2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology

An Architecture for Bootstrapping Lexical Semantics and Grammatical Structures

Tom Armstrong
Wheaton College
Norton, Massachusetts 02766
tarmstro@wheatoncollege.edu

Tim Oates
University of Maryland, Baltimore County
Baltimore, Maryland 21250
oates@cs.umbc.edu

***Abstract*—Computational approaches to language acquisition typically reduce the problem to one of either learning syntax, or learning a lexicon – isolated tasks. These problem simplifications have led to disconnected solutions and algorithms with no expectation of grounded inputs. We propose a language learning framework for bootstrapping the acquisition of a grammar and learning lexical semantics. We describe the framework and provide an instantiation using context-free grammars and a λ-calculus semantic representation.**

Keywords: lexical learning, natural language learning, bootstrap learning

I. Introduction

When human language learners begin acquiring their native tongues, they do not first spend time exclusively learning the words in the language and then move to learning the grammatical structures. Rather, over time, they acquire components of a grammar and collections of words as they grow their language repertoire. Language learning algorithms typically assume complete knowledge of either lexical information or grammatical structures. These algorithms do not address the origin of the additional information. In this paper, we propose a high-level architecture for bootstrapping syntactic and semantic learning to acquire a context-free grammar and lexicon. This bootstrap leverages the salient information available in the learner's environment combined with existing knowledge. Then, it learns additional lexical semantics for novel words that enables the system to learn grammatical structures. We introduce the formalism for representing lexical semantics, point to supporting work that can serve as components in the bootstrap, and outline the bootstrap architecture.

II. Background

Boole and Montague pioneered a formal mathematical treatment of natural language. In Montague's conception, he used the λ-calculus as the basis for representing word denotations, or what we will refer to as lexical semantics [1], [2]. In this representation, word meanings are functions represented as λ-calculus expressions. Each word's λ-calculus expression has an accompanying *semantic type* (see Table I). Semantic types are ordered pairs that correspond to the types of the inputs to the function and outputs of the function. The λ-calculus has a simple system of reductions to compute the output of a sequence of expressions: α- and η-conversions; and β-reductions. Sentential denotations are arrived at by composing functions and applying these reductions.

Word	λ-Calculus Expression	Semantic Type
Nathaniel	NATHANIEL	`e`
Isabel	ISABEL	`e`
loves	λx.λy.LOVES(y, x)	`<e,<e,t>>`

Table I
LEXICAL TERMS AND λ-CALCULUS REPRESENTATIONS

Given a grammar and lexicon with associated λ-calculus expressions, we can use the structure and semantic types to direct the function composition. For example, consider the sentence *"Nathaniel loves Isabel"* and its bracketing [NATHANIEL [LOVES ISABEL]] from some grammar using the lexicon in Table I. To arrive at the sentence denotation of the sentence, we follow a series of λ-calculus β-reductions in Table II. The result is a first-order logic expression.

III. Related Work

Our bootstrap architecture covers two different learning problems: learning syntax using lexical semantics and learning lexical semantics using syntax. In the *learning in the limit* paradigm, a characteristic sample of positive example strings is not sufficient to guarantee learning target grammars for all languages [3]. Additional information is often required, and that can take the form of, for example, negative example strings [4], [5], or structural information [6]. Additionally, learnability can follow from restrictions on the class of languages from which the target is drawn [7], [8] or on the method for selecting examples [9].

Establishing learnability for broader language classes often requires more information. A number of results on the learnability of context-free languages from positive examples require each string to be paired with its unlabeled derivation tree [10]. Lexical semantic information, combined

978-0-7695-4513-4/11 $26.00 © 2011 IEEE
DOI 10.1109/WI-IAT.2011.188

[NATHANIEL [LOVES ISABEL]]	=	[NATHANIEL [[λx.λy.LOVES(y, x)] ISABEL]]
	$\Rightarrow_\beta$	[NATHANIEL [λy.LOVES(y, ISABEL)]]
	$\Rightarrow_\beta$	LOVES(NATHANIEL, ISABEL)

Table II
λ-CALCULUS DERIVATION OF *"Nathaniel loves Isabel"*

with positive example strings, is sufficient to induce unlabeled derivation trees for the samples [11]. Fully lexicalized grammars (e.g., combinatory categorial grammars [12]), those where the syntactic information is embedded within the semantic descriptions, require only word-composition rules with no additional grammatical rules needed [13], [14]. However, learning these grammars reduces to the same pair of tasks: learning the functional descriptions of each word and learning the grammatical structures for the language. The literature exploring learning lexical semantics from syntactic information is sparse [15], but the λ-calculus expression types are learnable from certain context-free grammars [11].

IV. BOOTSTRAPPING FRAMEWORK

Our proposed bootstrap architecture consists of two learning components: 1) a structure learner; and 2) a meaning learner (see Figure1). The input to the bootstrap is a declarative representation of the learner's environment and a set of strings, or utterances – each describing something about the environment. Outside of the scope of this paper is an initial associative learning component through which the bootstrap is initialized from data in the environment. That is, an approach to solving the symbol grounding problem acquires a mapping between objects or attributes in the environment and the strings.

When the system processes strings containing words already in the lexicon, the grammatical inference proceeds without intervention. The declarative representation and lexicon are used to build derivation trees, then a learning algorithm builds a grammar with the trees [11]. However, when the system encounters a novel word two tasks must complete in our architecture before the word is added to the lexicon. First, we must determine the semantic type of the new word with an algorithm called TYPEINFER. Second, we must determine the λ-calculus expression for the new word with an algorithm called LAMBDALEARNER. This allows the system to: 1) learn semantic representations of novel words online; and 2) incorporate new syntactic knowledge into future novel word discovery.

We aim to assist the structure learner with additional structural information via derivation trees. The second half of the bootstrap's goal is to take strings and augment them with derivation trees. These structural example strings – complete parse trees – are then fed as input into the structure learner, a grammatical inference algorithm (e.g., KRCFG [6]). Given the semantic types of all words in an utterance, we search over all possible derivation trees consistent with the types. We compute the denotations of each potential derivation tree with the λ-calculus expressions for each word. Ultimately, we select derivation trees for sentences with denotations consistent with the environment to use with a grammatical inference algorithm. These paired processes continue as new data are presented.

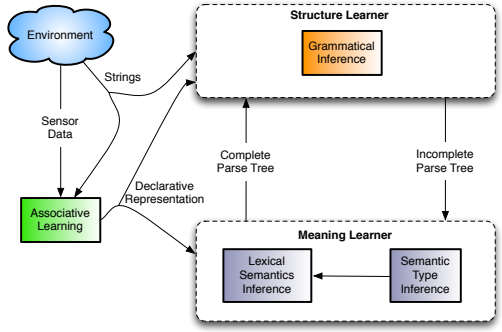

Figure 1. Syntax and Sematics Bootstrap Architecture

A. Meaning Learner – Type Inference

The meaning learner begins with an utterance where all save one word have a known semantic representation. Discovering the type of the word will help in the next stage of the bootstrap, and the more constrained the search, the more efficient the approach will be. Using the known lexical types, the algorithm determines what the possible types are for the novel word. That is, the current lexicon's types constrain the type for the unknown word between the simplest type and the most complex, a type that can consume any combination of types in the lexicon.

The currently known types and incomplete parse tree restrict the type from the simplest to a combination of all of the known types, but no more. For example, consider the utterance *"the cat hates the dog"*. Now suppose that we know the types of the λ-calculus expression for *the*, *cat*, *hates*, but not *dog*. The simplest type in this example is `<e,t>` and the most complex type is a depth-limited type expression using the existing lexical types as primitive types. The simplest type turns out to be the correct type, and the most complex type is a parasitic type that moves from right to left consuming each subsequent word. The output of the type inference algorithm is a non-empty set containing possible types for the novel word that result in a semantically-valid parse tree.

B. Meaning Learner – λ-Calculus Expression Learning

Our learner hypothesizes possible lexical semantics for novel words, and uses the environment and future environments to converge to the correct semantic representations. The input to our λ-calculus learning algorithm is an utterance with an unknown word, the type of the unknown word, the existing lexical semantics, and a declarative representation of the environment.

Learning occurs in two stages. In the first stage, we use the environment and the previously learned type information to constrain the search space of possible λ-calculus expressions. Second, we either know the correct lexical semantics for the word, or we are maintaining multiple hypotheses about its meaning. For example, in an environment with one dog and one cat jumping, a valid utterance is *"the dog jumps"*. However, without knowing the word for CAT(X), we are unable to know if the word *dog* maps to DOG(X) or if it maps to CAT(X). Instead, we must maintain both theories until we encounter an environment that disambiguates cats and dogs.

The algorithm is enumerative in nature with respect to the contents of the environment and how specific a lexical semantic representation we require. That is, the environment representation contains higher-level predicates like CAT(X) and ORANGE(X), but not PIXEL(X)(Y)(Z). Given a set of logical connectives (e.g., ↔, →, ←, ∨, ∧) and quantifiers (e.g., ∀, ∃), the algorithm creates well-formed formulae using elements from the environments. Moreover, these well-formed formulae are only of the type given in the input. There is a one-to-one mapping from types to λ-calculus expression forms. For example, given the type <e,t>, the form of the resulting λ-calculus expression contains a single λ-abstraction because the type represents a single function from *entities* to *truth values*. Each potential expression for the type <e,t> will be of the form λx.[PRED(x)] where PRED(X) is filled with a predicate found to be true in the environment. All of these mappings are known *a priori* and are fixed.

C. Ranking Candidate λ-Calculus Expressions

Finding that the sentence meaning is consistent with the environment does not entail that the candidate expression is correct. To address this issue, we discuss an approach to using a series of heuristics that evaluate the expressions. We negatively weight candidate expressions with undesirable qualities. Paul Grice posits a COÖPERATIVE PRINCIPLE in which conversation participants follow a set of mutually agreed upon guidlines – the Gricean conversational maxims [16]. Grice breaks down the principle into four maxims: Quantity, Quality, Relation, and Manner. Quantity describes a balance between precision and accuracy where the speaker is expected to provide just enough, but not more than enough information to the listener. Quality requires that the speaker provide evidenced information with high fidelity. Relation demands statement relevance, and manner imposes stylistic requirements (e.g., brevity, unambiguous, etc.).

We derive motivation for our heuristic from Grice's conversational maxims and ground weightings in the maxims. **Quantity** – We wish to avoid overly complex formulae. That is to say, we prefer semantic descriptions that avail themselves of an economy of representation. We negatively weight expressions that contain overly verbose, correct descriptions. For each logical connective, we penalize a candidate expression. This heuristic negatively weights the rank by 1 for each instance. **Quality** – Our goal is to learn the correct meanings of words, but no more and no less. To bias the learner to prefer statements that cover a wider set of inputs, universal quantification is preferable to existential quantification. This heuristic negatively weights the rank by 1 for each instance. **Relation** – The composition of each candidate expression is restricted to the environment or scene. This heuristic negatively weights the rank by 5 for each instance. **Manner** – A perennial challenge for all language acquisition algorithm is avoiding and overcoming ambiguity obstacles. This problem presents itself in candidate expressions when we have, for example, symmetric predicates like *near*. This heuristic negatively weights the rank by 2 for each instance.

V. THE LIFE OF A NOVEL WORD

Let us look at a bootstrap in progress. Suppose the environment is filled with a collection of animals that are jumping named FLUFFY, PUMPKIN, and ROVER. The following are true in this environment: 1) CAT(FLUFFY), CAT(PUMPKIN), DOG(ROVER); 2) GRAY(PUMPKIN), JUMPS(PUMPKIN), JUMPS(FLUFFY), JUMPS(ROVER); and 3) LOVES(FLUFFY, PUMPKIN). The lexicon contains entries for the named entities and the denotation of the words *"cat"* and *"jumps"*. Suppose our learner encounters the sentence *"Every cat jumps"*. We are now tasked with discovering the type of *"every"* and its corresponding λ-calculus expression. LAMBDALEARNER returns a ranked list of λ-calculus expressions for *"every"*.

In this example, there are two top-ranked expressions for *every*: 1) λP.λQ.∀x:[P(x) ↔ Q(x)]; and λP.λQ.∀x:[P(x) → Q(x)]. The edit distance between the two expressions is small, but given the state of the environment, it is not possible to know if the relationship between jumping and cats is one of implication or biconditionality. To determine this, the learner must maintain these hypotheses until she encounters an environment that disambiguates between the situations. In this case, if the learner sees an environment where a dog is also jumping (i.e., the environment contains an additional predicate JUMPS(ROVER)), then the learner can succeed. One of the candidate expressions λP.λQ.∀x:[P(x) ↔ Q(x)] is no longer true with respect to the environment, and the candidate is excluded. Whereas, λP.λQ.∀x:[P(x) → Q(x)] is true and maintains the best ranking. At this state,

Ranking	λ-Calculus Expression
1	$\lambda P.\lambda Q.\forall x:[P(x) \leftrightarrow Q(x)]$
1	$\lambda P.\lambda Q.\forall x:[P(x) \rightarrow Q(x)]$
2	$\lambda P.\lambda Q.\exists x:[P(x) \wedge Q(x)]$
3	$\lambda P.\lambda Q.\exists x:[P(x) \vee Q(x)]$
4	$\lambda P.\lambda Q.\forall x:[P(x) \vee Q(x)]$
5	$\lambda P.\lambda Q.\forall x:[P(x) \vee Q(x)]$
5	$\lambda P.\lambda Q.\exists x:[P(x) \vee Q(x)]$
6	$\lambda P.\lambda Q.\exists x:[P(x) \leftrightarrow Q(x)]$
6	$\lambda P.\lambda Q.\exists x:[P(x) \rightarrow Q(x)]$
6	$\lambda P.\lambda Q.\exists x:[P(x) \vee Q(x)]$
6	$\lambda P.\lambda Q.\forall x:[P(x) \leftrightarrow Q(x)]$
6	$\lambda P.\lambda Q.\forall x:[P(x) \rightarrow Q(x)]$
7	$\lambda P.\lambda Q.\exists x:[P(x) \wedge Q(x)]$
8	$\lambda P.\lambda Q.\ \exists x:[P(x) \rightarrow Q(x)]$
8	$\lambda P.\lambda Q.\ \exists x:[P(x) \vee Q(x)]$
9	$\lambda P.\lambda Q.\exists x:[P(x) \rightarrow Q(x)]$
11	$\lambda P.\lambda Q.\exists x:[P(x) \leftrightarrow Q(x)]$
11	$\lambda P.\lambda Q.\exists x:[P(x) \rightarrow Q(x)]$

Table III
LAMBDALEARNER λ-CALCULUS CANDIDATE EXPRESSIONS AND RANKINGS FOR *every* (LOWER RANK IS BETTER)

we continue to process sentences using the top ranked λ-calculus expression learned for the word *every*.

VI. CONCLUSIONS

To avoid negative theoretical results and the acquisition of an entire grammar and lexicon, we propose a bootstrap architecture. This architecture combines the tasks of learning the syntax of a formal language and the semantics of individual words. Our bootstrap includes two algorithmic components: 1) Structure Learner; and 2) Meaning Learner. Our preliminary work demonstrates that the simultaneous, incremental acquisition of syntax and semantics is an area of interest for the grounded acquisition of language.

To begin the bootstrap, we introduced a series of algorithms and motivation for a heuristic to initialize the syntax-semantics bootstrap and connect up inputs and outputs to close the bootstrap loop. Our learner exploits her position embedded in an environment to learn not only type-theoretic information about words, but grounded, functional lexical semantics. We developed an approach for inferring the structure, or unlabeled derivation tree, of a string using the words in sentences to impose lexical semantic constraints. Our inspiration comes from computational linguistics and compositional semantics combined with the motivating type-theoretic results. Future work will proceeds in three directions. First, we will increase the contents of the learner's environment. Next, we will explore the utility of the learned grammatical structures in learning semantic representations. Last, we will compare the system's performance to that of results from first language acquisition.

REFERENCES

[1] D. R. Dowty, R. E. Wall, and S. Peters, *Introduction to Montague Semantics*. Dordrecht: Reidel, 1981.

[2] B. Partee and H. Hendriks, "Montague grammar," in *Handbook of Logic and Language*, J. van Benthem and A. ter Meulen, Eds. Amsterdam: Elsevier Science and The MIT Press, 1996, pp. 5–91.

[3] E. M. Gold, "Language identification in the limit," *Information and Control*, vol. 10, pp. 447–474, 1967.

[4] J. Oncina and T. García, *Inferring Regular Languages in Polynomial Update Time*. World Scientific Publishing, 1992, pp. 49–61.

[5] T. Armstrong and T. Oates, "Learning in the lexical-grammatical interface," in *FLAIRS Conference*. AAAI Press, 2008.

[6] T. Oates, D. Desai, and V. Bhat, "Learning k-reversible context-free grammars from positive structural examples," in *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[7] T. Oates, T. Armstrong, L. B. Bonache, and M. Atamas, "Inferring grammars for mildly context-sensitive languages in polynomial-time," in *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*, 2006.

[8] T. Koshiba, E. Makinen, and Y. Takada, "Inferring pure context-free languages from positive data," *Acta Cybernetica*, vol. 14, no. 3, pp. 469–477, 2000.

[9] F. Denis, C. D'Halluin, and R. Gilleron, "PAC learning with simple examples," in *Symposium on Theoretical Aspects of Computer Science*, 1996, pp. 231–242.

[10] Y. Sakakibara, "Efficient learning of context-free grammars from positive structural examples," *Information and Computation*, vol. 97, pp. 23–60, 1992.

[11] T. Oates, T. Armstrong, J. Harris, and M. Nejman, "On the relationship between lexical semantics and syntax for the inference of context-free grammars," in *Proceedings of the 19th National Conference on Artificial Intelligence*, 2004.

[12] K. Ajdukiewicz, "Die syntaktische konnexitat," *Studia Philosophica*, 1935.

[13] I. Tellier, "Meaning helps learning syntax," *Lecture Notes in Computer Science*, vol. 1433, 1998.

[14] T. Kwiatkowksi, L. S. Zettlemoyer, S. Goldwater, and M. Steedman, "Inducing probabilistic CCG grammars from logical form with higher-order unification," in *EMNLP*, 2010, pp. 1223–1233.

[15] K. Gold, M. Doniec, C. Crick, and B. Scassellati, "Robotic vocabulary building using extension inference and implicit contrast," *Artificial Intelligence*, vol. 173, no. 1, pp. 145 – 166, 2009.

[16] P. Grice, "Logic and conversation," in *Speech Acts*, P. Cole and J. L. Morgan, Eds. New York: Academic Press, 1975, vol. 3, pp. 41–58.