation, we list some examples.

The words with the largest ingredient value are exotic food words, like ”tahini”, ”bulgur”, ”lassi”, ”edam”, ”bratwurst” and ”goulash”. This makes sense, since these words would be used exclusively with other ingredients. Furthermore, these words would also be used with other exotic ingredients, meaning that the average would be kept high.

At the bottom of the list are non-English words and mispelled words. Some of the instructables are not written in English. Since our initial ingredient lists were all in English, these non-English words would naturally not be identified as ingredients. The mispelled words similarly have no reason to commonly be found with ingredients.

Right above these last results are words that are commonly used in non-recipe instructions. Some of these words include ”shoebox”, ”legalese”, ”patents”, ”lugnuts”, and ”cardboard”.

4 Binary Classification of Instruction Sets as Recipes

In this sub-problem, we will classify instructions sets as either recipes or not recipes. Recipes would have to do with food, while not recipes would not. Since our ingredient list only deals with food, our parsing will only work well with recipes.

4.1 Baselines

We implemented three baselines. These baselines only use the starting ingredient list.

The first baseline looks at the number of words in the title that are in the starting ingredient list. The second baseline looks at the fraction of words in the title that are in the starting ingredient list. The third baseline looks at the fraction of words in the instruction text that are in the starting ingredient list.

We also tried each of these baselines with the trained ingredient values rather than a switch on being in the starting ingredient list, but those baselines with the trained ingredient values actually did worse than just with the starting ingredient list.

We also attempted to implement a supervised neural network, but it did not work very well. The solution that it found was to classify all instruction sets as recipes. Since the unsupervised training of ingredient values worked extremely well, we did not think it was important to have a supervised baseline.

4.2 Algorithm

The algorithm to classify recipes is actually very simple. It uses the ingredient values that we found in the unsupervised training. We list it below.

1. For each step in the instruction set, find the average ingredient value of all words in the step. Call this the step value.

2. If the step value is greater than a cutoff, classify it as a recipe. Otherwise, the instruction set is not a recipe.

In practice, we found that a cutoff of 0.061 works best. We also attempted this with ingredient bigrams, which works best with a cutoff of 0.064.

4.3 Results

We evaluated our baselines and algorithm on a set of 6830 instruction sets. Of these, 1276 were annotated as being recipes and 5554 were annotated as being not recipes.

We show the results in table 2. As shown, the ingredient values with unigrams get a stunning F1

6830 Instructables	Food words in titles >= 2	Food fraction in titles >= 0.34	Food fraction in steps >= 0.1	Ingredient unigram values >= 0.061	Ingredient bigram values >= 0.064
Accuracy	0.8730	0.8347	0.9685	0.9836	0.9602
Precision	0.6528	0.9497	0.8958	0.9482	0.8580
Recall	0.6800	0.1187	0.9402	0.9646	0.9427
F1	0.6661	0.2110	0.9175	0.9563	0.8984

Table 2: Results of classifying instruction sets as recipes or not.

score of 0.95. This is a very high F1 with an unsupervised approach that improves upon all of the baselines.

We also attempted to use bigram ingredient values. They did not work as well as the unigram ingredient values. We think that this is because we do not have enough data. We would probably need massive amounts of data for any bigram to work well.

5 Binary Classification of Steps as Ingredient Lists

In this sub-problem, we classify steps within instruction sets as containing ingredient lists or not containing ingredient lists. Our parsing depends on dependency parsing, which does not work well on lists. Therefore, we would like to avoid attempting to parse on steps that do contain ingredient lists.

This sub-problem is slightly challenging because of the nature of our data. On instructables.com, users often but do not always include a list of ingredients. Even when they do, they may embed that list of ingredients in with other description of the task. In addition, the list of ingredients may not come as an actual list but as a general description in text of what is needed. The large variety of ingredient lists makes this a challenging and not well-defined problem.

5.1 Baselines

We implemented four baselines and a supervised neural network. The baselines do not use any of the unsupervised training that we implemented.

The first baseline just looks for the presence of a list tag, as used in the html. The second, third, and fourth baselines look at the fractions of nouns, verbs, and numbers respectively in the step text.

The supervised neural network is the supervised alternative to the unsupervised approach that we use here. The neural network is a multi-layer perceptron. As features, we use the presence of a list tag and the fractions and counts of all parts of speech tags. This is a total of 73 features.

5.2 Algorithm

Again, the algorithm is rather simple. We use the ingredient values that we obtained from the unsupervised training.

1. For each word in the step, compute the tanh of the ingredient value of the word.
2. Compute the average over all words of the tanh of the ingredient values.
3. If the average is greater than a cutoff, then the step is an ingredient list. Otherwise, it is not.

We found that a good cutoff is 0.5. In this case, we found that using a tanh activation for the ingredient values worked better than just the ingredient values. This makes sense because it prevents high ingredient values from raising the average too much.

5.3 Results

We evaluated our baselines and algorithm on a set of 429 steps. Of these, 49 steps were ingredient lists and 380 steps were not ingredient lists. Since this sub-problem was not critical to the success of our overall problem, we chose not to annotate more steps.

We show in table 3 the results of our baselines and algorithm. With our unsupervised method, we were able to improve upon all of the attempted baselines. Although we were not able to get as close to the supervised method F1 score as we would have liked, the simplicity and speed of our simple unsupervised classifier definitely wins over that of the supervised method.

6 Binary Classification of Words as Ingredients

In this sub-problem, we classify words as ingredients or not ingredients. This sub-problem is criti-

429 steps	List tag presence Baseline	Num. fraction <= 0.05 Baseline	Noun fraction <= 0.36 Baseline	Verb fraction >= 0.11 Baseline	Multi-layer perceptron Supervised	Simple classifier >= 0.5 Unsupervised
Accuracy	0.8997	0.8671	0.8787	0.8414	0.9457	0.8764
Precision	0.6363	0.4230	0.4666	0.3855	0.64133	0.4655
Recall	0.2857	0.4489	0.4285	0.6530	0.6154	0.5510
F1	0.3943	0.4356	0.4468	0.4848	0.6281	0.5046

Table 3: Results of classifying steps as ingredient lists or not.

cal for the overall problem to work, since our next sub-problem, parsing from instruction sets with labeled ingredients, does not work if the ingredients are not labeled.

6.1 Baselines

We implemented two baselines and a supervised neural network. The two baselines do not use the ingredient values from our unsupervised training and only use properties of the inherent word and the starting ingredient list.

The first baseline sees if the word in question is in the starting ingredient list. The second baseline sees if the word in question is in the starting ingredient list and is a noun.

Again, the supervised neural network is the supervised alternative to the unsupervised approach that we use here. The neural network is a multilayer perceptron. As features, we use word vectors for the trigram centered around the word, the ingredient value of the word, the average ingredient value of the step, the average ingredient value of the instruction set, features of the word like if it's alphabetical, if it has punctuation, if it's lowercase, etc., and a one-hot vector for the part of speech tag. This is a total of 213 features.

6.2 Algorithm

Again, the algorithm is extremely simple. We use the ingredient values from the unsupervised training.

1. Test if the ingredient value of the word is greater than a cutoff.
2. If it is, then the word is an ingredient. Otherwise, it is not.

We found that a good cutoff is 0.097.

In addition to this simple classifer, we also attempted to train a neural network. We used the results of the simple classifier in order to train a neural network. Because the simple classifier is unsupervised, the neural network is in fact unsupervised as well. The neural network we used is exactly the same as the supervised neural network we implemented as a baseline. The only difference is that the training dataset comes from the simple classifier results rather than annotated data.

6.3 Results

We evaluated the baselines and our algorithm on a set of 1112 words. Of these words, 233 words were labeled as ingredients and 879 words were labeled as not ingredients.

We show the results in table 4. As shown, both our simple classifier and our unsupervised neural network outperform our baselines.

Interestingly, training a neural network on the simple classifier actually increases the F1 score. This implies that the "mistakes" that the neural network makes in classifying against the simple classifier results are actually correcting the mistakes that the simple classifier makes.

As expected, neither unsupervised approach could reach the F1 score of the supervised approach. However, the unsupervised neural network had a very close F1 score of 0.75 compared to the F1 score of 0.77 of the supervised neural network. Considering that the supervised network required much more human annotation and the unsupervised neural network did not, this is a significant accomplishment.

7 Parsing Actions from Instruction Sets with Labeled Ingredients

Our last sub-problem is to finally parse actions from instructions. In this sub-problem, we assume that ingredients are already labeled as such, and so our task is simply to parse actions from the sentences that involve the ingredients.

7.1 Algorithm

Since we already have ingredients that we should focus on, we just need to extract some information

1112 words	Food word Baseline	Food noun Baseline	Multi-layer perceptron Supervised	Simple classifier >= 0.97 Unsupervised	Multi-layer perceptron Unsupervised
Accuracy	0.7913	0.8812	0.9004	0.8862	0.8810
Precision	0.5	0.7451	0.7254	0.7313	0.6623
Recall	0.7543	0.6681	0.8222	0.7101	0.8879
F1	0.6013	0.7045	0.7708	0.7205	0.7587

Table 4: Results from classifying words as ingredients or not.

from the dependency parsing.

1. First, we obtain the dependency parsing and construct a tree.

2. Sort each step with a topological sort. The roots of the sentences should come first, and the leaves of the tree should be last.

3. Loop through the words in each step in topological order. If the word has already been processed or the word has not been annotated as an ingredient, skip it. The word is then an unprocessed ingredient.

4. Loop from the root to the ingredient in focus, looking at each dependency along the way. This will allow us to find the largest sentence fragment that uses the ingredient.

5. Complete the parsing corresponding to the dependency type:

   (a) Direct object: The action is at the head of the current dependency.

   (b) Preposition: The action is at the closest verb that is a head of the current dependency.

   (c) Nominal subject: The action is at the head of the current dependency.

   (d) Nominal passive subject: The action is at the closest passive auxiliary that is a head of the current dependency.

   (e) Indirect object: The action is at the closest direct object that is a head of the current dependency.

6. The action is the pair of the verb and the set of all ingredients that are in the corresponding subtree.

7. Mark all ingredients involved in the action as processed and proceed to the next word in topological order.

This algorithm only requires simple processing of the dependency parsing. The casework in this algorithm is incomplete, since this is a well-researched problem. However, this shows a basic version of the algorithm that can process the most common elements.

7.2 Results

We first show an example of the results. We process the step shown below.

```
First cut your onions into small
pieces, then cut your garlic
cloves into very fine cubes. Put
some oil in your pot and roast
the onions and garlic until
golden-brown. Take the pot from
the stove.
```

Verb	Text	Labeled Ingredients
cut	your onions	onions
cut	into small pieces	pieces
cut	your garlic cloves	garlic, cloves
cut	into very fine cubes	cubes
Put	some oil	oil
Put	in your pot	pot
roast	the onions and garlic	onions, garlic
Take	the pot	pot
Take	from the stove	stove

Table 5: Actions resulting from the above step.

In addition to these results, we can also parse general statements. For example, the sentence ”Here is the point where you can add whatever other ingredient you might like.” is correctly parsed as commentary.

Because we only parse sentences that contain ingredients, we have a much smaller chance of false positives. For example, the sentence ”Please like my instructable!” is correctly ignored.

However, our system is sensitive to dependency parsing errors. For example, in the sentence ”Pour

with tomato sauce and serve fresh.", the word "pour" is incorrectly parsed as a noun. Therefore, the sentence incorrectly has "serve" as the root of the sentence, meaning that the sentence will be incorrectly parsed.

In addition, the dependency parsing is sensitive to mispellings and punctuation errors. Both "Tak[e] some salt off" and "Leave the pot, chop the vegetables" will be incorrectly parsed.

Finally, the system is incapable of disambiguating pronouns and references. Since all of these errors (handling dependency parsing errors, detecting mispellings and punctuation errors, and disambiguation) are both all out of the scope of this project and would increase errors substantially, we chose not to evaluate our system quantitatively. However, we can see from the qualitative results that the parsing does work well in the right context.

8 Conclusion

We were able to have significant progress with a unsupervised method versus a supervised baseline. We first completed unsupervised training of ingredient values with an EM-like algorithm. These ingredient values helped us with classifying instruction sets as recipes, classifying steps as ingredient lists, and classifying words as ingredients. Finally, we produced a basic parser that extracted actions from a dependency parsing of the instructions.

There are three natural next steps to this work. First, the whole system should be evaluated together. We did not do this in this work since that would require a massive amount of annotation. Many more instruction sets would have to be fully annotated, and there can be up to 300 words in a single instruction set.

Second, we should look at training an ingredient vector, similar to the ingredient values. Somewhat surprisingly, only a single value was required to represent if a word was an ingredient or not. We only trained one value and got appreciable results. Training multiple values in a vector could possibly obtain much more.

Third, we would want to extend this work to non-food instruction sets. We started with lists of food ingredients, but if we can start with lists of hardware components, for example, then we can see if the same methodology would work well.

Our unsupervised approaches to all of these problems were able to approach a perfect performance or the supervised alternative, which should be able to do strictly better.

My code can be found at (Park,).

References

Monica Babes-Vroman, James MacGlashan, Ruoyuan Gao, Kevin Winner, Richard Adjogah, Marie desJardins, Michael Littman, and Smaranda Muresan. 2012. Learning to interpret natural language instructions. In *Proceedings of the Second Workshop on Semantic Interpretation in an Actionable Context*, pages 1–6, Montr{'eal, June. Association for Computational Linguistics.

S.R.K. Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90, Suntec, Singapore, August. Association for Computational Linguistics.

S.R.K. Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1268–1277, Uppsala, Sweden, July. Association for Computational Linguistics.

S. R. K. Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. 2012. Learning high-level planning from text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 126–135, Stroudsburg, PA, USA. Association for Computational Linguistics.

ESHA. Esha databases. `http://www.esha.com/solutions/additional-databases/`.

FSA. Fsa dataset. `http://tna.europarchive.org/20110116113217/http://www.food.gov.uk/science/dietarysurveys/dietsurveys/`.

Instructables. Instructables. `www.instructables.com`.

Jermsak Jermsurawong and Nizar Habash. 2015. Predicting the structure of cooking recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 781–786, Lisbon, Portugal, September. Association for Computational Linguistics.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages

982–992, Lisbon, Portugal, September. Association for Computational Linguistics.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal, September. Association for Computational Linguistics.

Allen Park. My code. `https://github.mit.edu/apark93/6864Project`.