ATLANTIS PRESS

Advances in Intelligent Systems Research, volume 153

International Conference on Applied Mathematics, Modeling and Simulation (AMMS 2017)

Learning to Follow Directions with Untagged Data

Zhidan Yang[1,*] and Zhiting Yang[2]

[1]TSL School of Business and Information Technology, Quanzhou Normal University, Quanzhou 362000, Fujian, China

[2]Yuanyuan Zhu Haoyouxing Information Technology LTD, Quanzhou 362000, Fujian, China

[*]Corresponding author

Abstract—**We describe a program learns to follow directions using a corpus, without human preprocessing. The program only build a semantic lexicon instead of semantic grammar to learn from an untagged corpus. Without grammar, the program uses a language-independent parser to find the boundaries between steps, then parse the steps. The rest of paper explains semantic interpreter and genetic algorithm.**

Keywords-component; untagged;lexicon; interpreter;program; semantic; directions; corpus

I. Introduction

In the field of natural language understanding, the problem of following instructions or directions is of great interest. Its practical value is easy to see, and it raises the important theoretical issue of connecting language to perception and action.

Experience suggests that the best approach is not to build a grammar by hand, but to learn the grammar from a corpus of directions. One possibility is to tag each set of directions with the desired semantic representation, and then use an algorithm like WASP (YukWah Wong et al., 2006) [1] or KRISP (Rohit J. Kate et al., 2007) [2] to learn a grammar that maps directions to the their representations. On the other hand, it is clear that humans learn to understand language without using tagged data. A human being learning her native language must rely on natural language utterances in context -- there is no other evidence available. Imagine a corpus in which each example has two parts: a set of directions, and the route that they describe. The problem of learning to follow directions from such a corpus is roughly similar to the problem faced by humans learning their native language.

Chen and Mooney's program learns to follow directions using a corpus collected by Kuipers. They first divided each set of directions into sentences, and paired each sentence with a sequence of actions, throwing away 300 sentences that did not describe any action. By comparing each sentence to the actions it describes, their program assigns a semantic representation to each sentence. The program uses Rohit Kate's KRISP to learn a semantic grammar from these representations.

Our program learns to follow directions using a corpus similar to Kuipers's, but without human preprocessing. Each example consists of the text typed by an experimental subject and the path that it describes. The program does not build a semantic grammar. It builds only a semantic lexicon.

II. Outline of a Solution

If we want to learn from an untagged corpus, we should reconsider the decision to start by tagging each sentence with a semantic representation. Clarke et. al. [3] have described an approach that does not require such a step. They assume that we start with a parser and semantic interpreter. The parser takes as input the text and the grammar; it creates a semantic representation. The semantic interpreter takes as input the semantic representation, and a starting point on the map. Its output is another point on the map. If the program is successful, this point will be the desired destination -- and we can use the corpus to check. Clarke points out that in principle, this is all the information we need to learn a grammar. Given a candidate grammar, we can use it to assign a semantic representation to each set of directions. Then we run the semantic interpreter, using the given representations as input. We can measure the success of the grammar by counting the number of times the semantic interpreter finds the correct goal. And if we can measure the success of the grammar, we can search through the space of possible grammars until we find a good one. Each candidate grammar assigns its own representation to each set of directions. We are not searching for a grammar that assigns correct representations -- we are searching for a grammar that can follow directions. Clarke calls this approach “learning from the world's response”.

This direct approach is attractive, but it raises the question: how do we search the space of possible grammars? Many popular search algorithms cannot be handle this problem, because they require a subroutine that takes an imperfect solution and finds a way to improve it. To learn a hidden Markov model, we start with a random assignment of weights and repeatedly change the weights in a way that improves performance. The algorithm stops when it reaches a local maximum. Inductive logic programming algorithms also work in this way -- they nearly always rely on a subroutine that takes a program and modifies so that it agrees better with the data. In principle it might be possible to apply this method to our problem. We might find an algorithm that takes a gramma and modifies it so that the rate of success in direction-following improves. However, this sounds unlikely.

Not every search algorithm relies on a subroutine that improves a given solution. Hill-climbing is an obvious example. It uses a subroutine that takes a candidate solution and finds its neighbors in the search space -- other solutions that are only slightly different. It does not require that these neighbors are improvements on the original solution. It searches through them, testing the quality of each solution and choosing one (if any) that improves on the original. Unfortunately, simple hill-climbing is not enough to solve difficult search problems.

Clarke solved the problem with a new learning algorithm of his own invention. Our solution depends on two ideas. First, one can learn to follow directions without learning a grammar. It is enough to learn a semantic lexicon: a mapping from words to concepts. Second, one can learn a semantic lexicon by using a genetic algorithm. Genetic algorithms, like hill-climbing, do not require a subroutine that takes a solution and improves it. Their fundamental operation is to combine two existing solutions into a new solution. On some occasions, the new solution will combine the best parts of the existing solutions, so that it improves on its parents. This is what allows the algorithm to make progress.

Here is an outline of the learning program. The back end consists of the parser and the semantic interpreter. Given a semantic lexicon, the parser creates a semantic representation for each set of directions in the corpus. The semantic interpreter uses this representation in an attempt to follow the directions. By counting the number of times it succeeds, we can measure the quality of the lexicon.

The genetic algorithm begins by creating a large number of lexicons at random, and measuring the quality of each one. It repeatedly chooses two lexicons of high quality and combines them to form a new lexicon. This continues until the average quality of the lexicons in the population stops improving. At this point we take the best lexicon found so far, and improve it further with a simple hill-climbing algorithm. This is common practice --genetic algorithms are not good at fine- tuning.

III. Collecting a Corpus of Directions

Our work uses a corpus of directions similar to the one Kuipers collected. In the first round of experiments we used a small virtual building. We later replaced it with a larger building, containing about 90 rooms. Our virtual building is simpler than the one that Kuipers used -- he added extra landmarks, such as paintings on the walls, and decorated the halls in different colors. Kuipers's program also requires the agent to move from place to place in a series of discrete steps. We have removed this apparently unmotivated requirement, allowing the direction-follower to move continuously. This makes our direction- following task more like direction-following in real life.

Our subjects sit in front of a terminal that displays one of the hallways in this building. The screen also displays two buttons labeled “Show Path” and “Done”. When the subject clicks on “Show Path”, the display changes. The point of view moves down the hall, around a corner, and ultimately into one of the offices. The subject can watch this animation as many times as she likes, until she feels confident that she knows the way. She then types her directions and clicks on “Done”. The directions are sent to the experimenter, who sits at another computer in an adjoining room. The experimenter's computer displays the subject's directions and the starting point of the path. The experimenter has no way of knowing where the destination is, except by reading the directions. The experimenter tries to follow the directions, and then clicks a button marked “Check the Answer”. The result, success or failure, is sent back to the subject, and the experiment continues until the experimenter has succeeded ten times. The experiment is designed to offer the subject some motive for writing good directions. The experimenter is waiting to follow her directions, and she is payed only when the experimenter is able to understand her and reach the desired destination.

We have collected 1324 sets of directions from 133 subjects. One can divide each path into a series of decisions. Each time you turn into a side hall, you have made a decision. Each time you enter an office, you have made a decision. The paths used in our experiment contain from 1 to 3 decisions, with an average of 2 decisions.

IV. The Lexicons

A lexicon is a mapping from words to meanings. The possible meanings are atomic symbols, and there are only a few of them. First come the directions, represented by the constants leftp, rightp and forwardp. Then come the ordinals: firstf, secondf, thirdf and lastf. The predicate doorp means that the destination is a doorway, not a side hall. Finally, some words and punctuation signs are interpreted as separators. They mark the boundaries between steps. In English, the word “and” and the period and comma are separators. The program searches for words that represent these concepts. This is a top-down approach to learning the lexicon. The program does not start with a word and search for its meaning. It starts with a meaning, and searches for words that represent that meaning.

V. The Parser

If the program does not learn a grammar, how can it parse? The rather surprising answer is that it uses a parser which is supposed to be language-independent. The parser's first assumption is that the directions consist of a series of steps, and these steps appear in the same order that we are to execute them. This is not logically necessary -- one could imagine a language in which people always give directions backwards. But that seems unlikely. So the parser begins by assuming a simple language universal. This assumption is both plausible and useful.

The parser's first task is to find the boundaries between steps. In some cases, the separators in the lexicon will mark these boundaries. Separators alone will not solve the problem, for two reasons. In some cases the separators are just not there. More important, the parser must function to some extent in the early stages of the learning process -- when most of the lexicons do not contain accurate information about separators. In the absence of reliable separators, we can use semantic cues to find the boundaries between steps. It is not possible for the same object to be on the left, and also on the right. Therefore two words that mean Left and Right should not appear in the same step. In the same way, an object cannot be both the first and second member of a set -- so two words that mean First and Second should not appear in the same step. Exceptions exist (“Go past the doors on the left and right”), but they are rare in our corpus. One can find the step boundaries with fair accuracy using a simple greedy algorithm: keep adding words to a step until you find either a separator, or a word that is semantically inconsistent with the words that are already in the step.

Next comes the problem of parsing the steps. We are of course not interested in building parse trees for their own sake. We are trying to build a semantic representation by combining the meanings of keywords, and we want to use syntactic clues to find the correct combination. Compare “Take the second hall on the right” and “Turn right into the second hall”. In the first example the word “right” is in the scope of the ordinal “second”. To find the referent of “second hall on the right”, one must take the intersection of the set of halls and the set of objects on the right, then choose the second member of this set. To find the referent of “second hall”, one must find the second member of the set of halls; this hall should also be on the right. The location of the word “right” tells us whether it is in scope of “second” or not.

This example shows that sometimes we need syntactic cues to combine the meanings of keywords correctly. The next question is: “How often do we need these cues?”, and the answer is “Very seldom”. The syntactic clue used in the last example is not even reliable. The sentence “Take a right into the first hall” is often synonymous with “Take the first hall on the right”. Our parser ignores the syntax of the steps, and combines the meanings of keywords with the following simple rule. Suppose the keywords describe the ordinal “n-th”, a direction D and a type T (either door or hall). The representation is n-th DT. That is: we assume the direction is in the scope of the ordinal.

So the parser relies on a simple syntactic assumption which is claimed to be true for all human languages, combined with semantic parsing techniques that are also supposed to be universal. The claim that an object cannot be on my left and my right is correct no matter what language we are speaking. The phrase “language universal” suggests a profound discovery about human language. The language universals used here are not exciting discoveries, but they are useful for the task at hand.

VI. THE SEMANTIC INTERPRETER

This part of the program is very simple. It starts by filling in default values for certain features. If no ordinal is given, it uses the ordinal “firstf”. So “Take a right” is understood as “Take the first right”. If the symbol “doorp” does not appear, the program assumes that the type is “hallp” -- that is, the destination is a side hall. So “Take a right” means a hall on the right, not a doorway. This is not always true. When the directions reach the last step, some speakers consider it obvious that we are about to enter a doorway, and they say “Take the second right” meaning “second door on the right”. But such examples are too rare to have much effect on performance.

Given a type T, a direction D, and an ordinal N-th, the semantic interpreter takes the intersection of the set of currently visible objects of type T and the set of currently visible objects whose direction is D. From this set it chooses the N-th element. It moves the point of view to a point inside the doorway, or a point a short way down the hall. Then it proceeds to the next step. If it reaches a point inside an office, it assumes that this is the desired destination, and ignores the rest of the input.

VII. THE SEARCH ALGORITHM

The genetic algorithm represents a lexicon as a vector of words. Four positions in the vector are reserved for each of the possible meanings, but some of these positions can be left empty. So the vector contains up to four words that mean “left”, up to four words that mean “right”, and so on. The words are chosen from a list of the 87 words that occur most frequently in our corpus. Rare words cannot be very useful because in most of the examples, they never appear at all. So in building the initial population, the program fills each slot as follows. With probability 0.2, the slot is left empty. If the slot is filled, the probability of choosing a word is equal to its frequency in the corpus.

The genetic algorithm itself is quite routine. Parents are chosen by tournament selection, with a tournament size of 4. The population size is 12,000 -- a bit on the high side. The measure of fitness is the number of decisions that are made correctly. So if a path contains three decisions and a certain lexicon gets only the first decision right, it still gets one point. The algorithm typically makes progress for 50 to 60 generations before it plateaus. The result is a pretty good, but not excellent, solution.

The next phase of learning is hill-climbing. This phase is effective because the genetic algorithm is able to reach a part of the search space that is well-behaved -- a place where simple hill-climbing can get results. Each step of hill climbing starts with a cleanup. The program removes any word-meaning pair that fails to improve performance by at least 0.75 percent. Next it addresses a particular weakness of the genetic algorithm. It tends to confuse the ordinals -- thinking that “third” means “last”, for example. The hill-climbing algorithm checks each pair (W,O), where W is a word and O is an ordinal. If it is possible to improve performance by replacing ordinal O with another ordinal, the program does so.

When the cleanup phase is complete, the hill-climbing algorithm considers every lexicon formed by adding a single pair to the current lexicon, choosing the one with the best performance. As usual, it repeats the cleanup and addition of a pair until no more improvement results.

Given a lexicon and a starting point, the semantic interpreter returns a path. Comparing this path with the correct path in the data, one can mark each decision on the path as correct or incorrect. The performance of the path is the number of correct decisions. If the first decision on the path is wrong, performance is zero. Otherwise it is one plus the performance of the rest of the path. Perfect performance is equal to the number of decisions on the correct path.

A ten-fold cross validation yields an average performance of 73 percent of perfect performance. The program returned the same lexicon eight times. It was as follows.

separators: comma,period,and,enter,go,then

leftp: left

rightp: right

doorp: door,doorway,office,room

firstf: first

secondf: 2nd,second

thirdf: third

lastf: last

Notice that the verbs "go" and "enter" appear as separators. Most of the data consists of imperative sentences, and an English imperative starts with the main verb. If "go" and "enter" are common main verbs, they will often appear at the beginning of step, and the system can correctly count them as separators.

The other two lexicons created by the system are as follows. One adds the word "3rd" to the lexicon above, while the other omits the word "then". All definitions in these three lexicons are correct.

ACKNOWLEDGEMENTS

The authors would like to appreciate and memorize Associate Professor Andrew Haas from SUNY Albany, who details the theory in this paper. Since he passed away in 2013, his contact and other information are omitted to avoid unreponsing to requests. The authors are mainly responsible for engineering verification of the theory, therefore most parts of paper are kept as original draft from Andrew. Any questions are welcome, especially those understand the thoery. However, authors may not understand some of the theory part.

REFERENCES

[1] YukWah Wong and Raymond J. Mooney, 2006, Learning for Semantic Parsing with Statistical Machine Translation, In Proc. Of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-2006). pp. 439- 446

[2] Rohit J. Kate and Raymond J. Mooney, 2007, Learning Language Semantics from Ambiguous Supervision, In Proc. Of the 22nd Conference on Artificial Intelligence (AAAI-07). pp. 895-900

[3] James Clarke, 2010, Driving Semantic Parsing from the World's Response, In Proc. Of the Fourteenth Conference on Computational Natural Language Learning.