Learning Action Preconditions from Step-by-step Intructions in Planning Domains

Maurício Steinert
Master Degree Program in Computer Science - School of Technology
Pontifical Catholic University of Rio Grande do Sul - PUCRS
Porto Alegre, Brazil
mauricio.steinert@edu.pucrs.br

***Abstract*— Automated Planning devises plans to achieve a specific goal. A planning domain must be provided in a formal language like Planning Domain Definition Language (PDDL) to be processed into a computer. Learning domain description from unstructured raw data is desirable. In this work we propose using natural language sentences to describe a planning domain, given a sequence of step-by-step instructions to accomplish a task. Specifically, we perform experiments first to identify actions in sentences using word embeddings similarity with predefined keywords. Next, we train a neural network to learn if exist relationship between ordered step-by-step instructions in natural language format.**

I. Introduction

Automated Planning devises plans to achieve a specific goal. Given computers are only able to understand formal languages, we must translate a domain description and problems into a formal language like PDDL. Hence, that requires that users of automated planning solvers learn these formal languages.

Researchers investigate using unstructured raw data as input to formalize a planning domain. LatPlan [1] is a planning domain solver that uses images as input so computers can identify state transitions and actions to devise plans based only on that information. Framer [5] and StoryFramer [4] are solutions that accept text as input and translate it into PDDL, however they require intense interaction with users to fill in the blanks and to eliminate redundancies.

Language modeling inside a computer is a challenging task, where approaches vary from using indexed dictionaries (Bag-Of-Words) to vector representation (embeddings) capable of capturing semantic relationship between words based on vicinity [6]. Many tasks like question-answering and text summarization attained good results using word and sentence embeddings [3].

In this work, we first automate actions identification within sentences based on word similarities. To compute similarity between words, we use word embeddings representation and compare each word within a sentence with a set of predefined keywords that are common action names in planning tasks. Second, we train a Recurrent Neural Network to verify if a sequence of step-by-step instructions respect possible preconditions between steps. This text is organized as follows: Section II details our dataset generated from WikiHow website, embedding dictionaries, followed by our methods for action identification task within sentences and precondition's evaluation between step-by-step instructions. Section III details our experiments and results. Section V concludes this text and discuss future research possibilities.

II. Method

In this section describes Wikihow dataset and embedding dictionaries, followed by our actions identification and sentence preconditions evaluation methods.

A. Dataset

To perform our experiments, our first step is building a dataset that fits our problem at hand. Given that we are interested in working with natural language input to describe planning domains, we need a large source of information with step-by-step instructions on how to perform tasks. Thus, we use WikiHow[1] as data source to generate a dataset. Wikihow articles are divided in categories and, in our case, we choose to work with cooking recipes' category.

Analyzing Wikihow pages, we observed that recipes share a similar data structure:

- The first sentence of each step summarizes the step, where the following sentences refine how to perform that step.
- More complex tasks are divided into sections, where each section is a set with independent steps. For example, making pasta recipe have a section on how to cook pasta and another section on sauces' preparation in the same recipe.

fill a large pot about 2/3 full of water.
cover the pot and bring the water to a boil.
add salt and 1 pound (450 g) of pasta to the boiling water.
set a timer for 3 to 8 minutes.
stir the noodles occasionally as they boil.
bite into a noodle to see if it's cooked enough for you.
scoop out about 1 cup (240 ml) of pasta water and set it aside.
set a colander in the sink and put on oven mitts.
pour the pasta into the colander and shake it.
avoid adding oil or running cold water over the pasta if you plan on using sauce.
put the pasta back into the pot and toss it with your choice of sauce.
toss short noodles with pesto or vegetables.
mix cheese into macaroni or shells to make a creamy pasta.
serve meaty sauce over tubular or wide pasta.

TABLE I
Instance extracted from Wikihow dataset

[1]https://www.wikihow.com/

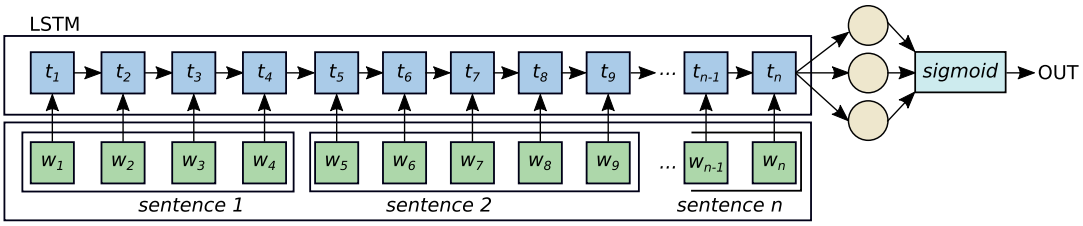

Fig. 1. Recurrent Neural Network many-to-one architecture.

To generate this dataset, we developed a script that download HTML documents from WikiHow based on a text file with URLs of categories pages. After that, we preprocess these documents by identifying instructions steps. For our experiments, we only use the first sentence of each step. Table I shows an instance extracted from this dataset. After preprocessing, we have a total of $5,519$ recipes with an average of 13.75 steps per recipe, each step with an average of 8.74 words. In Table II shows top 10 most frequent verbs in all dataset.

Verb	Frequency
add	8,305
serve	3,024
place	2,976
remove	2,650
cook	2,490
make	2,292
pour	2,257
cut	2,186
mix	2,168
stir	2,128

TABLE II
TOP 10 MOST FREQUENT VERBS IN DATASET

B. Embedding dictionaries

Embedding dictionaries represent words by a *n*-dimensional vector trained through an unsupervised learning method based on words vicinity over large volumes of raw text. We use *word2vec* [6] and *sent2vec* [7] to generate dictionaries with words and sentences embedding respectively, training over Wikipedia text corpus. *word2vec* dictionary contains 2,969,018 words and *sent2vec* dictionary contains 2,809,163 words. *word2vec* returns $\vec{v} \in \mathbb{R}^{100}$ for each word in dictionary, and *sent2vec* computes $\vec{v} \in \mathbb{R}^{100}$ for a whole sentence.

C. Action Identification task

Action Identification consists of identifying potential actions names within sentences. Given that natural language verbs are used to describe actions, our objective is to identify verbs and report them as actions.

Thus, we use word embedding [6] representation of words to identify potential verbs by comparing each word within a sentence with a set of predefined keywords that are commonly used as actions names. Equation 1 details how we compute similarity $\mathcal{S}$ between a candidate word and keywords, where $\mathcal{K}$ is a set of keywords, $\vec{v}_i \in \mathcal{K}$ is the vector representation of keyword i, $\vec{v}_{candidate\_word}$ is the vector representation of candidate word and distance returns values between 0 and 1 of how far two vectors are from each other. Given that we are interested in similarity, we subtract distance from 1, *i.e.* if both words are equal, we have distance equals 0 and consequently similarity equals 1.

$$\mathcal{S} = \frac{1}{|\mathcal{K}|} \sum_{i}^{\mathcal{K}} 1 - distance(\vec{v}_i, \vec{v}_{candidateword}) \qquad (1)$$

There are many libraries and dictionaries broadly available that are capable of identifying classes of words like nouns, verbs, adjectives and so on. However, we consider our method relevant because it works totally in an unsupervised manner: it only requires a word embedding dictionary, that is also trained without supervision, a set of few keywords for similarity comparison, and a similarity threshold value.

D. Step preconditions evaluation

Tasks in natural language are usually described as a series of step-by-step instructions that must be performed to accomplish a goal. These instructions most of time must be performed in provided order because there are a relationship among them. Examining Wikihow dataset instance in Table I, it is clear that step 2 *"cover the pot and bring the water to boil"* requires that step 1 *"fill a large pot about 2/3 full of water"* be performed first. If we give a closer look to this instance, we observe that there are steps that do not have any explicit relationship with previous steps and could be performed isolated from others like *"set a timer for 3 to 8 minutes"*, however performing this step in isolation or out of order would not help to reach a goal.

For this experiment, we train a Recurrent Neural Network (RNN) to classify if preconditions between steps are met. However, we evaluate an instance as a whole, without identifying relationship between specific steps. This RNN (Figure 1) is composed by Long-Short Term Memory (LSTM) layers that receive as input an encoded sequence of steps for a whole instance. The output of last LSTM time step n is connected to a fully-connected layer with one output, and over this one output we apply Sigmoid (Equation 2) activation function to keep output values between 0 and 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad (2)$$

III. Experiments

This section describes our experiments[2] in detail and present our attained results. We performed all experiments over Wikihow dataset and using embedding dictionaries described earlier. For each experiment we report precision $\mathcal{P}$, recall $\mathcal{R}$ and F1-score $\mathcal{F}$, as defined by Equations 3, 4 e 5, where $\mathcal{T}_s$ is a set of values identified by our method and $\mathcal{T}_g$ is a set of ground-truth values.

$$\mathcal{P} = \frac{\mathcal{T}_s \cap \mathcal{T}_g}{\mathcal{T}_s} \tag{3}$$

$$\mathcal{R} = \frac{\mathcal{T}_s \cap \mathcal{T}_g}{\mathcal{T}_g} \tag{4}$$

$$\mathcal{F} = 2 * \frac{\mathcal{P} * \mathcal{R}}{\mathcal{P} + \mathcal{R}} \tag{5}$$

A. Action Identification

As said earlier, action identification evaluates actions (verbs) within sentences given similarity of sentences words with a set of predefined keywords. To evaluate our methods performance, first we generate ground-truth values using Spacy[3] library that is capable of identifying words classes.

Next, we define our keyword set containing the following words: `take`, `put`, `drop`, `give`, `walk`, `move`, `pick`, `jump`, `eat`, `wait`, `shake`, `pull`, `push`. We avoid reusing most frequent verbs in dataset (Table II) to make this task harder to perform. For similarity threshold we use values 0.3, 0.4 and 0.5.

Table III summarizes our results, where we observe that similarity threshold 0.3 we have the best recall value, *i.e.* a larger number of relevant verbs successfully retrieved, but presents low precision. Similarity threshold 0.4 have the best precision and F1 scores, and thresholds 0.5 and 0.6 have lower values for all metrics.

similarity threshold	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{F}$
0.3	0.2734	0.8719	0.3931
0.4	**0.4958**	**0.5449**	**0.4951**
0.5	0.2628	0.1944	0.2132
0.6	0.0571	0.0378	0.0433

TABLE III
ACTION IDENTIFICATION RESULTS

For qualitative analysis, we store top 10 most frequent words identified as verbs for each similarity threshold (Table IV). For low similarity thresholds (0.3 and 0.4) we observe a high number of misclassified words, as already observed in precision score in Table III. As similarity threshold increases, the number of false positives decreases, and we do not have any non-verb word in top 10 table.

[2]available at https://github.com/mauriciosteinert/nlp_planning
[3]https://www.spacy.io/

Similarity 0.3		Similarity 0.4		Similarity 0.5		Similarity 0.6	
Verb	Freq	Verb	Freq	Verb	Freq	Verb	Freq
the	69,970	to	15,701	**remove**	**2,666**	**put**	**1,747**
and	23,261	**add**	**8,353**	**cut**	**2,337**	**bring**	**889**
a	20,139	**serve**	**3,032**	**make**	**2,293**	**take**	**828**
to	15,701	over	2,720	**put**	**1,747**	**get**	**479**
in	13,425	**remove**	**2,666**	**boil**	**1,629**	**break**	**193**
of	11,236	you	2,379	**let**	**1,568**	**pull**	**164**
for	9,582	them	2,352	**turn**	**918**	**drop**	**149**
with	8,425	**cut**	**2,337**	**bring**	**889**	**give**	**132**
add	**8,353**	**make**	**2,293**	**take**	**828**	**go**	**122**
your	7,375	**stir**	**2,228**	**allow**	**756**	**hold**	**119**

TABLE IV
TOP 10 MOST FREQUENT WORDS IDENTIFIED AS VERBS GIVEN SIMILARITY THRESHOLD.

B. Step preconditions evaluation

For step preconditions evaluation, giving that we are performing a binary classification task, we need to assign labels for all dataset instances. We are going to return 1 for dataset instances that met step preconditions and 0 for the ones that do not. For simplicity and considering that we do not have an annotated dataset, we assign label 1 to all dataset instances and, for each instance, we produce a second one but this time processing steps backwards, and assign label 0, assuming that if a task is performed in reverse order it does not respect steps preconditions. This operation doubles our dataset size, going from 5,519 to 11,038 total instances, with a balanced dataset with same number of positive and negative instances.

In order to train a neural network, we divide the dataset in training and test sets, with a total of 8,831 instances for training and 2,207 for testing. The neural network is composed by 2 LSTM layers with 16 parameters each. Before inputting data into neural network, we convert input text to embedding representation.

Given that we have two embedding dictionaries available, we performed two experiments, first using *word2vec*, that we refer as *steps-eval-word2vec*, and second using *sent2vec* dictionary to convert sentences to embeddings, this one called *steps-eval-sent2vec*.

Due to vanishing gradient issues in preliminary experiments, we have to truncate our input length to a maximum of 2,000 values. In short, considering that $\vec{v}_i \in \mathbb{R}^{100}$, we can input a maximum of 20 words for *steps-eval-word2vec* and a maximum of 20 sentences for *steps-eval-sent2vec* experiments for each dataset instance.

During training, we use Stochastic Gradient Descent optimizer with learning rate 0.01, momentum 0.9 and Binary Cross-Entropy loss function. We trained our model for 50 epochs, value empirically attained during experimentation, without mini-batches to avoid vanishing gradient issues.

Table V reports our results, where we observe that *steps-eval-sent2vec* performs better with a $\mathcal{F}$ score of 0.8749. We suppose that sentences embeddings performs better first because we truncate our input information to 2,000 values, second because *sent2vec* uses negative sampling to compensate out-of-vocabulary words [7].

Experiment	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{F}$
steps-eval-word2vec	0.8615	**0.8315**	0.8498
steps-eval-sent2vec	**0.9235**	0.8312	**0.8749**

TABLE V
STEP PRECONDITIONS EVALUATION RESULTS

steps-eval-word2vec performs well with a $\mathcal{F}$ score 0.8498 even with input limited to 20 words for a whole task. Considering we have an average of 13.75 steps for each task, an average of 8.74 words for each step, and that we augmented our dataset by reverting instructions to create not correctly ordered tasks, we believe that *steps-eval-word2vec* is learning relationships between at most the three first (normal instance) and three last (reverted instance) steps because no intermediate information is provided due to truncated input, with a loss of approximately 76% of information for each instance. *steps-eval-sent2vec*, however, receives a compact but full representation of whole task, representing better a whole dataset instance when compared to *steps-eval-word2vec*.

IV. Related Work

Using unstructured raw data to define a problem domain is an active area of research. Framer [5] and StoryFramer [4] are methods that accept text as input and translate them into PDDL. Actions and objects are acquired based on Named Entity Recognition (NER) systems that perform syntactic analysis for each word within sentences and assigns a label to them for further processing: verbs are included in actions set; nouns are included in objects set. Users must manually eliminate redundancies and assign preconditions for actions. By default, these systems consider that each action is performed in some location by someone, requiring that users provide this information to be used as preconditions.

Instead of natural language input, other data formats are used to describe problem domains. Asai et al developed LatPlan [1], which accepts images of states transition as input and, using State Auto-Encoders, generates a latent space representation of states, transitions and actions that can be computed to devise a plan. LatPlan returns or a PDDL model (Action Model Acquisition 1) or a set of images that represents the order of actions that must be performed to achieve a goal (Action Model Acquisition 2).

LatPlan's Action Model Acquisition 1 generates large state transition models to be handled by off-the-shelf planners. To overcome this limitation, *Cube-Space Autoencoder* [2] translates unstructured raw data into PDDL model by jointly computing State Auto-Encoder and Action Auto-Encoder instead of computing them independently, which results in a smaller state transition model. An advantange of converting raw unstructured data to PDDL is that it makes possible using domain-independent heuristics to improve planning performance.

V. Conclusion

Describing planning domains with input data other than formal language description is a challenging task. This work is just a small step towards an architecture fully capable of interpreting natural language input with minimal or no human supervision.

We expected to achieve better results with action identification experiment using embeddings. It is possible that these results are related to embeddings dictionaries trained over different domain texts, *i.e.* texts in general use declarative form, where sentences that express steps use imperative form. Using pre-trained embeddings dictionary over large text corpus like Wikipedia and performing training over text in domain could possibly improve performance.

For future research, we would like to explore solutions like truncated backpropagation through time or Transformer [8] architecture to avoid vanishing gradient in longer sequences, a problem that we solved in this work by truncating our inputs size, allowing this model to converge at the cost of significant data loss when working at word level information. To enrich data information, we would like to use a joint combination of word-level and sentence-level embeddings as input to a neural network, in an attempt to improve learning. Other aspects of interest for further research include learning direct correatation between steps, *i.e.* what steps are preconditions for a given step, and organizing tasks into an hierarchical way similar to Hierarchical Task Network (HTN), given that this level of information is already available in this dataset.

References

[1] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary, 2017.

[2] Masataro Asai and Christian Muise. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to strips), 2020.

[3] Mahak Gambhir and Vishal Gupta. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, 47(1):1–66, 2017.

[4] Thomas Hayton, Julie Porteous, Joao Ferreira, Alan Lindsay, and Jonathon Read. Storyframer: From input stories to output planning models. pages –, June 2017. Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). The 27th International Conference on Automated Planning and Scheduling (ICAPS), KEPS 2017 ; Conference date: 18-06-2017 Through 23-06-2017.

[5] Alan Lindsay, Jonathon Read, Joao Ferreira, Thomas Hayton, Julie Porteous, and Peter Gregory. Framer: Planning models from natural language action descriptions. pages –, June 2017. 27th International Conference on Automated Planning and Scheduling, ICAPS 2017 ; Conference date: 18-06-2017 Through 23-06-2017.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[7] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features. In *NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.