Modeling Cooking Tutorials using Hidden Markov Models Robot Learning from Demonstration

Aishwarya Padmakumar Akanksha Saran

***Abstract*— With the advancement of personal robotics, learning from demonstration has become a popular technique to teach robots new tasks. However, for a robot to adapt different tasks and environments, it is not feasible for end-users to provide demonstrations for all possible tasks and scenarios. Instead if robots could leverage the large amount of instructional data available on the web to perform tasks, such as video and text tutorials, the amount of effort required by end-users to teach robots could be minimized. In this project, we look at the problem of recovering the latent structure present in textual instructions sets for cooking recipes - specifically peanut butter and jelly sandwich and chocolate cake recipes. We manually collect a set of 20 recipes for each recipe type and train Hidden Markov Models to recover the latent steps generating these recipes. We use four different types of HMMs which vary based on the language models used for computing the probabilities of textual observations given the latent state. We discover that a single HMM is not able to capture all desired properties of a model assumed to be generating recipes from a set of latent steps - representativeness, discrimination and alignment.**

I. Introduction

As robotic technology is advancing, its potential to impact lives of ordinary people has increased. Specifically, personal robots which can be deployed in people's homes could be used to help out in daily chores like doing laundry or cooking. However in order for a robot to adapt to any home, it must be reprogrammable by end-users to learn new tasks and adapt to different contexts. Programming by Demonstration or Learning from Demonstration[1] is one possible way to realize this goal. It can be tedious for end-users to give multiple reasonable demonstrations if they are not familiar with a platform or do not very well understand the kinematics of the robot. Thereby, it would be desirable to make end-users do the minimum amount of work to teach robots how to perform a task by taking advantage of the large amounts information available on the web in the form of text and videos. The data available online can be used to ground a single demonstration an end-user provides, to structured/semantic concepts. This is the motivation of our project and for the purpose of this work, we focus on the task of extracting semantic structure for cooking recipes using natural language tutorials available online, which can potentially be combined with a single end-user demonstration to teach recipes to a robot.

The research problem to be answered for the purpose of the project is 'given a list of instructions for a task from different sources, are there a set of "latent steps" - a sequence of operations in some latent space, that generate all these instruction sets'. Different tutorials may phrase instructions differently, skip or combine steps. The goal of our project is to identify such a latent representation which would appropriately map tutorials of similar and different tasks (recipes in our case) so that tutorials of the same task can be identified by good alignment. A latent space may not strictly be necessary for the alignment of two texts but by using such a representation, the technique is more likely to be conducive to alignment with tutorials in other modalities such as video or physical demonstrations.

To extract the latent states for text tutorials, we assume that a generative model like a Hidden Markov Model (HMM) can be used to model a recipe. This has a natural interpretation in that hidden states correspond to latent steps from which textual steps (observations) are generated. The probability of generating a textual observation given a latent state is modelled using a language models. Thus, each hidden state has a separate language model. In this work, we also present and evaluate different types of language models which could be used for modeling probability of observations.

II. Related Work

The final goal of this project is to enable a robot to use tutorials downloaded from the web in lieu of demonstrations. This is similar to the work of Beetz et al [2] where robots download a tutorial for making pancakes from the web, and generate robot action plans to make pancakes using these. We would like to eventually achieve a similar goal except that we are aiming for a more general technique that does not require much task-specific knowledge and can scale to a library of tasks. Another robotic system that performs cooking tasks is the Bakebot by Bollini et al [3] which is a PR2 that can bake cookies. However, the knowledge of the recipe is hard-coded in this case.

The works of Malmaud et al [7] and Sener et al [13] are also related to our project. The former attempts to align recipe text to the speech transcript of a video and hence to the video itself, enabling a robot to make use both textual and visual instruction. Our work is complementary to this and could be used to fill in gaps in the aligned recipe text from another tutorial that is more detailed in that aspect. The work of Sener et al aims to parse a collection of videos into semantically meaningful steps in an unsupervised manner using both visual information and language information from subtitles. They also generate a textual description of the step using information from subtitles. This is similar to our

work, except is inthe domain of vision rather than language. Another related work in recipe understanding is that of Tenorth et al [14] which converts recipes into logical forms using semantic parsing and use these to plan for robots. However, they too depend on a single recipe to provide all necessary information. Another related task is that of script learning [11] which attempts to learn a representation for a sequence of events that allows logical reasoning over them.

In the NLP community, there are many works that use representations that could be useful for aligning text. Macine Translation involves the aligning of parallel corpora at the word level [6], typically using latent variable models or intermediate representations. There is also work that explicitly model paraphrasing of the same content, such as the FSA models of paraphrasing by Pang et al [10]. Multi-document text summarization also involves integrating information from multiple documents into a coherent whole [4].

III. Background

A. HMMs

A Hidden Markov Model (HMM) is a doubly embedded stochastic process with an underlying stochastic process that is hidden and can be observed only through another set of stochastic processes that produce the sequence of observations [12]. In an HMM, in order to explain a sequence of observations, we assume each observation was generated from some hidden state and the sequence of hidden states is Markov. There are three important problems associated with an HMM

- Estimation problem - Calculate the probability that an observation sequence o was generated by the HMM
- Identify the sequence of hidden states most likely to have generated a given observation sequence
- Training problem - Optimize the parameters of the HMM to maximize the likelihood of generating some given observation sequences

These are respectively solved by the Forward-Backward, Viterbi and Baum-Welch algorithms.

Consider an HMM with n hidden states, $S_1, S_2, \ldots S_n$. Let q_t be the hidden state at time t, A_{ij} be the probability of transitioning from state S_i to state S_j, $b(o, s)$ be a function that returns the probability of observation o given state s, π_i be the probability that the first hidden state $q_1 = S_i$. Then for each observation sequence $o^{(k)}$, a forward variable $\alpha_t^k(i) = Pr(o_1^{(k)}, o_2^{(k)} \ldots o_t^{(k)}, q_t = S_i/A, b, \pi)$ and a backward variable $\beta_t^k(i) = Pr(o_{t+1}^{(k)}, o_{t+2}^{(k)} \ldots o_{T_k}^{(k)}, q_t = S_i/A, b, \pi)$. These can be calculated recursively as,

$$\begin{aligned} \alpha_1^k(i) &= \pi_i b_i^k(o_1) \\ \alpha_{t+1}^k(j) &= b_j(o_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{ij} \end{aligned}$$

$$\begin{aligned} \beta_T(i) &= 1 \\ \beta_t(i) &= \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \end{aligned}$$

Also, the probability of observing the sequence O^k, given model parameters is

$$P_k = \sum_{i=1}^{N} \alpha_T^k(i) \tag{1}$$

Two more variables are used to train the HMM. These are

$$\begin{aligned} \xi_t^k(i,j) &= Pr(q_t = S_i, q_{t+1} = S_j / o^k, A, b, \pi) \\ &= \frac{\alpha_t^k(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}^k(j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_t^k(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}^k(j)} \\ \gamma_t^k(i) &= Pr(q_t = S_i / o^k, A, b, \pi) \\ &= \sum_{j=1}^{n} \xi_t^k(i,j) \end{aligned}$$

Then, the update rules for an iteration of the Baum-Welch algorithm are

$$\begin{aligned} \bar{a}_{ij} &= \sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_{k-1}} \alpha_t^k(i) a_{ij} b_j(o_{t+1}^k) \beta_{t+1}^k(j) \\ \bar{\pi}_i &= \frac{1}{K} \sum k = 1^K \gamma_1^k(i) \end{aligned}$$

The probability function b will also in general be parameterized and the update rules for these parameters are specific to each parameterization.

B. Language Models

Language models are probabilistic models that attempt to assign a probability value to any sentence in a language[5]. They commonly do this by assigning a probability to seeing a word given some of the previous words. One of the most commonly used type of language models is the N-gram model. This model assumes that

$$Pr(w_k / w_{k-1}, w_{k-2}, \ldots w_1) = Pr(w_k / w_{k-1}, w_{k-2}, \ldots w_{k-n-1})$$

Some special cases are the unigram and bigram models. Given a sentence $s = (w_1, w_2, \ldots, w_n)$, the unigram probability of s is

$$Pr(s) = Pr(\langle s \rangle) * Pr(w_1) * Pr(w_2) * \ldots * Pr(\langle / s \rangle)$$

and the bigram probability of s is

$$\begin{aligned} Pr(s) &= Pr(w_1 / \langle s \rangle) * Pr(w_2 / w_1) * \ldots \\ &\quad * Pr(\langle / s \rangle / w_s) \end{aligned}$$

where $\langle s \rangle$ and $\langle / s \rangle$ are special tokens indicating the start and end of a sentence respectively.

While modelling language, it is possible to encounter words not seen in the training data. Estimating the above probabilities using counts of unigrams and bigrams seen in a training corpus would lead to sentences with a new word

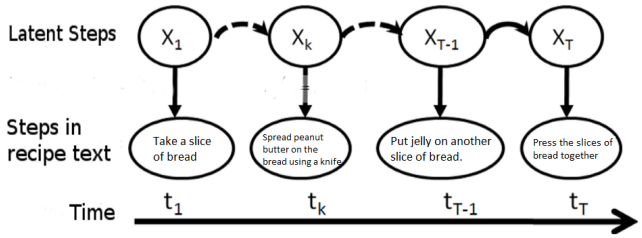

Fig. 1. HMM Model for natural language tutorials.

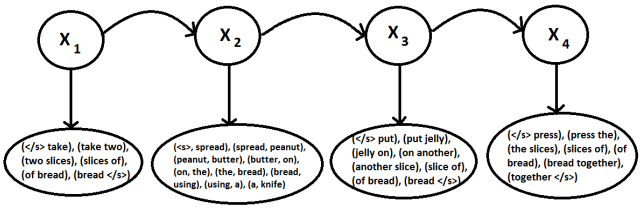

Fig. 2. HMM using all bigrams in a sentence as observations.

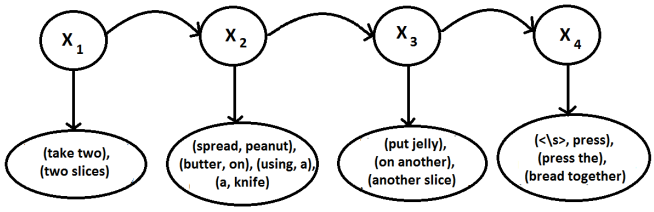

Fig. 3. HMM using bigrams in a sentence as observations except the common bigrams present in the vocabulary considered as background

being assigned a probability of zero. One way to avoid this is to replace all unknown words by a special token $-UNK-$ and assigning some probability to $-UNK-$ and normalizing during training [5]. There are more principle techniques such as smoothing to deal with this issue but applying them to a small corpus heavily skews the probabilities of known words.

IV. Approach

To model tutorials, we hypothesize that to perform a task, there is some canonical sequence of steps in a latent space that describe how it is done. Tutorials are generated from this canonical sequence using primarily superficial linguistic transformations. This idea is naturally modeled by an HMM where hidden states correspond to latent steps and observations from a latent step are sentences of tutorials. An example can be seen in Fig. 1.

To complete this generative model, we need to define the transition and observation probabilities of the HMM. The transition probabilities are modelled in a standard manner using a transition matrix. No special constraints are placed on transitions. The observation model of a state must capture the probability of seeing a sentence of a tutorial given a latent step (probability of an observation given a latent state). This is a standard problem in Natural Language Processing and the models used for this task are language models [5]. We compare 4 different language models in their ability to model tutorials. For each of these, we need to derive update rules for the model parameters to be used in the Baum Welch algorithm. The language models along with their respective update rules are as follows.

A. Bigram

This is a standard bigram model [5]. A bigram model estimates the probability of a sentence by assuming that the probability of seeing a word fully depends on the word seen just before. This model maintains the following parameters

- $uni[j][w]$ - Unigram count of the word w in hidden state j - Number of times w is seen in a sentence generated by the state j
- $bi[j][(w_1, w_2)]$ - Bigram count of the bigram (w_1, w_2) in hidden state j - Number of times w_2 occurs just after w_1 in a sentence generated by the state j

Then, given a sentence $s = (w_1, w_2, \ldots w_s)$, we have,

$$\begin{aligned} Pr(s/j) &= Pr(w_1/\langle s\rangle, j) * Pr(w_2/w_1, j) * \ldots \\ &\quad * Pr(\langle/s\rangle/w_s, j) \\ &= \frac{bi[j][(\langle s\rangle, w_1)]}{uni[j][\langle s\rangle]} * \frac{bi[j][(w_1, w_2)]}{uni[j][w_1]} * \ldots \\ &\quad * \frac{bi[j][(w_s, \langle/s\rangle)]}{uni[j][w_s]} \end{aligned} \tag{2}$$

An example can be seen in Fig. 2. The probability of the sentence is the product of the probabilities of all individual bigrams.

The parameters are re-estimated from observations as follows. Let $c(o, w_1, w_2)$ be the number of occurrences of the bigram (w_1, w_2) in observation o. Then,

$$bi[j][(w_1, w_2)] = \sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t(j) c(o^k, w_1, w_2) \tag{3}$$

We add a small probability for out-of-vocabulary words and then normalize this so that $\sum_{w'} bi[j][(w, w')] = 1 \ \forall \ j \in \{1, 2, \ldots n\}, w \in V$ where V is the vocabulary. Then,

$$uni[j][w] = \sum_{w'} bi[j][(w, w')] \tag{4}$$

B. Background

The bigram model gives equal importance to every token in the tutorial text. However, there are a number of words such as articles that occur in every latent step and hence it is not useful to consider them separately. To ignore these, we model a background distribution and subtract this out when we use the observation probabilities in the HMM. The parameters for this are $uni[j][w]$ and $bi[j][(w, w')]$ for all hidden states

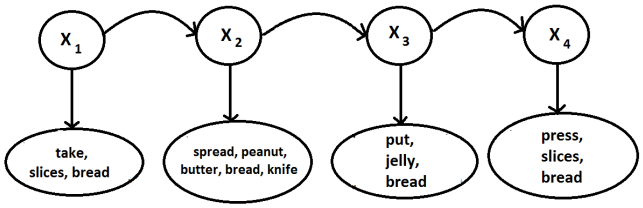

Fig. 4. HMM using verb-noun unigrams in a sentence as observations.

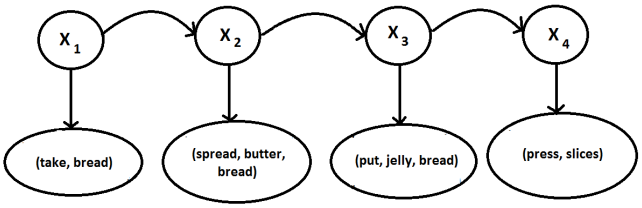

Fig. 5. HMM using verb-noun tuples in a sentence as observations.

j and all words w and w' in the vocabulary as in the bigram model and additional parameters $uni[w]$ and $bi[(w, w')]$ for all words w and w' in the vocabulary. Then, given a sentence $s = (w_1, w_2, \ldots w_s)$, we have,

$$\begin{aligned} Pr(s/j) &= Pr(w_1/\langle s\rangle, j) * Pr(w_2/w_1, j) * \ldots \\ &\quad * Pr(\langle/s\rangle/w_s, j) \\ &= h\left(\frac{bi[j][(\langle s\rangle, w_1)]}{uni[j][\langle s\rangle]}, \frac{bi[(\langle s\rangle, w_1)]}{uni[\langle s\rangle]}\right) \\ &\quad * h\left(\frac{bi[j][(w_1, w_2)]}{uni[j][w_1]}, \frac{bi[(w_1, w_2)]}{uni[w_1]}\right) * \ldots \\ &\quad \ldots * h\left(\frac{bi[j][(w_s, \langle s\rangle)]}{uni[j][w_s]}, \frac{bi[(w_s, \langle s\rangle)]}{uni[w_s]}\right) \end{aligned}$$

where $h(x, x') = \begin{cases} x - x' & \text{if } x - x' > 0 \\ 0 & \text{otherwise} \end{cases}$

An example can be seen in Fig. 3. It is similar to the bigram model but the probability is calculated as if the observation did not incude the bigrams that are common across states.

The parameters $uni[j][w]$ and $bi[j][(w, w')]$ are updated as before. The other parameters are updated as follows

$$bi[(w, w')] = \sum_j bi[j][(w, w')] \quad (5)$$

$$uni[w] = \sum_j uni[j][w] \quad (6)$$

C. Verb-noun unigram

On examining tutorials, we can see that the superfluous words in the text are primarily adverbs or adjectives. It appears that looking at only the verbs and nouns of the sentence would be better at identifying the crux of the sentence. This model capitalizes on this idea as follows. Given a sentence s, let $w_1, w_2, \ldots w_s$ be the verbs and nouns in the sentence,

$$Pr(s/j) = uni[j][w_1] * uni[j][w_2] * \ldots * uni[j][w_s] \quad (7)$$

An example can be seen in Fig. 4. Now, the probability of a sentence is the product of the probabilities of each verb or noun in it. The parameters $uni[j][w]$ are updated as in the previous models.

D. Verb-noun tuple

The previous model can be further refined by only considering the nouns that are the objects of the verbs in the sentence. In this model, the observations are not sentences. Each sentence is broken into tuples of verbs with the objects the act upon. Such a tuple is then the observation from a latent state. An example can be seen in Fig. 5. A major difference between this and the previous model is that is ignores nouns not essential to the action, for example, it only considers what is being spread and on what for the verb spread. How it is spread is not important. The probability of an observation o given state j is then just $uni[j][o]$ which is updated as in the previous models, except that the keys of $uni[j]$ are now tuples instead of individual words.

V. Experiments

Modeling cooking tutorials using Hidden Markov Models has a natural interpretation that hidden states correspond to latent steps from which textual steps (observations) are generated. The probability of generating a textual observation given a latent state is modelled using a separate language model. To evaluate the effectiveness of our language models for the task of extracting a underlying latent structure from tutorials, we perform experiments to test the following three hypotheses:

- Representation - The underlying latent steps modeled by the HMMs should represent all tutorials of the task on which it was trained
- Discrimination - The underlying latent steps modeled by the HMMs should be able to determine whether a new tutorial corresponds to the task the HMM was trained on
- Alignment - The most likely sequence of latent states obtained from the HMM for recipes can be used to align them in a useful manner.

We use a dataset of 20 tutorials each for 2 recipes - (i) Peanut Butter and Jelly Sandwich, (ii) Chocolate Cake. To train each HMM model pertaining to a recipe, we use 10 recipes. The remaining 10 recipes are used for testing. Prior to training, we pre-process each tutorial using the Stanford CoreNLP suite [8] to provide a generalized form of input to our model. Primarily, we convert the text of tutorials into lower case, perform tokenization and lemmatization using part-of-speech tags, and remove non-ASCII characters which appear in HTML. The preprocessing is aimed at avoiding

a bias to the way information is presented and allows the algorithm to focus more on the generic content. To get the verb-object combinations for the verb-noun unigram and verb-noun tuple models, we again use the Stanford Parser [8] to preprocess our data. For each sentence in the recipe, the head verb is identified from its parse tree and the object(s) it acts upon. We thus convert a recipe into a sequence of verb-object tuples. For training our discrete hidden markov models (Baum Welch algorithm) we used publicly available code [1].

A. Representativeness

To test the representative power of the proposed models, we compute the negative log likelihood of the test data, which is a measure of how likely is the data to be generated by the model. We compare the four proposed models as shown in Fig. 6. A lower value of negative log likelihood depicts that a recipe was more likely to be generated by a particular HMM. We also vary the number of latent states of the HMM to get a sense of how many latent states are sufficient to represent a recipe. For further analysis, we choose 4 latent states to represent the peanut butter and jelly sandwich recipes and 6 steps to represent the chocolate recipes. Overall, the Verb-Noun tuple HMM performs the best uniformly across different number of states. It intuitively captures the most important words of an instruction step such as 'spread' and 'jelly'. The Bigram model performs the worst since it captures a lot of background noise, focusing on nothing specific in a sentence.

B. Discrimination

To test how well a Hidden Markov model discriminates tutorials of its own type versus others, we compare the log likelihoods of our recipe test data using all eight models (4 HMM types trained on 2 recipe classes). We discovered that none of the models can discriminate very well between the two recipe classes. We show the data plot for log-likelihoods of all the test recipes for the verb-noun tuple HMMs (Fig. 7), verb-noun unigram HMMs (Fig. 8), background HMM (Fig. 9) and bigram HMM (Fig. 10). We find that the models trained on the chocolate cake recipes are slightly more discriminative than the one trained on peanut butter and jelly. Overall, there was not a clear linear boundary between the likelihoods of the two recipe classes across any of the HMMs. One reason why we see such an outcome is because the model trained on one recipe class does not learn from negative data of the other class. Hence, a single HMM for each recipe type may be sufficient to represent recipes of its own type but not for classifying different recipes.

C. Alignment

For the first alignment metric, test recipes were manually annotated using a canonical sequence of latent steps. Each recipe has an associated sequence of states from the ground truth annotation and the HMM. For all possible pairwise matchings between ground truth states and HMM states, we calculate edit distance between the mapped sequences and then choose the minimum such edit distance and normalize it by the length of the recipe. This metric is a measure of similarity between the semantics of ground truth and HMM states.

[1] https://github.com/guyz/HMM

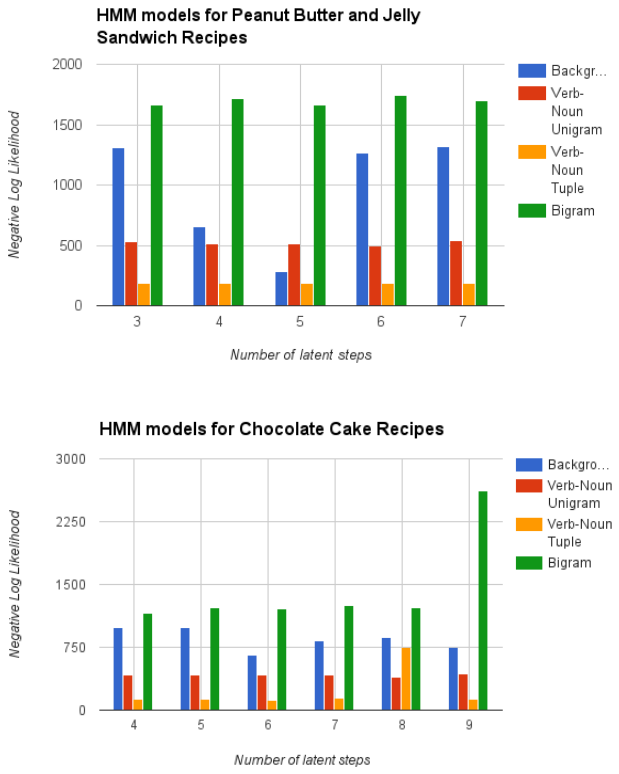

Fig. 6. Evaluation of the representative power of our HMM language models.

For the second alignment metric, edit distance between two recipes is calculated for the sequence of states from the ground truth annotation and the output of the HMM. Absolute difference of these two edit distances is used as a measure of alignment. A lower value indicates that the HMM aligns recipes to the same extent as in the ground truth. Hence a lower value is better. Annotation semantics for latent states can vary across users. This metric is less sensitive to such variation than the previous metric.

We show alignment results using both metrics for all the proposed HMM types in Fig. 11. Firstly, the second metric provides consistent results across the four proposed HMMs, whereas the first one does not. It inherently captures the pairwise similarity between recipe steps generated by the HMM. On the other hand, the first metric is more biased towards the interpretation of the annotator. We find that the verb-noun tuple HMM performs the worst among all others despite capturing the important concepts of the verb and noun referred to in a latent step. One reason for this is the inaccuracies of the semantic parser in perfectly providing verbs

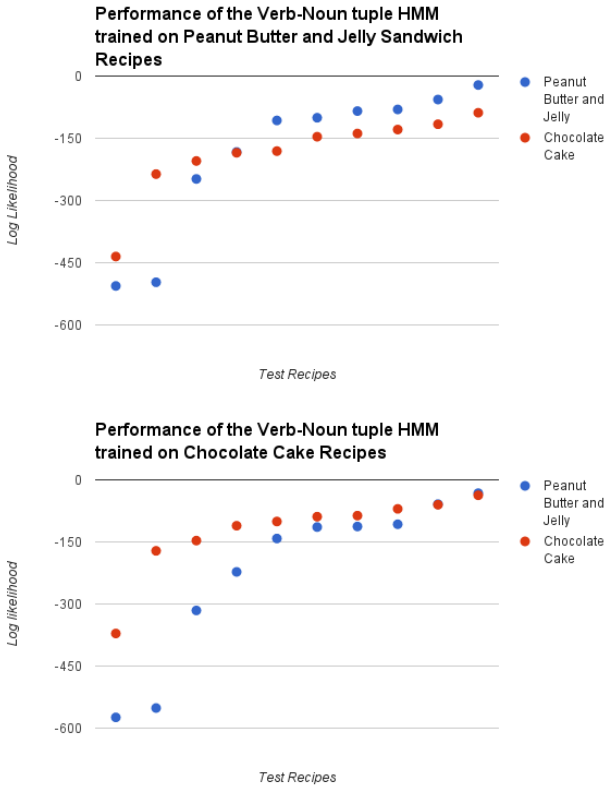

Fig. 7. Evaluation of the discriminative power of our Verb-Noun Tuple HMM language models.

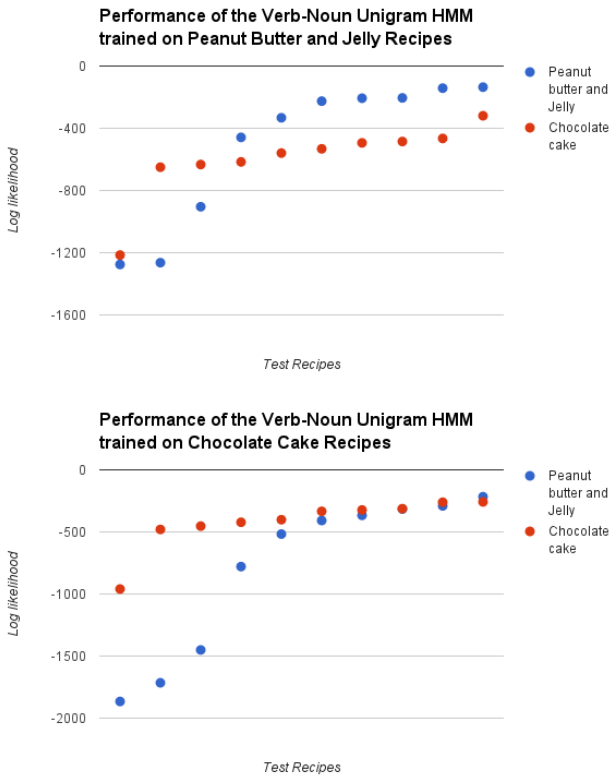

Fig. 8. Evaluation of the discriminative power of our Verb-Noun Unigram HMM language models.

and associated nouns from the instruction steps. Overall, the verb-noun unigram model and the bigram model does better at alignment. Surprisingly, the more complex chocolate cake recipes are better-aligned than the simpler peanut-butter and jelly sandwich recipes.

However, on examining the resultant sequence of hidden states, we found that often, most of the recipe steps get the same state or follow some pattern that does not really correspond to any human-understandable semantics. Some example case can be seen in tables I and II. Such state sequences in fact allows the HMMs to score fairly highly on the second metric used for alignment because now the edit distance of HMM states between any two recipes becomes almost equal to the difference of their lengths. Since the edit distance in ground truth is close to this, the HMM scores highly despite having not performed any useful alignment. This gives the impression that the HMM is not really learning much useful information and the metrics are not clear enough to indicate this.

VI. Discussion

From our experimental results, we find that no one model can capture all the desired properties of representativeness, discrimination and coherent alignment. Verb-noun tuple HMM is the most representative model for both Peanut Butter Jelly sandwich recipes and Chocolate Cake Recipes, however none of the HMM models proposed are very discriminative. Since HMMs are a generative model, our results show that HMMs alone are probably not well-suited primarily for classification. For aligning tutorials of the same task together, the Verb-noun tuple HMM does not do as well as expected. However the verb-noun unigram model and bigram models perform relatively better. One limitation of our approach is the lack of a large dataset for training and testing. It may not be sufficient to work with a small dataset for natural language tutorials as training a good language model generally requires a large training corpus. Also, HMMs cannot learn from negative examples during training which is possibly why they are poor at discrimination. Moreover, annotations for ground truth are provided by only one annotator which might have heavily biased the results towards the annotator's preferences and interpretations, thereby unable to capture the possible variations in the interpretation of latent states across a larger set of people.

Further, the latent states in the HMMs do not appear to have the desired semantics, as could be seen by the patterns of state assignment. In the case of the background, verb-noun unigram and verb-noun tuple HMMs, there is possibly insufficient information to distinguish between states. Also, the Stanford CoreNLP suite [8] in not always accurate in

HMM state - Bigram	HMM state - Background	HMM state Verb-noun unigram	HMM state Verb-noun tuple	Ground Truth State	Recipe text
3	2	1	1	1	Open the bread and remove two slices , laying them side by side .
2	2	1	3	2	Open peanut butter jar .
1	2	1	3	2	Remove some peanut butter with the knife .
1	2	1	3	2	Spread the peanut butter onto the bread until you have as much as you want , removing more from the jar as needed .
1	2	1	3	3	Open jelly jar .
1	2	1	3	3	Remove a table spoonful of jelly .
1	2	1	3	3	Spread the jelly on the bread until you have as much as you want , removing more from the jar as needed .
1	1	1	3	4	Place the two slices together , peanut butter and jelly facing inwards , press down slightly , and enjoy

TABLE I

AN EXAMPLE OF HMM STATES ASSIGNED TO A PEANUT BUTTER AND JELLY SANDWICH RECIPE

HMM state - Bigram	HMM state - Background	HMM state Verb-noun unigram	HMM state Verb-noun tuple	Ground Truth State	Recipe text
3	1	2	1	1	Preheat an oven to 325 F.
3	1	2	1	3	Fill a saucepan with water, bring it to a boil, turn off the heat and rest the bowl on the inside rim of the saucepan, just above the height of the water.
1	0	3	2	3	This ensures that the chocolate does not get overheated.
0	1	0	0	3	Boil the water using a saucepan on the stove and then whisk the chocolate into the boiling water. The chocolate must still be warm in order prevent it from seizing.
2	0	1	1	3	Place the room-temperature butter and sugar into a mixing bowl.
0	1	3	2	3	Using a paddle attachment, turn the mixer to medium speed and cream until the mixture becomes light and fluffy.
0	0	0	0	3	This process of aeration is key to creating a light and tender final product.
0	1	1	1	2	The butter will cream more easily if it is soft and at room-temperature.
0	0	3	2	2	Be sure to occasionally scape down the sides of the bowl to ensure that the ingredients are being incorporated evenly.
2	1	0	0	3	Gradually add the egg yolks and vanilla and beat until fully incorporated.
0	0	1	1	3	Adding eggs a few at a time helps prevent the batter from separating.
2	1	0	2	3	Room temperature eggs also aid in creating a more cohesive batter.
0	5	1	0	3	Beat in the melted chocolate.
0	2	0	1	3	Sift all the dry ingredients (flour, soda, powder, and salt).
0	4	1	2	3	Add the dry ingredients and the sour cream alternately into the batter.
0	0	0	0	3	To do this, add one-third of the dry ingredients and then about half of the sour cream.
0	1	1	1	3	Repeat this until all of the dry ingredients and the sour cream has been incorporated evenly.
0	0	3	2	3	This gradual and alternating addition process allows the batter to accept a larger quantities of liquid, creating a moister and more delicious final result.
0	1	0	0	3	Beat the batter until smooth, but be careful not to over-mix.
2	0	1	1	4	Prepare baking pans with pan spray and divide the batter evenly.
0	1	0	2	4	Bake until the cakes spring back slightly to the touch.
2	0	1	0	5	Baking times will vary depending on the individual ovens and a number of environmental elements, so it is best to use your senses of touch, smell and sight, rather than rely on a timer.

TABLE II

AN EXAMPLE OF HMM STATES ASSIGNED TO A CHOCOLATE CAKE RECIPE

detecting the verbs and nouns from a sentence which has a domino effect on the performance of the verb-noun tuple and verb-noun unigram HMMs. For the bigram HMM, however, we are not sure why that pattern emerges. Recipes found for the same dish online vary considerably in details such as order of independent tasks, choice of words, level of detail and even in the use of some ingredients. It is not clear that an HMM can model all this.

Another aspect of the problem is that learning a model for

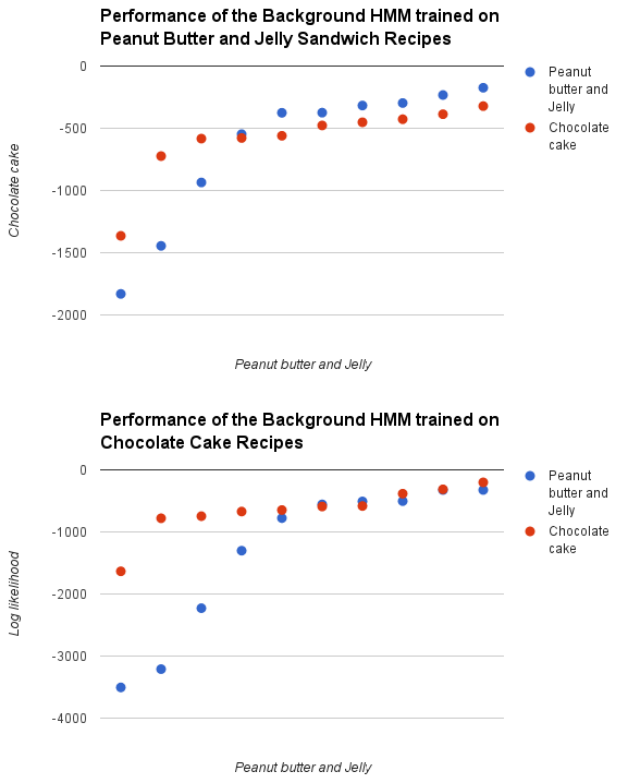

Fig. 9. Evaluation of the discriminative power of our Background HMM language models.

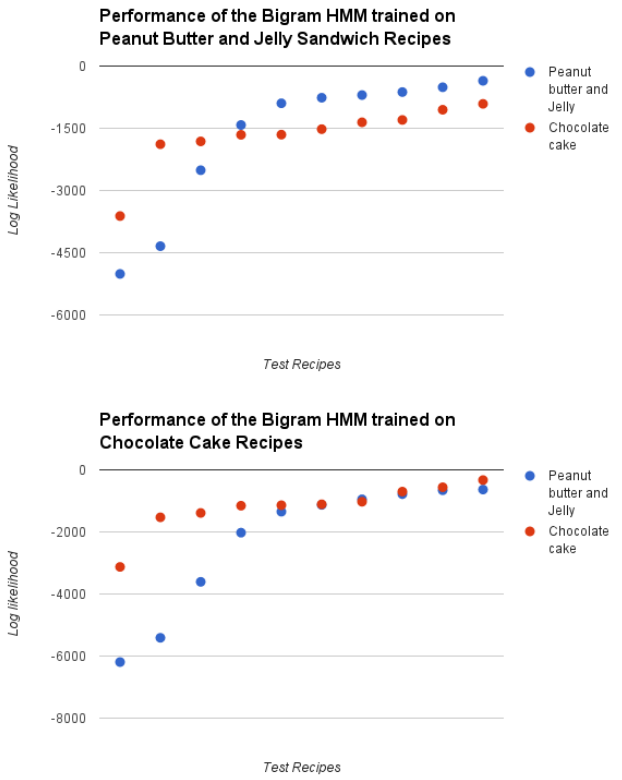

Fig. 10. Evaluation of the discriminative power of our Bigram HMM language models.

each task individually is probably not very scalable. On one hand, it is difficult to obtain a reasonable amount of data to train something like a language model. Further, a robot that is expected to be versatile may need to learn too many individual models. One possibility is to canonicalize primitive actions with object arguments that a robot can perform and attempt to convert textual recipes to such a form. This would then be similar to script learning or information extraction.

VII. Future Work

For future work, it may be more beneficial to work with a hierarchical model with a CRF on top or an approach like topic modeling to discriminate between recipes and learn an HMM for each category at the next level. Alignment with tutorials in other modalities such as video or physical demonstrations may be needed to recover semantically more meaningful hidden states.

Another direction of future work is tutorial"understanding. This involves identifying sequences of robot actions for each step obtained from aligned tutorials. This will enable a robot to complete a task using a tutorial. We could also integrate this with videos using ideas from works such as that of Malmaud et al [7]. Changes in visual information such as appearance of red jelly or yellow peanut butter on a bread may be more starking features than semantic information in natural language. It is possible that mapping to canonical robot actions will be easier using the joint visual and language input. Extensions to the changepoint detection algorithm [9] can also provide interesting insights to recover latent steps for instructional activities such as cooking or assembling furniture.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[2] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth. Robotic roommates making pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2011), Bled, Slovenia, October 26-28, 2011*, pages 529–536, 2011.

[3] M. Bollini, J. Barry, and D. Rus. Bakebot: Baking Cookies with the PR2. In *International Conference on Intelligent Robots and Systems (IROS) PR2 Workshop*, Sept. 2011.

[4] S. Gholamrezazadeh, M. A. Salehi, and B. Gholamzadeh. A comprehensive survey on text summarization systems. *Proceedings of CSA*, 9:1–6, 2009.

[5] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, chapter N-grams. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[6] A. Lopez. Statistical machine translation. *ACM Comput. Surv.*, 40(3):8:1–8:49, Aug. 2008.

[7] J. Malmaud, J. Huang, V. Rathod, N. Johnston, A. Rabinovich, and K. Murphy. What's cookin'? interpreting cooking videos using text, speech and vision. *arXiv preprint arXiv:1503.01558*, 2015.

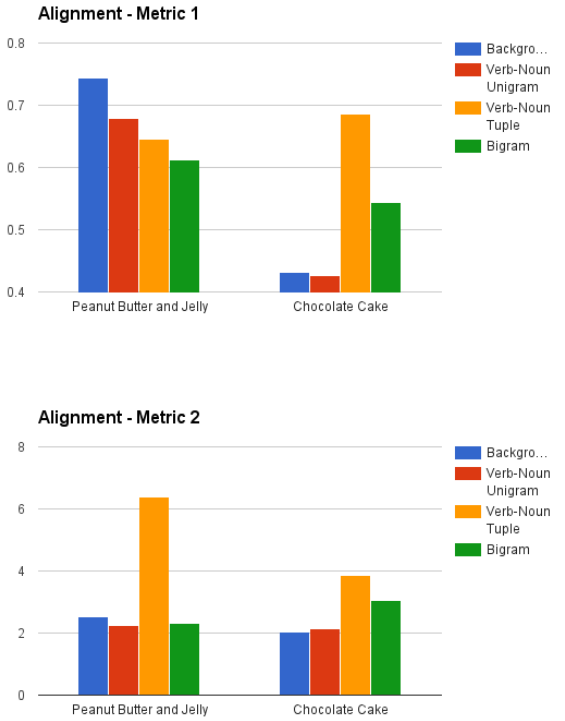

Fig. 11. Evaluation of alignment of recipes using our four proposed HMM types.A lower value of both alignment metrics is better. Values lie in the range [0, 1].

[8] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[9] S. Niekum, S. Osentoski, C. G. Atkeson, and A. G. Barto. Online bayesian changepoint detection for articulated motion models. In *submitted to) IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[10] B. Pang, K. Knight, and D. Marcu. Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 102–109, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[11] K. Pichotta and R. J. Mooney. Statistical script learning with multi-argument events. *EACL 2014*, page 220, 2014.

[12] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[13] O. Sener, A. Zamir, S. Savarese, and A. Saxena. Unsupervised semantic parsing of video collections. *arXiv preprint arXiv:1506.08438*, 2015.

[14] M. Tenorth, D. Nyga, and M. Beetz. Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1486–1491, Anchorage, AK, USA, May 3–8 2010.