Learning Household Task Knowledge from WikiHow Descriptions

Preprint · September 2019

CITATIONS
0

READS
51

3 authors, including:

Yilun Zhou
The University of Sydney
12 PUBLICATIONS **11** CITATIONS

SEE PROFILE

Learning Household Task Knowledge from WikiHow Descriptions

Yilun Zhou
MIT CSAIL
yilun@
mit.edu

Julie A. Shah
MIT CSAIL
julie_a_shah@
csail.mit.edu

Steven Schockaert
Cardiff University
SchockaertS1@
cardiff.ac.uk

Abstract

Commonsense procedural knowledge is important for AI agents and robots that operate in a human environment. While previous attempts at constructing procedural knowledge are mostly rule- and template-based, recent advances in deep learning provide the possibility of acquiring such knowledge directly from natural language sources. As a first step in this direction, we propose a model to learn embeddings for tasks, as well as the individual steps that need to be taken to solve them, based on WikiHow[1] articles. We learn these embeddings such that they are predictive of both step relevance and step ordering. We also experiment with the use of integer programming for inferring consistent global step orderings from noisy pairwise predictions.

1 Introduction

For AI agents to serve as competent (digital or physical) assistants in everyday environments, they need an understanding of the common tasks that people perform. In contrast to factual knowledge, which is encoded to some extent in knowledge graphs such as Freebase (Bollacker et al., 2008), there are currently no resources that capture such knowledge in a comprehensive way.

As a natural solution, in this paper, we consider the problem of learning procedural knowledge from text descriptions, focusing on household tasks as a case study. There are two reasons for this particular focus. First, household tasks require a rich amount of commonsense knowledge, which makes them challenging to deal with for AI agents. Second, learning such knowledge has important applications in the context of household robots and smart home technologies, among others.

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

The biggest challenge associated with household tasks is the lack of explicit structured information. While specialized datasets for some aspects of household tasks are available (e.g. cooking recipes (Yagcioglu et al., 2018), in-home navigation commands (Matuszek et al., 2013), human action trajectories for chores (Koppula and Saxena, 2013)), general information only exists in natural language format as descriptions intended for human consumption. With recent advances in deep learning and text mining, it is natural to wonder whether, and to what extent, we can acquire knowledge about household tasks from existing textual sources. To start answering this question, in this paper we tap into WikiHow, one of the largest online databases of procedural knowledge. Our aim is to jointly learn two types of knowledge: (i) whether a certain step pertains to a certain task and (ii) how to order two (potentially non-sequential) steps for a given task. We evaluate our learned model both in terms of the performance achieved on these two tasks and by analyzing the resulting embeddings.

2 Related Work

Knowledge Representation A large number of knowledge graphs have already been constructed, capturing a wide variety of human knowledge. These graphs, such as Freebase (Bollacker et al., 2008) and ConceptNet (Liu and Singh, 2004), all share the common structure of using nodes to represent concepts and using edges to represent relations. Among many applications, Williams et al. (2017) showed that ConceptNet can enable better story understanding by capturing some aspects of commonsense knowledge.

However, there is little procedural knowledge in ConceptNet. Instead, planning approaches have traditionally been used to model such knowl-

edge. In classical planning languages, such as PDDL (McDermott et al., 1998), the environment is described with a set of predicates and actions are defined in terms of pre-conditions and post-conditions. This turns planning into a search problem. While efficient and provably optimal in small domains, it is hard to model the full spectrum of real world environments with such exact definitions. By contrast, in our work we take a complementary approach, acquiring implicit knowledge from large amounts of data.

Embedding learning Vector space embeddings are commonly used to represent the semantics of linguistic constructs such as words and sentences as vectors in a high-dimensional space. At word level, embedding models such as word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) are trained based on linguistic context, i.e. the representation of a word depends on the words surrounding mentions of that word in some text corpus. At sentence level, embeddings can be trained from context or learned for one or several downstream applications (Radford et al., 2018; Devlin et al., 2018).

Embeddings for various other kinds of data have also been studied. For example, Chung and Glass (2018) learn vector representations from audio data. Moreover, a large number of methods for embedding graph and network structures have been proposed (Goyal and Ferrara, 2018).

Script knowledge In our work, we learn embeddings for natural language instructions targeted to facilitate commonsense knowledge acquisition. One of the ways in which such embeddings can be used is to rank step instructions for a specific task. Specifically, given the name of the task and two steps, all expressed in natural language, we want our model to predict which step should be done first, using only the embeddings of the task name and steps as input. This problem falls into the general category of learning script knowledge, for which several models have already been proposed. For example, Chambers and Jurafsky (2009) proposed one of the first models to learn script knowledge based on estimated mutual information between events. Modi and Titov (2014) learned embeddings for events such that a linear ranking function operating on embedding space can be used to infer event orders. Pichotta and Mooney (2016a) learned orders from parsed event representation with a LSTM model. Pichotta and Mooney (2016b) used a sentence-level LSTM model that does not require explicit event parsing and extraction.

Knowledge Acquisition from WikiHow There have been many works on collecting knowledge from the web. With specific focus on WikiHow, Chu et al. (2017) used information retrieval and embedding-based clustering to distill a knowledge base of task execution. By using the OpenIE system (Angeli et al., 2015), they inferred relations between tasks and steps so that the distilled knowledge base recognizes that the task in a WikiHow article A is equivalent to a step B in another WikiHow article C, and the user, when reading article C can look up detailed instructions for step B by reading the automatically linked article A. In this way, a hierarchical structure among articles can be extracted.

Park and Motahari Nezhad (2018) used a neural network model to learn specific relations between the steps of each task. Specifically, three relations `is_method_of`, `is_subtask_of`, and `is_alternative_of` are learned using a hierarchical attention neural network that achieved superior performance than standard approaches using an information extraction pipeline.

3 Method

3.1 Dataset Description

We collected a corpus consisting of all WikiHow articles under the category of "Home and Garden[2]", which we believe is most relevant for our purpose of understanding household procedural knowledge. Each WikiHow article describes a particular task, which is composed of a number of steps. Some example tasks are shown in Table 1. Each step is represented by a *gist* and an *explanation*. The gist is a brief and concise summary of the step, such as "purchase packing supplies". The corresponding explanation gives additional contexts and details to the gist. For the previous example, the explanation is as follows:

> "Furniture should generally not be placed in a truck without wrapping it in some sort of protective material. After you've completed your inventory, con-

[2] https://www.wikihow.com/Category:Home-and-Garden

Remove Staples with Your Bare Hands
Buy a Shipping Container
Pick Up Broken Glass Splinters
Clean Fireplace Glass
Clean an Espresso Machine

Table 1: A random sample of task titles

Shake your clothes
Move the bowl
Dig a hole about 2 ft deep
Take out the trash
Steam clean older carpets

Table 2: A random sample of task steps

sider what you'll need to move each piece of furniture..."

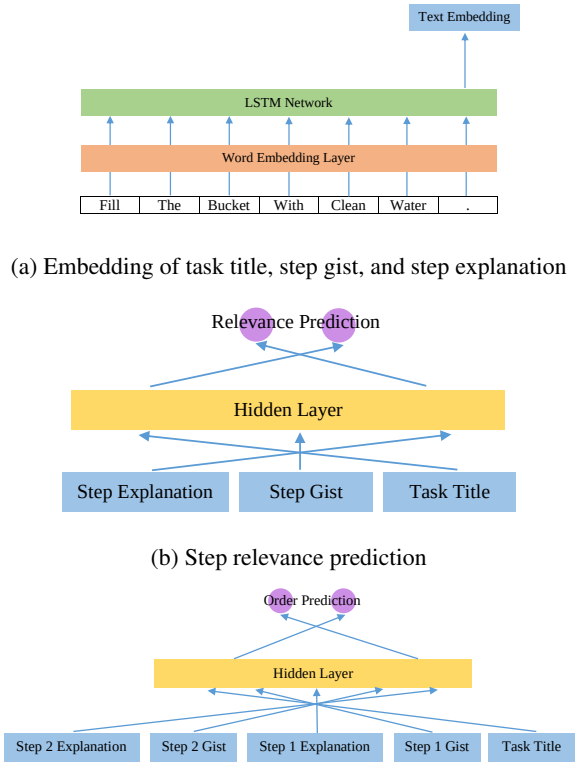
Some additional examples of step gists as shown in Table 2.

An additional particularity with WikiHow is that there are two types of article structures. About 30% of all articles have flat structures, which means that an article has a list of individual steps. The remaining ones have 2-level hierarchical structures, which means that an article has several titled subsections, and each subsection has a list of individual steps. For example, an article on "clean kitchen" may include subsections on "organize kitchen shelves", "clean countertop", and "remove oil stain on floor". We found that subsection titles are semantically and syntactically very similar to article titles, so we simply consider each subsection as a separate task. With this preprocessing, the dataset contains 12,431 articles with a total of 162,771 individual steps.[3]

3.2 Model Architecture

A task title t is a list of l natural language words $(w_1, ..., w_l)$. For each word, we use the pre-trained GloVe embedding (Pennington et al., 2014) to look up its vector representation. This embedding is fixed during training. Words which are not in the GloVe vocabulary are represented using a special <unk> token, the embedding for which is learned. We use $\vec{v}_i$ to denote the embedding corresponding to the word w_i. Hence the task title t is represented as $t = (\vec{v}_1, ..., \vec{v}_l)$. The representations for step gists and step explanations are analogous.

[3]The dataset and the model implementation can be downloaded at https://github.com/YilunZhou/wikihow-embedding/



(a) Embedding of task title, step gist, and step explanation

(b) Step relevance prediction

(c) Step order prediction

Figure 1: Model architecture

Figure 1 shows the general workflow of our model. First, three LSTM networks are used to encode the task titles, step gists and step explanations. The input to each of these LSTMs is a list of word vectors, as explained above, and the outputs are the corresponding embeddings, which are taken as the last hidden state of the LSTM encoder. The initial hidden units and memory units of the LSTMs are initialized to 0 (and not updated during training).

3.3 Step Relevance Prediction

Predicting the relevance of a step to a task is a binary classification problem. We first concatenate the embeddings for the task title, step gist, and step explanation together to form the input vector. Then this vector is passed through a hidden layer and an output layer to get the probability that the step is relevant to the task. Negative log-likelihood (NLL) loss is used during training.

3.4 Step Ranking Prediction

Given the task name and two steps from the WikiHow description of that task, we use a similar fully connected network to predict whether a step should happen before another step from the concatenation of embeddings of the task and the two steps, also with NLL loss.

3.5 Joint Prediction of Step Ordering

Given an unordered set of steps, for a given task, the aforementioned neural network can be applied to make a prediction for pairwise step orderings. However, since these predictions are made independently, they may be conflicting with each other (e.g. A is predicted before B, B is predicted before C, and C is predicted before A). Furthermore, we recognize that sometimes two steps can be done in parallel, and a penalty should not be incurred for incorrectly predicting the ordering in which these two steps happen to be ordered in WikiHow. Thus, to be fully flexible with the possibility of "ambiguous" ordering, we employ an integer programming (IP) formulation.

For each pair of steps (i, j), with $i \neq j$, we introduce two binary variables x_{ij} and x_{ji}. The meaning of $x_{ij} = 1$ is that step i has to occur (strictly) before step j, while $x_{ij} = 0$ means that either step i has to occur (strictly) after step j (if $x_{ji} = 1$), or that the ordering does not matter (if $x_{ji} = 0$). Then we set up the following IP problem:

$$\begin{aligned}
&\text{maximize} \sum_{i,j} w_{ij} x_{ij}, \\
&\text{subject to } x_{ij} \in \{0,1\} \quad \forall i, j, \\
&\qquad x_{ij} + x_{ji} \leq 1 \quad \forall i, j, \\
&\qquad x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \forall i, j, k, \\
&\qquad \sum_{i,j} x_{ij} \geq D.
\end{aligned}$$

In the objective function, we choose $w_{ij} = \log \Pr(i \text{ before } j)$, where this log probability is predicted by the neural network. The first constraint enforces the binary nature of x_{ij}. The second constraint requires that x_{ij} and x_{ji} cannot be both 1, as that would mean that step i is both strictly before and after step j, which is not possible. The third constraint enforces transitivity: if step i is strictly before step j, and step j is also strictly before step k, then we must have that step i is strictly before step k. At this stage, it is easy to see that since $w_{ij} \leq 0$ due to w_{ij} being a log probability, the optimal solution is achieved by choosing $x_{ij} = 0$ for each i and j. Indeed, no penalty is incurred if all pairwise relations are predicted to be ambiguous. For this reason, we have the final constraint which imposes that at least D pairs should be ordered. Note that for a task with T steps, D can be at most $T(T-1)/2$ (i.e. half of all total pairs). Otherwise the second constraint would be unsatisfiable.

3.6 Learning and Inference

We used PyTorch (Paszke et al., 2017) to implement the feed-forward and back-propagation of training, with Adam (Kingma and Ba, 2014) as the optimizer. To solve the integer programming problem, we used CVXPY (Diamond and Boyd, 2016).

4 Experiments

4.1 Data Preparation

We used a 80%/10%/10% split of training, validation, and test data. All reported statistics are from the test set, which is held out during training. Table 3 summarizes the results.

For step relevance prediction tasks, we collected each positive example by sampling a task title and a random step associated with the task. For negative examples, we sampled task titles and steps independently and made sure that the step does not belong to the task. The number of positive and negative examples are balanced.

For step ordering, for each example we sample a task and two steps. Then we randomly denote one of them as step 1 and the other as step 2, and set up the label accordingly.

4.2 Training Details

We used 500-dimensional embeddings throughout, but we found that the learning performance is not sensitive to the embedding dimension, as long as it is over 100. We zero-initialized the hidden and memory cells of the LSTM encoders. The learning rate for the Adam optimizer was set to 0.001.

4.3 Learning Performance

Our model performance is summarized in the first row. In addition, we also tried directly using a bag of word representation as the representation for step explanation, while still keeping the LSTM

	Step relevance	Step ranking
LSTM step explanation	0.911	0.752
bag step explanation	0.902	0.664
no step explanation	0.844	0.657

Table 3: Model performance on two prediction problems

Figure 2: Training (blue) and validation (orange) loss and accuracy for two tasks

encoder for step gist (second row). Specifically, the embedding of the step explanation is calculated as the average of all embedding vectors for words in the explanation. We also tried not using step explanation information at all, whose performance is shown in the third row.

We see that for both relevance and ranking predictions, the full model with step explanation encoded by an LSTM model performs the best. However, using a bag of words vector representation of step explanations still performs better than not using the step explanation at all, although the improvement is small in the case of step ranking prediction.

The learning curve in Figure 2 shows that while the model is able to get very high accuracy on the training set, the validation accuracy stabilizes after a few thousand iterations, indicating that the model is overfitting to the training set afterwards. The test performance in Table 3 was calculated on the test set using the model iteration that achieves the highest validation accuracy.

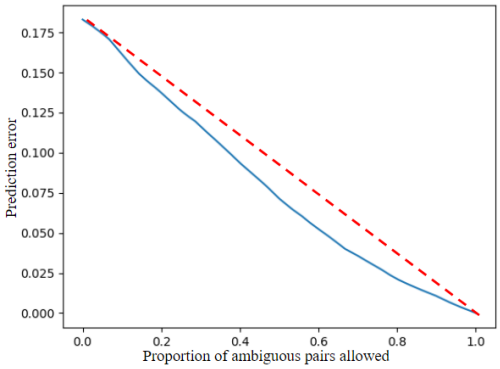


Figure 3: Rank order inference using integer programming

4.4 Integer Programming Inference

In this section, we study if using integer programming inference can provide better ordering performance if we allow the ordering among some pairs to be undecided (i.e. if we set D to be strictly less than half the total number of pairs). Figure 3 presents the result.

The horizontal axis shows the proportion of ambiguous pairs allowed, and the vertical axis shows the proportion of ordering errors. For comparison, we would achieve the red dashed line if we randomly mark pairs as ambiguous, and thus not penalized. We can see that the integer programming inference method is indeed better at identifying ambiguous pairs that, when marked as such, would lead to better performance. However, the improvement is not very substantial, maybe because at training time, the neural network tries to satisfy ambiguous pairs in a way that is more or less arbitrarily defined by the training data, bringing the overall performance down. Thus, one specific idea for future work would be to allow the neural network to intentionally make ambiguous predictions, for which it would not be penalized. Clearly, however, some form of regularization would need to be in place to prevent the network from making the ambiguous prediction too frequently.

4.5 Embedding Visualization

Figure 4 visualizes the embedding of 50 randomly selected tasks, using t-SNE (Maaten and Hinton, 2008) to reduce the dimension to 2. We can see that several clusters of semantically related tasks can be identified, which are indicated by ellipses.

Figure 4: Embedding visualization.

5 Conclusion and Future Work

In this paper, we collected a dataset of natural language instructions for a diverse set of household tasks. We learned a joint embedding of task title and step text for two problems, predicting if a step belongs to a task title and ordering two steps given the task title. We showed that the step relevance can be predicted with a high accuracy if we include the step explanation as part of our model. However, step ordering turned out to be a more challenging problem. We believe that this is at least partly due to noise in the dataset, especially the fact that the ordering of some steps tends to be exchangeable. We also noticed some issues that are specific to WikiHow. For example, the steps which are mentioned for some tasks are more like tips and suggestions.

In terms of future work, one direction is to ground the learned knowledge into some physical systems, such as an in-home robotic platform. Another direction is to try to recognize steps that do not have well-defined orders (Section 4.4), although this is likely to require some additional supervision signal.

6 Acknowledgments

Steven Schockaert has been supported by ERC Starting Grant 637277.

References

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 344–354.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM.

Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised learning of narrative schemas and their participants. 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: Volume 2-Volume 2*, pages 602–610. Association for Computational Linguistics.

Cuong Xuan Chu, Niket Tandon, and Gerhard Weikum. 2017. Distilling task knowledge from how-to communities. In *Proceedings of the 26th International Conference on World Wide Web*, pages 805–814. International World Wide Web Conferences Steering Committee.

Yu-An Chung and James Glass. 2018. Speech2vec: A sequence-to-sequence framework for learning word embeddings from speech. *arXiv preprint arXiv:1803.08976*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Steven Diamond and Stephen Boyd. 2016. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913.

Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Hema Koppula and Ashutosh Saxena. 2013. Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation. In *International conference on machine learning*, pages 792–800.

Hugo Liu and Push Singh. 2004. Conceptneta practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. 2013. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. Pddl-the planning domain definition language.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3111–3119.

Ashutosh Modi and Ivan Titov. 2014. Inducing neural models of script knowledge. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 49–57.

Hogun Park and Hamid Reza Motahari Nezhad. 2018. Learning procedures from text: Codifying how-to procedures in deep neural networks. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 351–358. International World Wide Web Conferences Steering Committee.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Karl Pichotta and Raymond J Mooney. 2016a. Learning statistical scripts with lstm recurrent neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Karl Pichotta and Raymond J Mooney. 2016b. Using sentence-level lstm language models for script inference. *arXiv preprint arXiv:1604.02993*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Bryan Williams, Henry Lieberman, and Patrick H Winston. 2017. Understanding stories with large-scale common sense. In *COMMONSENSE*.

Semih Yagcioglu, Aykut Erdem, Erkut Erdem, and Nazli Ikizler-Cinbis. 2018. Recipeqa: A challenge dataset for multimodal comprehension of cooking recipes. *arXiv preprint arXiv:1809.00812*.