LEARNING FROM NATURAL INSTRUCTIONS

BY

DAN GOLDWASSER

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Dan Roth, Chair, Director of Research
Professor Gerald DeJong
Assistant Professor Julia Hockenmaier
Professor Raymond Mooney, University of Texas at Austin

Abstract

In this work we take a first step towards Learning from Natural Instructions (LNI), a framework for communicating human knowledge to computer systems using natural language. In this framework the process of learning is synonymous with language interpretation, the process in which natural language sentences are converted into a logical representation which can be understood by an automated agent.

While the motivation behind this framework is clear, the practical aspects involved in constructing it are non-trivial: communicating effectively with computer systems has been one of motivating forces behind artificial intelligence research since its inception. The rigid way in which computer systems naturally take instructions, via programming, and the flexible and ambiguous way in which humans naturally provide instructions, via natural language, rendered this task extremely difficult.

At the heart of this work stands the problem of semantic interpretation, viewed through the perspective of LNI. In this work we consider realistic settings in which LNI is applicable, by framing semantic interpretation as a machine learning problem and suggesting training protocols that help facilitate LNI and reduce the level of human effort involved.

Most current works view semantic interpretation as a structured learning problem, in which a structured predictor is trained in a supervised manner using pairs of sentences and their corresponding meaning interpretations expressed as a logical formula. This type of annotation is extremely costly. Moreover, the training process is *domain dependent*, resulting in a task specific interpreter. The LNI settings call for a more flexible process, as the communicated knowledge is not limited to a single task. The supervised settings described above require repeating the training process from scratch when approaching a new domain. Alleviating these problems is a necessary step towards effective LNI.

In this work we suggest an alternative to learning in the supervised settings. Our solution is situated in the LNI settings and exploits a simple observation: the communicated knowledge should result in an observable change of the agent's behavior. We move away from the traditional supervised learning settings by using a feedback signal derived by exploiting this observation: desirable changes in behavior are considered as positive feedback and the vice versa. Consider, for example, teaching an automated agent

the rules to a game by providing it with natural language explanations of the game rules. Successful interpretation of these rules would result with the agent possessing the ability to make correct decisions in a game scenario. By observing the agent's behavior and using it as feedback we can reinforce or penalize the semantic interpretation model leading to the observed behavior.

In contrast to the supervised settings, in which learning depends on annotated data in the form of pairs of sentences and the corresponding logical structure, the learner now has access to sentences and binary labels corresponding to judgments of the resulting behavior. From a machine learning perspective this approach can be thought of as learning a structured predictor using binary feedback. We discuss learning in these settings, and suggest several response driven learning methods.

All response-driven learning algorithms share a common structure: they generate their own training data, by exploring the space of possible interpretations for the given set of inputs (according to its current semantic interpretation model) and updating it according to the feedback obtained by applying the resulting interpretation. Note that unlike the supervised settings, in which the model is updated with respect to a training set, in response-driven learning the model is updated using binary feedback alone. Response-driven learning algorithms differ in the way the model is updated given this unusual type feedback.

When the resulting interpretation is assigned a positive feedback, using it is straightforward – it can simply be used as an annotated structure. In this case response-driven learning is simply an iterative learning algorithm, which uses a supervised learning procedure to update the model. This approach ignores an important supervision signal in cases the predicted interpretation resulted in a negative feedback. We suggest a perceptron-based algorithm that uses both negative and positive feedback to drive learning, by combining both structured perceptron (for sentences resulting in positive feedback), and binary perceptron (for sentences resulting in negative feedback).

We take an additional step by taking a closer look at interpretations assigned a negative feedback. In this case we employ the binary perceptron demotion rule, penalizing uniformly all the semantic mapping decisions in the interpretation assigned a negative feedback. However, some of the decisions taken by the model are more likely to be correct while others are less probable. We suggest to approximate the probability of correctness of structural decisions and penalize the model accordingly.

Taking the correctness approximation approach even further, we suggest an unsupervised variant of our learning system, in which similar correctness estimation methods are used to drive learning even without the external, response based feedback.

In the case of response driven learning, the confidence estimation approach used for penalizing incorrect decision was based on ground truth – the response signal. In the unsupervised version, we rely on the

model's self-predictions to estimate the decisions confidence. We use the confidence score to re-train the model by using it as a filter generating better training data for a self-training protocol.

Finally, we take an initial stab at the problem of domain dependence when learning a semantic interpreter. A domain can be thought of as a fixed set of logical symbols representing entities and concepts in the domain. For example, these symbols in a card game domain would represent concepts such as cards, their properties, and actions involving them. When constructing a semantic interpreter the set of possible meaning interpretations is constrained by these symbols. Thus, when moving to a new domain, using a different set of symbols, a different semantic predictor needs to be learned.

In the traditional supervised learning setting, when multiple domains are learned, learning is done independently for each domain, using different sets of training examples. When thinking about semantic interpretation as the tool needed to accommodate LNI, where several tasks and domains can be considered, this approach seems wasteful.

Following the observation that high level semantic structures remain the same regardless of the domain learned, we suggest an intermediate representation capturing high level semantics and show that it can be learned in one domain and then used to facilitate interpretation in a different domain.

To my parents, my past and future teachers.

Acknowledgments

This work is the product of a long process, both academic and personal. I would like to acknowledge the help and guidance I got, in both aspects.

First, I would like to thank Dan Roth, my advisor, for his supervision, common sense, good advice and kindness. Our almost accidental initial encounter has sparked my interest in this field and led to many discussions resulting in me joining Dan's research group. Working with Dan was an unforgettable experience, his in-depth understanding of artificial-intelligence and the connections between its different disciplines, his commitment to research and his openness to new ideas, have all helped turn the Phd studies into a period I will miss and look back to for inspiration.

The answer most commonly heard when one is asked to describe their graduate studies (when one answers honestly), has typically something to do with the stress and uncertainty of that period. I have a deep appreciation and gratitude for Dan's ability to apply the same patience and understanding to personal matters, helping to alleviate the stresses of this period.

I would like to thank my past advisors, Ofer Strichman and Shai Fine, who helped shape my thinking process.

I had the privilege of having Gerald DeJong, Julia Hockenmaier and Raymond Mooney on my thesis committee. Their comments and guidance helped direct this work and also develop broader research intuitions. Their help is greatly appreciated.

Working in Dan's group was mind-broadening experience. I was fortunate to work with many remarkably smart people, among them: Kai-Wei Chang, Quang Do, Jeff Pasternack, Gourab Kundu, Alla Rozokovska, Mark Sammons, Vinod Vydiswaran, Yuancheng Tu, Vivek Srikumar, Rajhans Samdani, Wei Lu, Prateek Jindal, James Clarke, Jacob Eisenstein, Ivan Titov, Yee Seng Chan, Ming-Wei Chang, Michael Connor, Alex Klementiev, Lev Ratinov, Nick Rizzolo and Kevin Small.

I would like to thank Nick Rizollo and Mark Sammons for their endless patience and willingness to help explore various technical hurdles.

Research in its nature is a collaborative effort, and many of the ideas described in this work are the

result of discussions and collaborations. I collaborated closely with James Clarke, whose ability to connect research ideas to real world concepts was unmatched. I enjoyed our work, discussions and travels. I enjoyed and learned a lot from my discussions with Ming-Wei Chang, his deep understanding of machine learning and his high standards were inspiring. I had many fruitful discussions with Vivek Srikumar, his almost insatiable academic curiosity has helped me understand better many ideas and intuitions.

The cold Urbana winters would have been much colder without the company of several people. Talking with Yoav Sharon has always helped me to keep things in perspective, his sincere kindness and calm nature were a constant source of support. Knowing Lev Ratinov, with his unquenchable thirst for life, was a remarkable experience. Fellow Cog-comper, housemate, travel companion and a friend. I enjoyed the conversations and time I spent with Yonatan Bisk, Rajhans Samdani and Alla Rozokovska. Their company helped turn the work oriented Siebel center into a friendly place. Yonatan also helped me keep the rust off my Hebrew. I was very fortunate to find such a group of unique individuals.

I have a deep appreciation for my friendship with Yuval Link and Shmuel Raz, despite the dramatic changes in our lives, our friendship has remained unchanged.

I would like to thank my family, Miriam, Yehuda, Amit and Dora, for their support and understanding. Words cannot express my gratitude for your endless support and concern.

Finally, Rachel, whose support, patience and sacrifice has made so much difference.

Publication Notes

The work described in this their is built on work described in the following publications:

- *Learning from Natural Instructions.* Dan Goldwasser and Dan Roth. To appear in Machine Learning Journal, Special Issue on Learning Semantics. 2012.
- *Learning from Natural Instructions* Dan Goldwasser and Dan Roth. Proceedings of IJCAI 2011.
- *Confidence Driven Unsupervised Semantic Parsing.* Dan Goldwasser, Roi Reichart, James Clarke and Dan Roth. Proceedings of ACL 2011.
- *Driving Semantic Parsing from World's Response.* James Clarke, Dan Goldwasser, Ming-Wei Chang and Dan Roth. Proceedings of CoNLL 2010.
- *Reading to Learn: Constructing Features from Semantic Abstracts.* Jacob Eisenstein, James Clarke, Dan Goldwasser and Dan Roth. Proceedings of EMNLP 2009.

Other relevant works by the author:

- *Structured Output Learning with Indirect Supervision.* Ming-Wei Chang, Vivek Srikumar, Dan Goldwasser and Dan Roth. Proceedings of ICML 2010.
- *Discriminative learning over constrained latent representations.* Ming-Wei Chang, Dan Goldwasser, Vivek Srikumar and Dan Roth. Proceedings of NAACL 2010.
- *Constraint Driven Transliteration Discovery.* Dan Goldwasser, Ming-Wei Chang, Yuancheng Tu and Dan Roth. . Book chapter in RANLP 2009.
- *Unsupervised Constraint Driven Learning for Transliteration Discovery.* Ming-Wei Chang, Dan Goldwasser, Dan Roth and Yuancheng Tu. Proceedings of NAACL 2009.
- *Transliteration as Constrained Optimization.* Dan Goldwasser and Dan Roth. Proceedings of EMNLP 2008.
- *Active Sample Selection for Named Entity Transliteration.* Dan Goldwasser and Dan Roth. Proceedings of ACL 2008.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI	Artificial Intelligence
EM	Expectation Maximization
ILP	Integer Linear Programming
LBJ	Learning Based Java
LNI	Learning from Natural Instructions
ML	Machine Learning
NL	Natural Language
NLP	Natural Language Processing
NLIDB	Natural Language Interface to Data-Bases
POS	Part Of Speech
RL	Reinforcement Learning
SRL	Semantic Role Labeling
WNSim	WordNet Similarity

Chapter 1

Introduction

Constructing intelligent agents that can communicate effectively with humans and respond correctly to human-level instructions is one of the key Artificial Intelligence (AI) goals, which to a large extent has guided AI research. The difficulty of making progress towards this goal can be traced to the gap between the natural way in which computer systems take instructions, using a hard-coded set of rules programmed in advance, and the way humans provide them, using natural language, a flexible yet ambiguous and ill-defined medium.

A straight forward approach for accommodating human-machine communication is programming, simply put – defining a set of rules in a machine language capturing the intended behavior and knowledge required by a computer system. This approach, bridging the gap between human and computer communication languages, transfers the difficulty completely to the human side– it is the sole responsibility of the human teacher to express his knowledge in machine language, and as a result, has to be familiar with the system's internal workings.

The limitations of the rule-based approach, in which the agent's behavior is hard-coded in advance, gave rise to Machine Learning (ML) driven approaches. In its most conventional form this problem is approached using *supervised* machine learning methods: given a set of labeled examples the learner constructs a decision function generalizing over the observed training data.

The advantage of these methods is clear – it provides a far more flexible interface to human-machine communication. Rather than explicitly programming the automated agent's response for every possible eventuality, the desired behavior of the system is defined by generalizing over human judgments given related scenarios.

While this approach has been tremendously successful for many domains, it has several inherent drawbacks: first, the learner can only be as good as the data it is given, learning therefore depends on annotating considerable amounts of training data, an expensive and time consuming process. More importantly, successful learning often depends on ML expertise required for designing and calibrating the learning environment.

A preferable solution is to use a knowledge communication protocol that relies on *Learning from Natural Instructions* (LNI) rather than traditional example-based learning. This protocol draws motivation from human learning processes, in which the learner is given a "knowledge injection", in the form of a lesson describing a high level concept and a few specific examples to validate the correct understanding of the lesson. Example-based learning, an inductive process by nature, generalizes over the labeled examples. This contrasts directly with LNI, which attempts to learn the correct hypothesis directly.

In order to better clarify the notion of Learning from Natural Instructions, let's consider an example. In this thesis we will focus on card games as the domain of choice for Learning from Natural Instructions, our example is therefore taken from the Freecell Solitare card game. We assume a set of game-specific predicates (such as $\texttt{move}(\texttt{c},\texttt{f}), \texttt{card}(\texttt{c})$, etc.) describing a specific game state (denoted as s).

Example 1. *[Freecell Solitaire Rule]*

- *Natural Language*

  "Any top card can be moved to an empty freecell"

- *Logical Formula*

  $\texttt{move}(\texttt{c},\texttt{f}) \leftarrow \texttt{card}(\texttt{c}) \wedge \texttt{top}(\texttt{c},\texttt{x}) \wedge \texttt{freecell}(\texttt{f}) \wedge \texttt{empty}(\texttt{f})$

- *Linear Equation*

  $\mathbf{w}^T \Phi(\mathbf{s}) > \mathbf{0}$

This example describes a decision function, classifying the legality of game moves in Solitare card game. This decision function can be described in natural language, allowing humans to make such classification, or as a logical rule that can be evaluated by a machine. A different representation of the same logical rule can expressed as a linear equation, classifying the legality of game states using a linear threshold function defined over a feature representation of the game state and a real-valued vector. The third representation is a popular representation used by statistical machine learning algorithms. The difference between LNI and traditional machine learning can be summarized as follows - in LNI we wish to learn the second rule using the first rule as input, and in statistical machine learning the decision rule requires taking a considerably sized set of training examples .

Applicability The advantages of LNI make it a preferable solution is several cases. The ability to communicate instructions in an online manner is suitable for scenarios requiring real time communication, such as deployed robotic agents and online decision systems. It also allows the human teacher to focus on communicating knowledge, while the technical aspects of LNI are handled offline, in a separate process.

It should be noted that LNI is not a complete alternative to statistical machine learning; in some learning domains taking the example-based learning approach is more appropriate, such as when the target decision function cannot be concisely represented. Consider for example problems of object detection in computer vision, in this case a decision rule for identifying an object given a scene represented as a collection of pixels cannot be given easily. In such cases statistical machine learning is the clear preferred alternative. LNI is suitable for cases in which knowledge can be effectively communicated, and described succinctly as a rule. This still leaves a wide array of relevant tasks and domains, these include Natural Language Interface to Databases (NLIDB), card games, robotic soccer, navigational instruction and many more.

Learning as Semantic Interpretation Successful learning in the LNI settings depends on the agent's ability to correctly decode instructional text. This process is known as semantic interpretation - moving from a lesson describing complex interactions in NL to a representation which grounds these interactions in a domain language. This problem is one of the core AI problems, and at a first sight, assuming a natural language decoder as a requirement for LNI seems to render it uninteresting. In order to avoid this predicament, we reduce our requirement from a general purpose natural language interpreter to an interpreter tailored for LNI. We follow most works on semantic parsing, and suggest to construct a semantic interpreter for a given domain of interest, using machine learning methods. The heart of this thesis is therefore finding realistic methods for constructing a semantic interpreter for LNI.

In this thesis we attempt to take a first step towards LNI. We suggest an experimental framework for LNI and examine what are the key challenges in accommodating LNI and suggest solutions aimed to minimize the human effort involved in it. In the rest of this chapter we explain these challenges and our contributions. We begin with a brief overview of semantic interpretation and current approaches to it. We then move to explaining the key challenges we tackle in this work and finally, describe the key ideas in this work.

1.1 Semantic Interpretation

Semantic parsing can be succinctly described as a process mapping natural language text into a formal meaning representation. Given a meaning representation language, capturing the elements of some application domain (e.g., a database), this process consider the possible interpretations of the input sentence selects the most appropriate one.

The interpretation process is done by composing meaning units into a full meaning representation, starting with the meaning of words, phrases, up to the entire sentence. Consider the example described in Figure 1.1, *"Illinois borders Iowa"*. Each of the words appearing in the sentence has a well defined meaning.

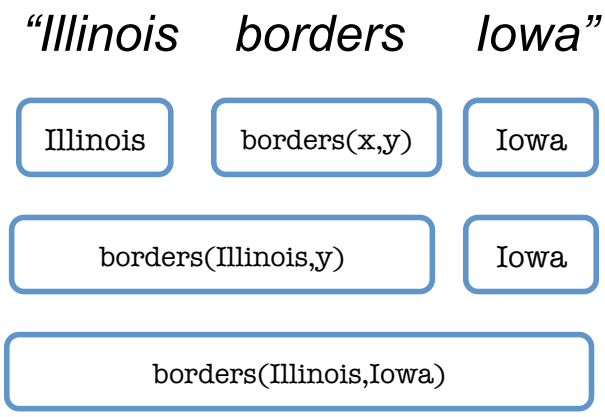


Figure 1.1: Example of compositional semantic interpretation steps. The meaning of the sentence is captured by composing the meaning if its constituents. In this example, taken from the Geoquery domain, the first row has the meaning of three constant symbols – the entities corresponding to Iowa and Illinois, and the function symbol $borders(\cdot)$. The value of the function symbol (and accordingly, the meaning of the word *borders* with respect to the given database) is the set of all pairs in the database for which the relation $borders(cdot)$ holds. In the second row, we consider the meaning of the phrase *"Illinois borders"*, which in this case refers to the set of all pairs in the database, such that the left hand side argument of the tuple is the constant symbol $Illinois$. Finally, the third row captures the meaning of the entire sentence, and assigns it a boolean value, which corresponds to the existence of the relation $borders(Illinois, Iowa)$ in the database.

The meaning of the entire sentence is revealed by composing the meaning of the words. In this example, the word *"Illinois"* refers to an entity, and the word *"borders"* refers to a function symbol. The meaning of the phrase *"Illinois borders"* is obtained by composing the meaning of the two individual words using function application.

Deconstructing the Interpretation Process The process of semantic interpretation can be broken down into two conceptual steps. Given a sentence, one step is to identify which concepts and entities appear in the interpretation. Theoretically the set of possible entities and functions is defined by the real world, or even broader the real world - the set of possible worlds (i.e., all logically sound sets of entities and relations). When approached from a computational perspective this definition is narrowed down to a finite set of logical symbols, typically used to describe a specific domain. This step consists of mapping words or short phrases to logical symbols. Most approaches employ a *lexicon*, a dictionary-like mapping between natural language words and phrases and logical entities.

The second step is identifying how the entities and relations appearing in the sentence's interpretation can be composed into a meaningful interpretation. Following the tradition of Montague semantics most approaches perform this step by mapping syntactic constructs into logical fragments incrementally. Constructing a computational system for this stage comprises of specifying a set of such mapping rules, these

can be done manually, as was done in early systems, and using machine learning methods in more recent systems.

Existing Work In this work we refer to the term *semantic interpretation* along the tradition of Montague semantics Montague [1970], in which the meaning of a sentence is represented using a logical language, and is constructed in a compositional manner, building up from the individual words in a sentence, to phrases, up to the entire sentence.

Early attempts to put this theory into practice consisted of manually construct systems, which mapped natural language sentences to a logical form in a very restricted domain according to a set of predefined rules, combining both lexical and syntactic considerations. Notable examples of that area include blocks world dialog system (Winograd [1972]), story understanding (Charniak [1972]) and prolog based systems (Pereira and Shieber [1987]).

More recent approaches take a supervised machine learning approach, in which a learning system is used to induce a set of parsing rules mapping between logical fragments and syntactic and lexical elements. A wide range of machine learning techniques have been employed for this task, such as inductive logic programming in early systems (Zelle and Mooney [1996]), support vector machines (Kate and Mooney [2006]) and various structured prediction algorithms (Zettlemoyer and Collins [2005]; Kwiatkowski et al. [2010]). These algorithms construct a semantic grammar mapping syntactic constituents to semantic constituents.

The difficulty of annotating training data for this task motivated work on using alternative methods of supervision. Several works study grounded language interpretation settings Chen and Mooney [2008]; Liang et al. [2009]; Chen and Mooney [2008, 2011]. These works learn the correspondence between language and meaning in a grounded settings using an ambiguously aligned streams of language and perceptual input. In Branavan et al. [2009] reinforcement learning is used to follow directions in a grounded world setting.

Our settings are not limited to grounded language interpretation. We drive language learning from the world's response. These settings were also studied more recently in (Clarke et al. [2010]; Liang et al. [2011]). We analyze and explain the difference between their approach and ours.

Leveraging textual instructions to improve game rules learning has been studied previously in (Eisenstein et al. [2009]) for Freecell solitaire. In that work textual interpretation was limited to mining repeating patterns and using them as features for learning the game rules over considerable amounts of game training data. Incorporating natural language advice in a game playing framework was also studied in (Branavan et al. [2011]). In their settings text interpretation is used to augment the state space representation in a

reinforcement learning framework.

1.1.1 Key Challenges

Minimizing Supervision Effort In recent years there have been several advances and success stories when applying supervised machine learning methods to this problem. In this set up the semantic-learner is given a set of examples - sentences paired with their logical interpretation. While advances are encouraging, from the LNI perspective, taking a supervised learning approach leaves us with a circular problem - learning the semantic interpreter could be even more difficult than the original learning problem we hoped to avoid. A major challenge on the road to making LNI a feasible framework is minimizing the supervision effort for learning semantic parsers.

Transferring Knowledge Across Domains A second issue that should to be considered is the domain-specific nature of this field. One of the major relaxations which make semantic parsing feasible is reducing the scope of semantic analysis to a relatively small set of logical symbols relevant for a specific domain. The target domain is typically a database system (e.g., geographical databases), simple dialogue systems (e.g., flight ordering system) and grounded directives (e.g., Robocup sport casting, navigational instructions). This approach, while needed to make the learning problem feasible, incurs the cost of learning semantic interpretation for every new domain. One of the key questions we ask in this thesis is how to generalize to new domains without restarting learning from scratch.

1.2 Overview of This Work

1.2.1 Minimizing Supervision Effort

Semantic interpretation can be viewed as a prediction process, given a set of words, produce a set of inter-related symbols that capture their semantics. This prediction problem can be considered as an instance of *structured prediction*, a process that maps complex inputs to output structures consisting of multiple decisions. In this work we suggest several methods for learning a semantic predictor in an indirectly supervised or even unsupervised settings. Our key contribution is identifying how the LNI settings can be used in order to extract a cheap supervision signal.

Response Driven Learning Response driven learning is a training regime for learning structured predictors which relies on an external source of supervision, that provides indication of correctness for predicted

structures without requiring gold structures. From a machine learning perspective, this amounts to structured learning using binary supervision, and requires learning algorithms that can learn in this new setting. We suggest a novel algorithm for learning in this setting, that combines both structural and binary learning principles. In order to accommodate the structure learning problem, the algorithm amplifies the binary feedback signal by learning to approximate the structural loss from previous predictions. This work was originally published in (Clarke et al. [2010]; Goldwasser and Roth [2011]) and is discussed in section 5.1.

Unsupervised Confidence Driven Learning for Semantic Parsing Taking this approach even further we suggest an unsupervised algorithm for semantic parsing. We attempt to replace the binary supervision provided when using response driven learning with a self-generated binary signal. This signal is generated using a confidence score computed over the model's predictions in an unsupervised way. We show that by decomposing the structures into individual decisions a reliable confidence score can be computed. This work was originally published in (Goldwasser et al. [2011]) and is discussed in section 6.

Learning Semantic Correspondence We also look into the grounded language acquisition settings, originally published in (Chen and Mooney [2008]). Most of the work done in this domain employed generative algorithms which constructed probabilistic models from weakly aligned data. We suggest a discriminative variant working in these settings. Although our algorithm works reasonably well, our key interest is to use this domain as a testing bed for extensions of our model. This topic is discussed in section 7.1.

1.2.2 New Intermediate Representation for Learning Semantics

The process of mapping between a natural language sentence and a formal meaning representation is defined over an intermediate layer which explains the semantic mapping decisions. In most published work on semantic interpretation, the interpretation is driven by learning a grammar (such as (Zettlemoyer and Collins [2005]; Wong and Mooney [2007])) and this representation consists of a derivation sequence mapping syntactic to semantic constituents. This rich model may not be suitable for the learning with weak supervision as in our model. We suggest a novel prediction model which relies on lexical decisions, while incorporating syntactic information, when needed, as part of the feature representation. We discuss this topic in section 4. We then further extend this representation to consider a deeper representation of syntactic and semantic information in 7.

1.2.3 Domain Independent Semantic Parsing

Finally we discuss how information can be shared between semantic parsers for different domains. We take an initial stab at this problem in section 7. The key idea behind the transfer mechanism is to add an additional intermediate layer capturing shallow semantic information. The new layer captures the semantic roles of arguments and predicates appearing in the input sentence. This new information facilitates the semantic interpretation decision, but in addition, since labeling semantic roles is not tied to a specific domain, once learned it can be reused for new domains. In order to avoid the cost of training such a model by annotating data for both tasks, we suggest a joint model for both tasks which uses the supervision signal for the semantic interpretation task in order to train the semantic roles labeler.

In addition, we show preliminary results suggesting that this layer, once learned over one semantic interpretation domain, can assist the interpretation decision in a different domain.

Chapter 2

Background

2.1 Machine Learning Background

This section provides the necessary machine learning background, it introduces the key machine learning concepts and algorithms used throughout this thesis. Most of the work in this thesis falls under the framework of *structured prediction*, or more accurately, adapting structured learning to a lightly supervised setting. Unlike the traditional structured learning protocols, which use annotated structures for learning, we suggest to learn from a supervision signal available from an external environment, such as a game-simulator. We will therefore cover some of the major topics leading up to structured learning and prediction, learning with latent variables and learning with indirect supervision.

The notation defined in this section and used throughout this thesis is defined in Table 2.1.

Table 2.1: Machine Learning Notation

Notation	Explanation
$\mathbf{w}$	Weight vector
$\mathbf{x}$	Input
y	Output (binary/multi-class case)
$\mathbf{y} = (y^1, y^2, ..., y^k)$	Output structure
$\mathbf{h} = (h^1, h^2, .., h^m)$	Latent information
$\Phi(\cdot) = (\phi^1(\cdot), \phi^2(\cdot), .., \phi^n(\cdot))$	Feature function, outputs a feature *vector*. Typically can be decomposed into individual feature functions
$\Phi(\mathbf{x})$	Feature function defined over the input only (binary/multi-class case)
$\Phi(\mathbf{x}, \mathbf{y})$	Joint feature function defined over input and output (structured case)
$\Phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$	Joint feature function over input,output and latent information
$\mathbf{y}(\mathbf{x}_i)$	All possible output structures for a given input
$\mathbf{h}(\mathbf{x}_i)$	All possible latent structures for a given input
$\mathbf{h}(\mathbf{x}_i, \mathbf{y}_i)$	all possible latent structures for a given input, while fixing the output
$\Delta(\mathbf{y}_i, \mathbf{y})$	Structural distance metric
$\ell : \mathbb{R} \rightarrow \mathbb{R}$	loss function (e.g., hinge loss)

Algorithm 1 The Perceptron Learning Algorithm.

Require: Learning rate η, Number of iterations N, Training Data $\mathcal{B} = \{(\mathbf{x}_i, z_i)\}_{i=1}^{l}$

1: $\mathbf{w} \leftarrow 0$
2: **for** $t = 1 \dots N$ **do**
3: **for** $i = 1 \dots l$ **do**
4: **if** $z_i \mathbf{w}^T \Phi(\mathbf{x}_i) \leq 0$ **then**
5: $\mathbf{w} \leftarrow \mathbf{w} + \eta(z_i \Phi(\mathbf{x}_i))$
6: **end if**
7: **end for**
8: **end for**

2.1.1 Binary Classification

Binary prediction is arguably the simplest prediction model and therefore it is an intuitive starting point for this review – given an input object classifying it to one of two classes is what is known as binary classification. This decision is typical formalized as a linear threshold function, $\mathbf{w}^T \Phi(\mathbf{x}) > 0$, where $\mathbf{w}$ is a real valued weights vector and $\Phi(\mathbf{x})$ is a feature function mapping an input object $\mathbf{x}$ to a vector of features.

The weight vector $\mathbf{w}$ can be learned given a set of labeled training examples (denoted $\mathtt{T}$). Many learning algorithms can be employed for learning. Two of the most widely used algorithms are the Perceptron algorithm Rosenblatt [1958] and Support Vector Machine (SVM) Boser et al. [1992].

The perceptron learning algorithm is described in Algorithm 1. The perceptron training algorithm is an online mistake-driven algorithm, which only updates the weight vector (line 5) when the model makes a mistake (line 4).

The perceptron algorithm works in an online fashion, iteratively going over the training example and updating the model according to the mistakes it makes. Unlike the perceptron algorithm, the SVM algorithm performs batch training, or more formally, it obtains the weight vector $\mathbf{w}$ by solving the following optimization problem:

$$\min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i \in \mathcal{B}} \max(0, 1 - z_i \mathbf{w}^T \Phi(\mathbf{x}_i)), \tag{2.1}$$

where $\mathcal{B} = \{(\mathbf{x}_i, z_i)\}_{i=1}^{l}$, and C is a parameter that prevents overfitting. That is, C controls the balance between the regularization term $\frac{\|\mathbf{w}\|^2}{2}$ and the loss term. Note that throughout this thesis, for brevity, we write $i \in \mathcal{B}$ to indicate $(\mathbf{x}_i, z_i) \in \mathcal{B}$.

While the perceptron algorithm and SVM are different learning algorithms, they do share the same prediction function: $\mathrm{sgn}(\mathbf{w}^T \Phi(\mathbf{x}_i))$. That is, the model will predict the example as a positive example if and only if $\mathbf{w}^T \Phi(\mathbf{x}_i) \geq 0$.

Binary classification, discriminating between two classes, can be extended to multi-class classification

Allwein et al. [2000], in this case the classifier is required to discriminate between *k* classes (k is typically a small constant).

2.1.2 Structure Learning

Many natural language processing tasks require taking multiple interdependent decisions. For example, assigning Part-of-Speech (POS) labels to a sentence requires taking several decisions- each word is an input object that should be labeled, however these labels are clearly related.

Training a classifier (a multi-class classifier for the POS case) and making these prediction independently is typically referred to as *local* prediction – each decision is taken locally, without global considerations. This contrast with *structured prediction*, in which all of these decisions are taken together, by optimizing a global objective. In Figure 2.1 we provide a detailed example describing these concepts for the POS task.

More formally, a structured predictor is a mapping $\mathbf{X} \rightarrow \mathbf{Y}$, where a $\mathbf{x} \in \mathbf{X}$ is a complex input object (e.g., a sentence), and $\mathbf{y} \in \mathbf{Y}$ is an output structure (e.g., a POS sequence). The output structure is a vector consisting of multiple decisions – $\mathbf{y} = (y^1, .., y^k)$. Most structured prediction models are formulated as an optimization process maximizing a linear mapping:

$$\underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_i)}{\arg\max}\, \mathbf{w}^T \Phi(\mathbf{x}_i, \mathbf{y}). \tag{2.2}$$

The dependency between decisions is expressed via the feature function, denoted as $\Phi(\mathbf{x}, \mathbf{y})$. Note that unlike the feature function for the local classifiers, $\Phi(\mathbf{x})$, which is defined over the input object alone, structured models use a *joint* feature function which takes both input and output objects. This formulation allows the model to capture features which span several decisions. Consider the example in Figure 2.1: in order to identify the correct sense of the second and third words, and assign the correct POS label, a larger context is needed. This context is formulated as a feature function which considers bigram of decisions, consisting of both the current decisions (y^i) and the previous decision (y^{i-1}).

Structured learning is concerned with finding a weight vector, such that when used in Equation 2.2, the correct structure will be the result of the optimization process. In the following section we will review two learning algorithms for structured learning. These algorithms are extensions of the perceptron and SVM algorithms, adapted for the structured settings.

Structured Perceptron The Structured Perceptron algorithm (Collins [2002]) is a natural extension of the binary perception algorithm to the structured case. The pseudocode is given in Algorithm 2. The algorithm uses the online mistake driven approach as in the binary case, with one key difference - it now employs the

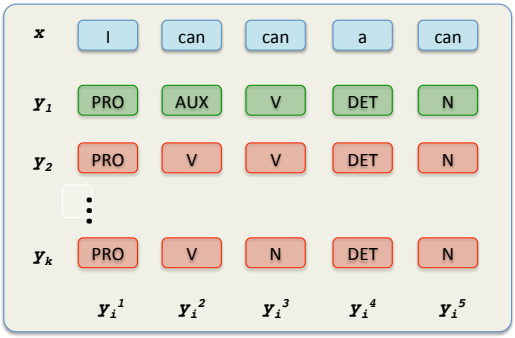


Figure 2.1: Example of structured prediction concepts for the POS task. The input sentence (denoted $\mathbf{x}$) defines the space of possible output structures. Each candidate structure (denoted $\mathbf{y}_i$) consists of multiple decisions (denoted $(y_i^1, y_i^2, .., y_i^k)$)

Algorithm 2 Structured Perceptron

Require: Number of iterations N, Training Data $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{l}$

1: $\mathbf{w} \leftarrow 0, \mathbf{w}_{avg} \leftarrow 0$
2: **for** $t = 1 \dots N$ **do**
3: **for** $i = 1 \dots l$ **do**
4: $\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_i)} \mathbf{w}^T \Phi(\mathbf{x}_i, \mathbf{y})$.
5: $\mathbf{w} \leftarrow \mathbf{w} + \Phi_{\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i)}$
6: $\mathbf{w}_{avg} \leftarrow \mathbf{w}_{avg} + \mathbf{w}$
7: **end for**
8: **end for**
9: **return** $\mathbf{w}_{avg}/(Tl)$

structured decision function instead of the binary one. Line 4 solves the maximization problem to find the optimal structure according to the current model. The weight vector is updated with the difference between the feature vectors of the true label and the prediction. The algorithm maintains an averaged weight vector (lines 6 and 9). Similar to the binary case using an averaged model has been shown to be superior and result in a model able to generalize better (Freund and Schapire [1999]).

Structural Support Vector Machines The Structural SVM (SSVM) introduced by (Tsochantaridis et al. [2004]), similarly to the structured perceptron algorithm, aims to find a weight vector for structured prediction, however unlike the structured perceptron, which is a mistake driven learning algorithm, which does not update the model unless a mistake is made, the SSVM algorithm aims to provide better generalization by introducing the notion of structural margin into the learning process– that is the score of the correct structure needs to be better than the next runner-up by a margin (unlike the perceptron which is satisfied if the correct structure is ranked first).

In order to accommodate this approach we first define a distance function: $\Delta(\mathbf{y}_i, \mathbf{y})$ which is 0 if and

only if two structures $\mathbf{y}_i, \mathbf{y}$ are identical and a positive number otherwise. The specific implementation of Δ depends on the output structure, a natural choice in many cases is the Hamming distance function which given two output structures, captures differences between finer grained structural decisions.

Training a SSVM entails solving the following global optimization problem:

$$\min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{l} L_S(\mathbf{x}_i, \mathbf{y}_i, \mathbf{w}), \tag{2.3}$$

where l is the number of labeled examples and $L_S(\mathbf{x}_i, \mathbf{y}_i, \mathbf{w})$ represents the loss function for the structured labeled examples. The function L_S can be written as

$$L_S(\mathbf{x}_i, \mathbf{y}_i, \mathbf{w}) = \ell(\max_{\mathbf{y}} \left[\Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \Phi_{\mathbf{y}_i, \mathbf{y}}(\mathbf{x}_i)\right]),$$

where $\Phi_{\mathbf{y}_i, \mathbf{y}}(\mathbf{x}_i) = \Phi(\mathbf{x}_i, \mathbf{y}_i) - \Phi(\mathbf{x}_i, \mathbf{y})$, and the function $\ell : \mathbb{R} \to \mathbb{R}$ can be instantiated by many commonly used loss functions such as hinge loss, with $\ell(a) = \max(0, a)$ or squared-hinge loss, with $\ell(a) = \max(0, a)^2$.

2.1.3 Learning with Latent Information

Structured prediction is a useful framework for many natural language processing learning tasks. The structured nature of linguistic constructs aligns well with the machine learning formulation. Unfortunately, this advantage comes with a cost - annotating sufficient data to capture the structural dependencies appearing in the data is a tedious and costly task. A useful extension to the supervised structured learning framework is concerned with learning from partial information. In this case we assume that the output is only partially labeled and more dependencies appear in the data, but are not explicitly annotated. As a clarifying example, consider the semantic interpretation task we are concerned with throughout this work. In this case the structured predictor is a mapping between a natural language sentence and a logical formula. Both the input and output are complex structured objects, consisting of several constituents (i.e., words and logical symbols), the mapping between the two objects depends on finding a mapping between the objects constituents. However, annotating these low level mappings is difficult, and more importantly - in many cases it is ill defined, and no clear a-priori mapping exist.

The preferred solution is to provide the learning algorithm with pairs of sentences and logical formulas, and allow the learning algorithm to find, and correctly parametrize, the low level correspondence required in order to find the mapping between these two complex objects. While this setup is sometime referred to

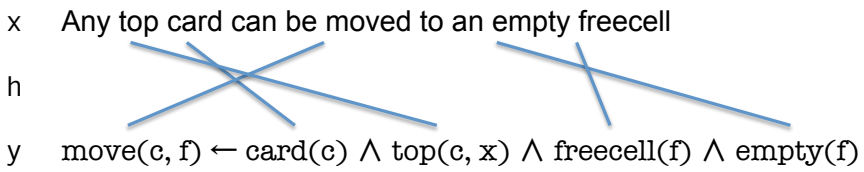


Figure 2.2: Example of structured prediction which relies on latent information. The mapping between an input object ($\mathbf{x}$ and an output object ($\mathbf{y}$) relies on a word level alignment. The correctness of the interpretation is not effected by the alignment decisions, however good alignments are more likely to generalize better, as they capture the correlation between words and predicate symbols.

as supervised learning, a more accurate characterization of it would be *learning with hidden (or latent) information*. We denote this missing information as $\mathbf{h} = (h^1, h^2, .., h^M)$, a vector consisting of decisions which do not appear in the annotated data, and more importantly - when evaluating the model's performance, these variables do not play a part directly (i.e., the correctness is not evaluated) but rather they provide additional information to support the output structure decision. Figure 2.2 exemplify this concept. In this case $\mathbf{h}$ is an alignment between the sentence's words and logical symbols. The output structure depends on this alignment.

Generally speaking, learning algorithms working in these setting aim to find a model that predict $\mathbf{h}$ reliably, to the extent that they support the final output decision. That is – the (annotated) output structure provides supervision for the latent decision. We will briefly describe two learning algorithms adapted for these settings - a latent variant of the structure perceptron algorithm, and a latent SSVM.

Latent Structured Perceptron Algorithm 3 describes the latent structure perceptron. In order to adapt the structure perceptron algorithm to the latent settings two modifications are needed. First, the maximization procedure done at line 4 now returns both the optimal output structure and assignments to the latent variables. Note that in this case the feature function $\Phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$ is now defined over the latent information as well, allowing the model to generate features that capture the dependence of the output decisions on the latent information. The second modification is concerned with the update stage (line 5), this operation consist of two stages, first compute the hidden variables values for the gold output structure, this is done by running the maximization process over the values of $\mathbf{h}$, while fixing the value of $\mathbf{y}$ to the gold structure. Or more formally: we denote $\arg\max_{\mathbf{h} \in \mathcal{H}(\mathbf{x}_i)} \mathbf{w}^T \Phi(\mathbf{x}_i, \mathbf{h}, \mathbf{y}_i)$ as $\mathbf{h}(\mathbf{x}_i, \mathbf{y}_i)$. Now that the complete information is available, the algorithm computes the feature values, and updates the model accordingly, as in the original version of structured perceptron. In Sun et al. [2009] a variant of this algorithm was presented and analyzed (however this algorithm was used prior to this paper).

Algorithm 3 Latent Structured Perceptron

Require: Number of iterations N, Training Data $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{l}$

$\mathbf{w} \leftarrow 0, \mathbf{w}_{avg} \leftarrow 0$
for $t = 1 \dots N$ **do**
  for $i = 1 \dots l$ **do**
    $\hat{\mathbf{y}}, \hat{\mathbf{h}} = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_i), \mathbf{h} \in \mathcal{H}(\mathbf{x}_i)} \mathbf{w}^T \Phi(\mathbf{x}_i, \mathbf{h}, \mathbf{y})$.
    $\mathbf{w} \leftarrow \mathbf{w} + \Phi_{\mathbf{y}_i, \mathbf{h}(\mathbf{x}_i, \mathbf{y}_i), \hat{\mathbf{y}}(\mathbf{x}_i), \hat{\mathbf{h}}(\mathbf{x}_i)}$
    $\mathbf{w}_{avg} \leftarrow \mathbf{w}_{avg} + \mathbf{w}$
  end for
end for
return $\mathbf{w}_{avg}/(Tl)$

Latent Structural SVM The latent structure equivalent of SSVM was presented in (Yu and Joachims [2009]), and provided a formulation of the new optimization function:

$$\min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2} + C \sum_i \left(\max_{\mathbf{y}, \mathbf{h}} \left(\Delta(\mathbf{y}_i, \mathbf{y}, \mathbf{h}) + \mathbf{w}^T \Phi(\mathbf{x}_i, \mathbf{y}, \mathbf{h}) \right) - \max_{\mathbf{h}} \left(\mathbf{w}^T \Phi(\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}) \right) \right), \tag{2.4}$$

Note that this function is very similar to the one used in SSVM, however it now uses a feature function defined over the latent information, and the maximization procedure also quantifies and predicts values to the hidden information.

It should be noted that unlike SSVM solving this optimization function is not trivial as it is a non convex function. We refer the reader to (Yu and Joachims [2009]) for further information about the optimization procedure used.

2.1.4 Learning with Indirect Supervision

Several recent works concerned with reducing the effort involved in training structured predictors have suggested using auxiliary sources of supervision. In this case we assume a supervision signal that is related but not directly applicable to the learning task at hand. The learning protocols described in this work are of that nature – the supervision signal is given for performance based on correct interpretation, not the interpretation itself.

Most similar to this approach is the work presented in Chang et al. [2010b]. This work describes who to use labeled *binary* examples when learning a structured task. In this case the learning problem is defined over both the annotated structural and binary data. The authors present a joint formulation of the SVM optimization function, minimizing the loss over both binary and structural information.

Chapter 3

Learning from Natural Instructions: Experimental Framework

Under a very broad definition, learning from natural instructions is concerned with communicating symbolic knowledge between a human teacher and an automated agent. This carries with it an immediate advantage, as it assists in focusing the human involvement on task-related expertise, rather than technical expertise. This advantage can come into play in several forms, in this work we focus on three key application domains which have been studied previously in the literature – learning decision rules from natural language instructions, natural language access to databases and interpreting natural comments in a grounded settings.

While the promise of this approach is clear, successful learning in these settings depends on correctly communicating relevant knowledge to an automated agent, which unfortunately proves to be a far from trivial task. This problem, often referred to as *semantic parsing*, is typically framed as a structured prediction task, mapping between natural language input and a formal meaning interpretation expressed in a logical language that can be understood by the target computer system.

Current approaches employ machine learning techniques to construct a semantic parser. The learning algorithm is given a set of input sentences and their corresponding meaning representations and learns a statistical semantic parser - a set of rules mapping lexical items and syntactic patterns to their meaning representation and a score associated with each rule. Given a sentence, these rules are applied recursively to derive the most probable meaning representation. While these approaches have been applied successfully to several semantic interpretation domains, they require a considerable annotation effort: pairing sentences with their corresponding logical meaning representation is a difficult, time consuming task.

In this work we aim to tackle the problem of minimizing the effort involved in supplying the supervision required for learning using alternative methods of supervision, and by suggesting a novel knowledge transfer model between different semantic parsing domains. We begin by introducing a *response-based* learning protocol, which uses a supervision signal obtained by observing the effect of using the interpretation in an actionable context. For example, in a card game domain, where NL sentences describe game rules, this signal is obtained by observing the outcome of playing the game using the interpretation of the NL rules.

We then introduce an unsupervised learning protocol for semantic interpretation.

In this chapter we explain the key elements of our framework and how it can be applied to different domains. We then explain the learning algorithm modifications that accommodate learning with this type of cheap feedback in chapter 5, and the knowledge transfer model in chapter 7.

3.1 Overall Framework

The purpose of our framework is to allow an automated agent to take natural language input and use it in its decision making process, by translating it into a machine understandable language. For example, the natural language input could describe a classification function, capturing an observable desired behavior of the learning system. However unlike traditional machine learning algorithms, which learn such functions from annotated examples, the agent should learn this function by interpreting correctly natural instructions describing the target concept.

In this chapter we give a bird's eye view of our framework implemented according to the principals of LNI, and describe its components. We begin by briefly defining the interpretation process and introducing the relevant notation in 3.2. The interpretation is defined as a structured prediction problem, an optimization problem selecting the proper interpretation for a given input sentence. We provide more details about this process and compare it to existing approaches in chapter 4.

The output of this interpretation process is a meaning representation of the input sentence, given in a logical language describing entities in a specific domain. In this work we consider three such domains - card games, geographic database queries and robotic soccer. These domains and the natural language inputs associated with them are described in section 3.2.2.

In our experiments evaluating the overall LNI approach, we used the Freecell solitaire card game, taking as input instructions describing the legality of game actions and used the instructions' interpretation to predict which moves are legal given specific game states. We also study different aspects of learning in these settings using Geoqeury and Robocup domains. Figure 3.1 describes examples of this process for the domains discussed in this thesis.

Machine learning in LNI, in its typical sense, pertains only to the agent's ability to better understand instructions. The supervision signal available to the learner for this learning process is derived from an external source, such as perceptual environment (which ambiguously relates to the natural language utterance) or a judgment of the system's behavior given the instruction interpretation (following the intuition that correct behavior corresponds to a correct interpretation of the input instruction). This weaker form

of supervision requires us to adapt the learning algorithm used. We describe our approach briefly in section 3.3, the response-based supervision signal in section 3.3.1 and our empirical evaluation in section 3.4. We provide a comprehensive explanation of this topic in chapter 5.

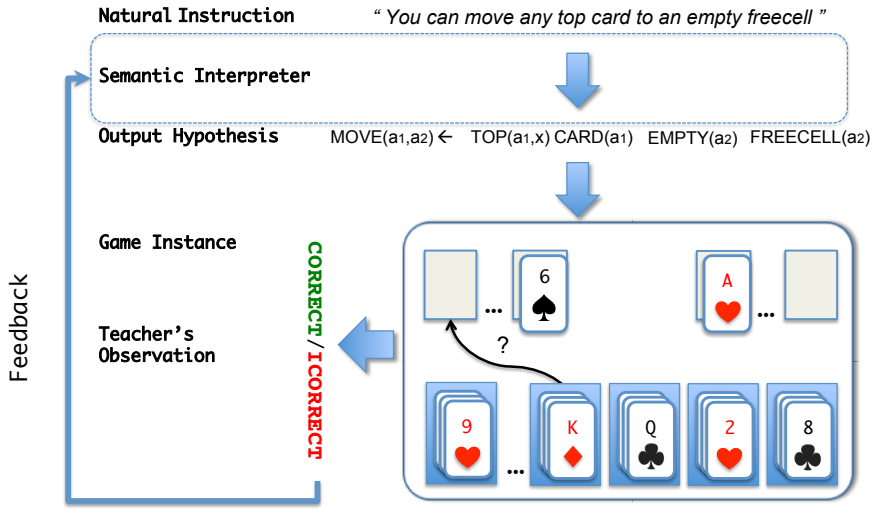
Finally, we describe how this work relates to other learning-by-instruction and semantic interpretation approaches in section 3.5.

3.2 Semantic Interpretation

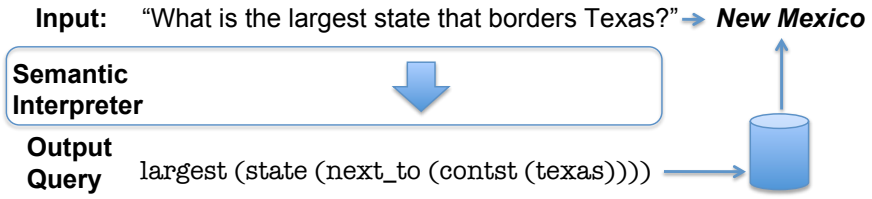
The process of semantic interpretation can be formulated as a function $\mathtt{I} : \mathtt{N} \rightarrow \mathtt{L}$, mapping a natural language sentence to its interpretation in a logical language. Since the input, a natural language sentence, is a complex and structured entity, this mapping inevitably relies on multiple decisions, capturing how each of the entities and relations mentioned in the input are represented in the output language.

There are several works discussing this mapping. Typically it is constructed by extracting parsing rules, which rely heavily on syntactic patterns, mapping natural language sentences to their logical meaning representation, restricting possible interpretations to previously seen syntactic patterns. This approach has been applied successfully in a supervised setting, where the learner has access to training data exemplifying the relevant patterns. However in our settings, in which annotated data is not available directly, this approach could be difficult to adapt.

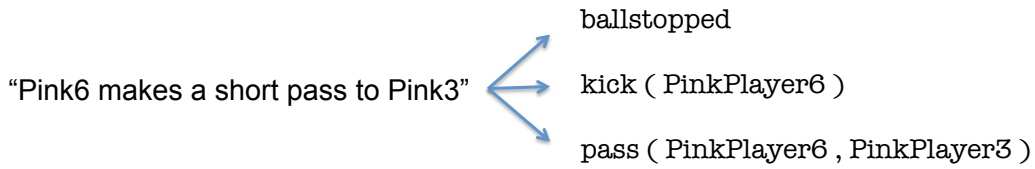
In order to account for the many syntactic variations associated with the output meaning representation, we propose a new model for semantic parsing that allows us to learn effectively. We replace the rigid inference process induced by the learned parsing rules with a flexible framework. We model semantic interpretation as a sequence of interdependent decisions, mapping text spans to predicates and use syntactic information to determine how the meaning of these logical fragments should be composed. We frame this process as an Integer Linear Programming (ILP) problem, a powerful and flexible inference framework that allows us to inject relevant domain knowledge into the inference process, such as specific domain semantics that restrict the space of possible interpretations. We present the relevant notation in the following subsection and describe the interpretation process in detail in chapter 4.



(a) LNI setup for the card game domain. The agent learns new game rules from NL, and tests its understanding by using these rules in a game. Human judgments about the correctness of the agent's game playing are used to drive the learning process for semantic interpretation.



(b) LNI setup for Geoquery, an NLIDB domain. The agent takes NL questions and converts them into logical database queries. These are used to query the database and obtain a result. Comparing these results with answers to the NL question drives the learning process for semantic interpretation.



(c) LNI setup for Robocup, a grounded language acquisition domain. The agent learns by observing the perceptual environment described by the natural language utterances.

Figure 3.1: Learning setup for language interpretation - from textual input to real world behavior.

3.2.1 Semantic Inference

We formulate semantic interpretation as a structured prediction problem, mapping a NL input (denoted $\mathbf{x}$), to its highest ranking logical interpretation (denoted $\mathbf{y}$). In order to correctly parametrize and weigh the possible outputs, the decision relies on an intermediate representation: an alignment between textual fragments and their meaning representation (denoted $\mathbf{h}$). Figure 3.2 exemplifies these concepts for the domains in discussed in this thesis.

In order to clarify these concepts, let's consider the mapping between the input text snippet "*top card*" and its interpretation in a logical language $\texttt{card(c)} \wedge \texttt{top(c, x)}$. In this case the input (x) consists of two input objects (x_1="*top*",x_2="*card*"), the output (y) consists of the following output decisions : y_1 = $\texttt{card}(\cdot)$, y_2 = $\texttt{top}(\cdot)$, and y_3 expressing the fact that both predicts are applied to the same variable. The lexical alignment (h) between the input and output objects consists of two elements -
h_1 = ("*top*", $\texttt{top}(\cdot)$) and h_2 = ("*card*", $\texttt{card}(\cdot)$).

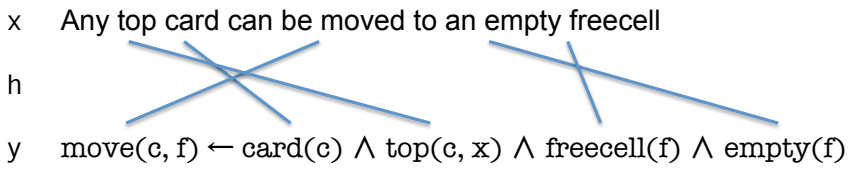
In our experiments, the input sentences $\mathbf{x}$ are natural language Solitaire game instructions, which describe the legality of possible actions, database queries or robotic soccer events descriptions. The output formula representation $\mathbf{y}$ is described using a formal language. We provide further details about it in section 3.2.2.

The prediction function, mapping a sentence to its corresponding interpretation, is formalized as follows:

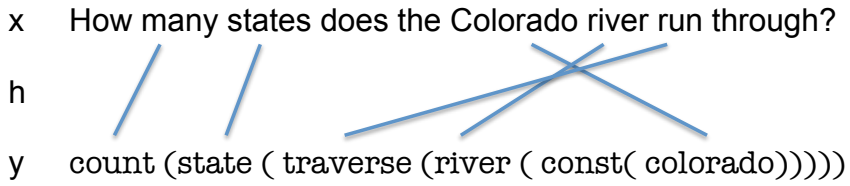
$$\hat{\mathbf{y}} = F_{\mathbf{w}}(\mathbf{x}) = \underset{\mathbf{h}\in\mathcal{H},\mathbf{y}\in\mathcal{Y}}{\arg\max}\ \mathbf{w}^T\Phi(\mathbf{x},\mathbf{h},\mathbf{y}) \tag{3.1}$$

Where Φ is a feature function defined over an input sentence $\mathbf{x}$, alignment $\mathbf{h}$ and output $\mathbf{y}$. This feature function maps $\mathbf{x}, \mathbf{h}, \mathbf{y}$ to a feature vector. The feature vector is simply an aggregation of the features used in the output decisions. The weight vector $\mathbf{w}$ contains the model's parameters, whose values are determined by the learning process. We refer the reader to chapter 2 for more details about these terms and structured prediction in general.

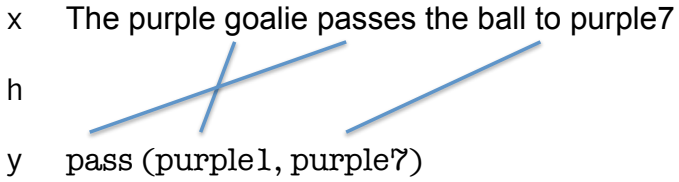
We refer to the $\arg\max$ above as the inference problem. Given an input sentence, solving this inference problem based on Φ and $\mathbf{w}$ is what compromises our interpretation process. section 4 provides more details about the feature representation and inference procedure used.



(a) Card game domain example



(b) Geoquery domain example



(c) Robocup domain example

Figure 3.2: Examples of natural language inputs (denoted $\mathbf{x}$), their corresponding logical forms (denoted $\mathbf{y}$) and the hidden intermediate representation (denoted $\mathbf{h}$)for three domains.

3.2.2 Meaning Representation

The output of semantic interpretation is a representation of the meaning input sentence, grounding the semantics of the input sentence in the relevant target domain (e.g., Solitaire card game, Geoquery or robotic soccer) using a logical language.

The target domain (denoted as **D** throughout the paper) defines the set of possible output symbols. These include typed constants (corresponding to specific cards, values, in the Solitaire domain, or states, cities, etc., in the Geoquery domain) and functions, which capture relations between domains entities, and properties of entities (e.g., $\texttt{value} : E \rightarrow N$). When viewed as a structured prediction process, the symbols in the target domain define the space of possible outputs.

The meaning of input sentences is expressed using the domain's symbols, combined using a fragment of first-order-logic. The complexity of the interpretation process is defined, among other things, by the level of expressivity the output language allows. Clearly, if it is possible to express meaningful statements in a given domain using a simple output language, it would help simplify the prediction process. The card game domain uses a general formulation, consisting of atomic symbols (defined above) and logical connectives. In the case of Geoquery and CLang we are able to express the database queries using a functional variable-free language (similar to Kate et al. [2005]). This restricts the space of possible outputs, thus reducing the complexity of the learning process. In the rest of this section we elaborate on these domains.

The Solitaire Card Game Our formulation of the Solitaire domain is an extended version of the Free-cell domain defined in the Planning Domain Definition Language (PDDL), which is used for evaluating automated planning systems.

The Solitaire domain consists of 87 constants and 14 predicates. These symbols are used to describe different attributes of cards and their location. A **game state** contains specific instantiations of the domain symbols, describing the relations between the entities in the state.

Given a game state, a logical formula defined over the domain symbols can be evaluated. For example, with regard to the game state described in Figure 3.1(a), the formula $\texttt{freecell}(\texttt{x}_1)\ \texttt{top}(\texttt{x}_2, \texttt{x}_1)\ \texttt{value}(\texttt{x}_2, 6)$, stating that a card with a value of 6 is on top of at least one freecell, will be evaluated to a `true`.

As can be observed in this example, dependency between the values of logical predicates is expressed via argument sharing. We use this mechanism to construct meaningful logical formulas from text, that can be evaluated given game states. Since our goal is to predict the legality of game actions (i.e., horn rules), we refer to the predicate corresponding to an action, `move`, as the **head** predicate and as a convention we denote its arguments as a_1, a_2. We define `args(p)`, to be a function mapping a predicate p to its list of

argument variables, and denote by p^i the i-th argument in this list.

Geoquery Database Queries Geoquery is a logical query language used to access a database of U.S geographical facts. The database was originally constructed by (Zelle and Mooney [1996]) using an extension of Prolog, and was later revised to use a simpler variable-free functional language (Kate et al. [2005]). In this work we also use a variable-free language.

The geoquery domain consists of constants, describing geographical locations (e.g., states, cities, rivers, etc.) and functions capturing relations in the database (e.g., neighboring states, population etc.). We refer the reader to (Zelle and Mooney [1996]) for more details about the Geoquery domain.

RoboCup Description Language This domain describes events in a robotic soccer match (RoboCup), and contains symbols describing these events. For example, constants include the players in all teams, and relations describe game events, such as losing the ball, passing the ball, etc. We refer the reader to (Chen and Mooney [2008]) for more details about the Robocup domain.

3.3 Learning

There are two forms of learning in LNI: the first refers to the knowledge gained by interpreting text, and the second to the ability to interpret text. In that sense LNI shifts the weight from learning the knowledge expressed in text, to learning a semantic parser for NL instructions describing it.

Unfortunately, interpreting natural language to the extent it can be used by a machine is not an easy task. Constructing a semantic interpreter manually or taking a supervised approach when learning the semantic parser results in an equally hard, if not a harder problem than the one we hoped to avoid.

Our learning framework avoids this difficulty by making the connection between the learning tasks explicit. In our settings, the learner has the ability to test its understanding. While this is clearly insufficient for constructing the target hypothesis, it allows the learner to reject incorrect lesson interpretations. We exploit this property to provide feedback to the semantic interpretation learning process, and base the learning algorithm on this feedback. In this paper we examine learning in such settings, where the output is executed by a computer program resulting with a response or outcome. We propose a response driven learning framework capable of exploiting feedback based on the response. This type of supervision is very natural in many situations and requires no machine learning expertise and thus can be supplied by any user.

For example, in the card game domain, this response can be generated by letting the agent *play the game*

using the classification rule it generated. We consider scenarios where the feedback is provided as a binary signal, correct +1 or incorrect -1. We describe the specific feedback mechanism for each domain in the following subsection.

This weaker form of supervision poses a challenge to conventional learning methods: semantic parsing is in essence a structured prediction problem requiring supervision for a set of interdependent decisions, while the provided supervision is binary, indicating the correctness of a generated meaning representation. To bridge this difference, we propose a novel learning algorithms suited to the response driven setting. We discuss learning in details in section 5 of this document.

3.3.1 Feedback from Real World Behavior

Our learning protocol relies on interacting with an external environment to provide supervision for learning. In this section we briefly describe how was this supervision obtained in each one of the domains we looked at in this work.

Throughout the paper we abstract over the implementation details and refer to the feedback mechanism as a binary function $\mathit{Feedback} : \mathcal{Y} \rightarrow \{+1, -1\}$, informing the learner whether a predicted logical form $\mathbf{y}$ when executed on the actual game, or geographical database, produces the desired outcome.

Solitaire Card Game Observing the behavior resulting from the card game instruction interpretation was the only feedback mechanism available to our learner. We envision a human learning process, in which in addition to high level instructions the learner receives a small number of examples used for clarification. For each concept taught, the learner had access to 10 positive and 10 negative labeled examples chosen randomly[1].

Geoquery In the Geoquery domain, we supplied the correct *answers*, and given the predicted logical meaning representation returned a positive feedback if the result of using predicted logical form to query a database was similar to the correct answer to the query.

Robocup In the Robocup domain no explicit supervision signal is provided, but it is derived by capturing the correspondence between two weakly aligned streams of natural language sentences describing soccer events and several robotic soccer events.

[1]Since our goal is to learn the target concept from instructions rather than from examples, we designed a weak feedback mechanism which cannot be used to learn the target concept directly

3.4 Empirical Evaluation

We test our approach in an actionable settings, in which the learner can evaluate its hypothesis by taking actions in a (simulated) world environment. We evaluate our learning algorithm and unique interpretation process on the well studied Geoquery domain (Zelle and Mooney [1996]; Tang and Mooney [2001]), a database consisting of U.S. geographical information and natural language questions. Although the textual input in this domain does not describe a classification rule, but rather a database query, it allows us to evaluate our learning algorithm and its properties, and more importantly, it allows us to compare it to existing semantic parsers that have studied this domain. We also evaluate our overall LNI approach over a set of Solitaire card game rules, in which the lessons describe preconditions on actions. We show that using our protocol an automated system can be taught to play games using NL instructions. Finally in the robocup domain we evaluate the model's ability to capture the correct mapping between a sentence and the correct event it describes, given multiple events.

3.5 Related Work

Human-Centric Learning Protocols In this work we study a novel learning protocol based on learning from instructions given by a human teacher. Instructable computing approaches leveraging human expertise have typically been studied in a reinforcement learning setting, in which a human teacher provides additional feedback to the learning process (a few recent examples include (Isbell et al. [2006]; Knox and Stone [2009]; Thomaz and Breazeal [2006])). The role of human intervention in our learning framework is different, as we simulate a NL lesson scenario. The approach closest to ours is described in (Kuhlmann et al. [2004]), integrating NL advice into a reinforcement learner. However, in their setting the language interpretation model is trained independently from the learner in a fully supervised process.

Machine Learning Approaches to Semantic Interpretation Converting NL into a formal meaning representation is referred to as *semantic parsing*. This task has been studied extensively in the natural language processing community, typically by employing supervised machine learning approaches. Early works (Zelle and Mooney [1996]; Tang and Mooney [2000]) employed inductive logic programming approaches to learn a semantic parser. More recent works apply statistical learning methods to the problem (Kate and Mooney [2006]; Wong and Mooney [2007]; Zettlemoyer and Collins [2005, 2009]). These works rely on annotated training data, consisting of sentences and their corresponding logical forms.

We learn to interpret NL instructions from game interaction feedback, instead of supervised learning.

Learning in similar settings for semantic interpretation has been studied recently by several works: (Chen and Mooney [2008]; Liang et al. [2009]; Branavan et al. [2009]) use an external world context as a supervision signal for semantic interpretation. However the semantic interpretation task is different than ours - the NL input is completely situated in an external world state, while in our case the NL input describes a high level rule abstracting over specific states.

Several recent works (Branavan et al. [2009]; Vogel and Jurafsky [2010]) look into learning language from an external world feedback using a reinforcement learning approach. Although there is some similarity between these works and the work presented in this document, our work differs conceptually from that line of work in an important aspect - the depth of the semantic output. In these works the authors suggest a combined interpretation and action model taking into account both linguistic input and action priority (according to behavioral rewards). In this model a semantic interpreter cannot be isolated from the action model, and therefore cannot be used independently. This contrast with our approach which treats semantic interpretation in isolation, and uses the world's behavior as feedback for improving interpretation, rather than embedding it into the interpretation process.

In (Artzi and Zettlemoyer [2011]) the structural loss is approximated according to conversational cues. Most relevant to our work are (Clarke et al. [2010]; Liang et al. [2011]; Goldwasser and Roth [2011]) which use these settings for learning a semantic parser. We show how to extend the learning protocol in order to better exploit the binary feedback. In (Liang et al. [2011]) the authors also utilize a response based mechanism for learning, however the focus of their work is on the semantic representation, rather than learning as in our work. We describe the differences between their learning procedure and ours in chapter 5.

The connection between structured prediction and binary classification over structural decisions was studied in Chang et al. [2010b]. In their settings a global optimization objective was defined over a *fixed* set of annotated structures and labeled binary examples. Our algorithm on the other hand does not have any annotated data, but rather creates its own dataset iteratively by receiving feedback.

Leveraging textual instructions to improve game rules learning has been studied previously in (Eisenstein et al. [2009]) for Freecell Solitaire. In that work textual interpretation was limited to mining repeating patterns and using them as features for learning the game rules over considerable amounts of game training data. Incorporating natural language advice in a game playing framework was also studied (Branavan et al. [2011]). In their settings text interpretation is used to augment the state space representation in a reinforcement learning framework.

Chapter 4

Semantic Inference

Semantic parsing is the process of converting a natural language input into a formal logic representation. This process is performed by associating lexical items and syntactic patterns with logical fragments and composing them into a complete formula. Most existing approaches rely on extracting a set of *parsing rules*, mapping text constituents to a logical representation, from annotated training data and applying them recursively to obtain the meaning representation. Adapting to new data is a major limitation of these approaches as they cannot handle inputs containing syntactic patterns which were not observed in the training data. For example, assume the training data produced the following set of parsing rules for the Geoquery domain:

Example 2. *Typical parsing rules*

(1) *NP* [λx.capital(x)] $\rightarrow$ *capital*

(2) *PP* [const(texas)] $\rightarrow$ *of Texas*

(3) *NNP* [const(texas)] $\rightarrow$ *Texas*

(4) *NP* [capital(const(texas))] $\rightarrow$

NP[λ x.capital(x)] *PP* [const(texas)]

Rules 1-3 describe lexical transformations, these rules are triggered by a raw input fragment (such as a word, or a short phrase) and generate a logical symbol and a syntactic category (such a Noun Phrase, Preposition Phrase etc.). Rule 4 describes a higher level transformation, in which two pairs consisting of a syntactic category and a matching logical fragment are unified into a single syntactic category and logical formula.

Given a sentence (such as the ones in Example 3) the meaning of a sentence is constructed by applying these rules recursively. It can be observed that despite the lexical similarity in these examples, the semantic parser will correctly parse the first sentence but fail to parse the second because the lexical items belong to different a syntactic category (i.e., the word *Texas* is not part of a preposition phrase in the second sentence). The third sentence will fail to parse due to missing lexical information–the term "Longhorn State" is not covered by any of the rules.

Example 3. *Syntactic variations of the same meaning representation*

Target logical form: `capital(const(texas))`

Sentence 1: *"What is the capital of Texas?"*

Sentence 2: *"What is Texas' capital?"*

Sentence 3: *"What is the capital of the Longhorn State?"*

The ability to adapt to unseen inputs is one of the key challenges in semantic parsing. Several works (Zettlemoyer and Collins [2007]; Kate [2008]) have suggested partial solutions to this problem, for example by manually defining syntactic transformation rules that can help the learned parser generalize better. Most similar to our approach is the work of (Kate and Mooney [2006]) which relies more heavily on lexical information, and as a result proved to be more resilient to noisy data.

Given the previous example (sentence 2), we observe that it is enough to identify that the function `capital(·)` and the constant symbol `const(texas)` appear in the target logical interpretation, since there is a single way to compose these entities into a single formula–
`capital(const(texas))`.

Motivated by this observation we define our meaning derivation process as a search over possible interpretation, legal according to the domain language, and use syntactic information as a way to bias the logical interpretation process. That is, our inference process considers the *entire* space of meaning representations irrespective of the patterns observed in the training data. This is possible as the logical interpretation languages are defined by a formal language and formal grammar.[1] The syntactic information present in the natural language is used as soft evidence (features) which guides the inference process to good meaning representations.

In addition we use existing external knowledge resources capturing lexical information to make up for missing lexical information. In this case (sentence 3), mapping between "Texas" and "Longhorn State".

This formulation is a major shift from existing approaches that rely on extracting parsing rules from the training data. In existing approaches the space of possible meaning representations is constrained by the patterns in the training data and syntactic structure of the natural language input. Our formulation considers the entire space of meaning representations and allows the model to adapt to previously unseen data and *always* produce a semantic interpretation by using the patterns observed in the input.

We frame our semantic interpretation process as a constrained optimization process, maximizing the objective function defined by Equation 3.1 which relies on extracting lexical and syntactic features instead of parsing rules. In the remainder of this section we explain the components of our inference model.

[1] This is true for all meaning representations designed to be executed by a computer system.

4.1 Semantic Interpretation as Constrained Optimization

Semantic interpretation, as formulated in Equation 3.1, is an inference procedure selecting the top ranking output logical formula. In practice this decision is decomposed into smaller decisions, capturing local mappings of input tokens to logical fragments and their composition into larger fragments. These decisions are converted into a feature representation by a feature function Φ, and parameterized by a weight vector.

We formulate the inference process over these decision as an Integer Linear Program (ILP), maximizing the overall score of active decisions, subject to constraints ensuring the validity of the output formula. The flexibility of ILP has previously been advantageous in natural language processing tasks (Roth and Yih [2007]; Martins et al. [2009]) as it allows us to easily incorporate constraints declaratively. These constraints help facilitate learning as they shape the space of possible output structures, thus requiring the learned model's parameters to discriminate between a smaller set of candidates. The flexibility offered by using an ILP solver comes with a computational cost – ILP is known to be computationally costly (exponential in the size of the inference problem). In this work we used an off-the-shelf solver incurring the full computational cost, which while efficient for the most part, restricted us in some of the experiments we performed. We consider using approximation and ILP-relaxation techniques as future work.

4.2 Decision Variables and Objective Function

The inference decision is defined over two types of decision variables. The first type, referred to as a **first-order decision**[2], encodes a lexical mapping decision as a binary variable α_{cs}, indicating that a constituent c is aligned with a logical symbol $\mathtt{s}$. The pairs connected by the alignment (**h**) in Fig. 4.1(a) are examples of such decisions.

The final output structure **y** is constructed by composing individual predicates into a complete formula, this is formulated as an *argument sharing decision* indicating if two functions take the same variable as input. We refer to this type of decisions as a **second-order decision**, encoded as a binary variable, β_{cs^i,dt^j} indicating if the j-th argument of $\mathtt{t}$ (associated with constituent d) and the i-th argument of $\mathtt{s}$ (associated with constituent c) refer to the same variable or constant symbol. For example, the decision variables representation of the formula presented in Fig. 4.1(b): $\mathtt{move(a_1, a_2)top(a_1, x_2)}$ include an active second-order variable indicating that the corresponding predicates share an argument.

[2]Note that the use of the terms *first-oder* and *second-order* to describe interpretation decisions refers to unigram and bigram (a decision dependent on other decisions) output decisions, and **not** to first and second-order logic

Objective Function given an input sentence, we consider the space of possible semantic interpretations as the space of possible assignments to the decision variables. The semantic interpretation decision is done by selecting a subset of variables maximizing a linear objective function. The objective function quantifies different value assignments to the decision variables, using a feature function mapping a decision into a feature vector. We denote the feature function mapping first-order decisions (α) as Φ_1, and second-order decisions as Φ_2. These functions are described in more details in section 4.4. The objective function is defined as follows -

$$F_{\mathbf{w}}(\mathbf{x}) = \arg\max_{\alpha,\beta} \sum_{c\in\mathbf{x}} \sum_{s\in D} \alpha_{cs} \cdot {\mathbf{w_1}}^T \Phi_1(\mathbf{x}, c, s) + \sum_{c,d\in\mathbf{x}} \sum_{s,t\in D} \sum_{i,j} \beta_{cs^i,dt^j} \cdot {\mathbf{w_2}}^T \Phi_2(\mathbf{x}, c, s^i, d, t^j) \quad (4.1)$$

where i, j iterate over $\texttt{args(s)}$[3] and $\texttt{args(t)}$ respectively, and D is the set of logical symbols in the domain.

Note, that for a given input sentence many of the decision defined above are mutually exclusive, for example, if a word is mapped to a predicate p_i, it cannot be mapped to a different predicate p_j. We constrain the possible assignment to the objective function by adding linear constraints. The following section defines the constants used to ensure a legal assignment.

4.3 Constraints

Given an input sentence, the space of possible interpretations is subsumed by the space of possible assignments to the decisions variables. For example, there is a clear dependency between α-variables and β-variables assignments, as functions can only share a variable (β decision) if they appear in the output formula (α decisions). In order to prevent spurious assignments, we restrict the decision space. We take advantage of the flexible ILP framework, and encode these restrictions as global constraints over Equation 4.1.

Lexical Mapping Decisions

- An input constituent can only be associated with at most one logical symbol.

$$\forall c \in \mathbf{x},\ \textstyle\sum_{s\in\mathbf{D}} \alpha_{cs} \leq 1$$

- The head predicate (e.g., `move`) must be active.

[3]Recall that we define $\texttt{args(p)}$ to be a function mapping a predicate p to its list of argument variables

$$\sum_{c \in \mathbf{x}} \alpha_{c,move} = 1$$

Argument Sharing Decisions

- Variable sharing is only possible when variable types match.
- If two predicates share a variable, then these predicates must be active.

$$\forall c, d \in \mathbf{x}, \forall s, t \in \mathbf{D},\ \beta_{cs^i,dt^j} \implies \alpha_{cs} \wedge \alpha_{dt}$$

  where i, j range over `args(s)` and `args(t)` respectively.

Global Structure: Connectivity Constraints In addition to constraints over local decision we are also interested in ensuring a correct global structure of the output formula. We impose constraints forcing an overall *fully-connected* output structure, in which each logical symbol appearing in the output formula is connected to the head predicate via argument sharing. This property ensures that the value of each logical construct in the output is dependent on the head predicate's arguments. In order to clarify this idea, consider the output logical formula described in example 4. The value of that formula, when given a game state, is evaluated to TRUE if the game state contains at least one vacant freecell, *not necessarily the target freecell* specified by the head predicate arguments.

Example 4 (Disconnected Output Structure)**.** *Free variable not bound to input arguments.*

$\texttt{move}(\texttt{a}_1, \texttt{a}_2) \leftarrow \texttt{top}(\texttt{a}_1, \texttt{x}_1) \wedge \texttt{card}(\texttt{a}_1) \wedge \texttt{freecell}(\texttt{x}_2) \wedge \texttt{empty}(\texttt{x}_2)$

Note that this constraint does not limit the model's effective expressivity as it only rules out spurious interpretations. From a natural language processing point of view, this process is very similar to the co-reference resolution problem of ensuring that all mentions refer back to an originating entity. In this case we assume that there are two concrete entities - the action's arguments, and the constraints ensure that all other variables are linked to these entities.

We encode the connectivity property by representing the decision space as a graph, and forcing the graph corresponding to the output prediction to have a connected tree structure using flow constraints. Fig. 4.1(c) provides an example of this formulation.

Let $G = (V, E)$ be a directed graph, where V contains vertices corresponding to α variables and E contains edges corresponding to β variables, each adding two directional edges. We refer to vertices corresponding to $\alpha_{\texttt{c,move}}$ variables as head vertices. Clearly, the output formula will be fully-connected if and only if the graph corresponding to the output structure is connected. We associate a flow variable f_{cs^i,dt^j} with every edge in the graph, and encode the following constraints over the flow variables to ensure that the resulting graph is connected.

- Only active edges can have a positive flow.

$$\beta_{cs^i,dt^j} = 0 \implies f_{cs^i,dt^j} = 0 \wedge f_{dt^j,cs^i} = 0$$

- The total outgoing flow from all head vertices must be equal to the number of logical symbols appearing in the formula.

$$\sum f_{*,\texttt{move}^i,*,*} = \sum_{c \in \mathbf{x}} \sum_{s \in \mathbf{D} \setminus \{\texttt{move}\}} \alpha_{cs}$$

  For readability reasons, we use * to indicate all possible values for constituents in $\mathbf{x}$ and logical symbols in $\mathbf{D}$.

- Each non-head vertex consumes one unit of flow.

$$\forall d \in \mathbf{x}, \forall t \in \mathbf{D}, \ \sum f_{*,*,dt^j} - \sum f_{dt^j,*,*} = 1$$

4.4 Features

The inference problem defined in Equation (4.1) uses two feature functions: Φ_1 for first-order decision and Φ_2 for second-order decisions. In general, Φ_1 represents lexical information while Φ_2 represents syntactic and semantic dependencies between sub-structures. Extracting some of these features require using additional resources (such as a dependency tree, WordNet), we mention the use of these external resources below in Tables 4.1 and 4.2, and evaluate their importance in the following chapter.

Linguistic Resources Used for Feature Extraction

- *WordNet-based Lexical Similarity (WNSim)* Models for semantic interpretation typically utilize a lexicon containing *known* mappings between words and logical domain-symbols. The space of possible interpretations is restricted de-facto, to inputs containing lexical items appearing in the lexicon. In an idealized settings, identifying semantic relatedness between the *known* mappings and previously unseen words and phrases,could help the model generalize better to previously unseen inputs.

  We use a WordNet based similarity metric (WNSIM, see (Do et al. [2009]) for more details). The metric is defined over the WordNet hierarchy, given two words w_1, w_2, the metic computes the closest common ancestor of the words. The similarity is computed based on that distance, with some consideration to the semantic relation between words (e.g., the antonymy relation incurs a negative penalty when computing this score).

  Identifying semantic similarity between lexical items offers a useful, though noisy contribution to semantic interpretation. The true meaning of words is dependent on their context, rather than an iso-

lated matching between words. Furthermore, semantic relatedness is a very coarse semantic measure indicative of semantic equivalence in some cases only. This fact prevents us from using WNSim directly to extend the lexicon, but rather we use it to extract a set of features (described in the subsequent paragraph) and learn weights capturing the resource's contribution to the learning process.

- *Dependency Parser* Compositional decisions require taking into consideration a larger context. This context typically takes into account syntactic information. Most works account for this context as part of the training process, by learning the syntactic categories aligned with semantic fragments. In this work we take a different approach and utilize an external resource, a dependency parser trained independently (Klein and Manning [2003]), and use it to provide syntactic cues for the interpretation process.

  A dependency parser provides a syntactic view of the input sentence consisting of asymmetric lexical relations (i.e., dependencies) between words. These relations take the shape of a tree, starting from a single head node. In this work we use this structure to provide a more meaningful measure of distance (compared to one based on order of occurrence).

First-Order Decision Features Φ_1 Determining if a logical symbol is aligned with a specific constituent depends mostly on lexical information. Following previous work (e.g., Zettlemoyer and Collins [2005]) we create a small lexicon, mapping logical symbols to surface forms. We initialize the lexicon by mapping the logical entities to lexical item carrying the same name (e.g., $\texttt{card}(\cdot)$ was mapped to "card"). We extend the lexicon during the learning process–whenever the model predicts a logical formula assigned a positive feedback, the lexical choices done by the model are added to the lexicon. For example, given an alignment $\mathbf{h}$ that maps the word "neighboring" to the predicate $\texttt{border}(\cdot)$, the pair will be added to the lexicon.

We use an external knowledge base, WordNet (Miller et al. [1990]), to extend the initial lexicon and add features which measure the lexical similarity between a constituent and a logical symbol's surface forms (as defined by the lexicon). In order to disambiguate preposition constituents, an additional feature is added. This feature considers the current lexical context (one word to the left and right) in addition to word similarity. The list of features used for first-order decisions is summarized in table 4.1.

Second-Order Decision Features Φ_2 Second-order decisions rely on syntactic information. In order to extract these features we use a dependency parser, trained independently (Klein and Manning [2003]), and extract features from the dependency tree of an input sentence. Given a second-order decision $\beta_{cs,dt}$, the dependency feature takes the normalized distance between the head words in the constituents c and d.

Feature	Description	Resource
LexiconMatch(w,p)	Does the stem of the word w match an entry in the lexicon for p	Stemmer, Lexicon
NullPredicate(w)	Does the word w map to an empty symbol	-
Lexicon+Context(w,p)	If w is in lexicon for p, does the left/right window contain a lexicon entry	Lexicon
WordNetSimilarity(w,p)	Top WordNet based similarity score for w and all lexicon items for p	WordNet
PrepositionContext	If the left/right context of a lexicon item is a preposition	POS tagger

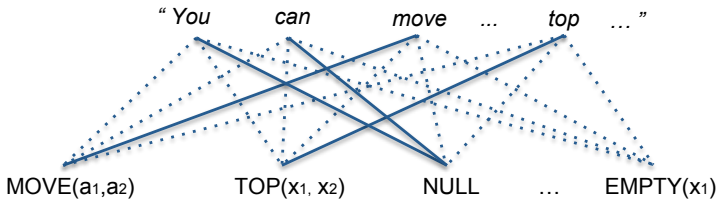
Table 4.1: Summary of features used for first-order decisions, and the resources utilized for extracting those features.

Feature	Description	Resource
DependencyDistance(w_1, w_2)	num of hops between w_1 and w_2	Dependency parser
Bigram($p1, p2$)	Occurrence of p_1, p_2 in output	-
TopPredicate(p_1)	p_1 is the top most predicate	-

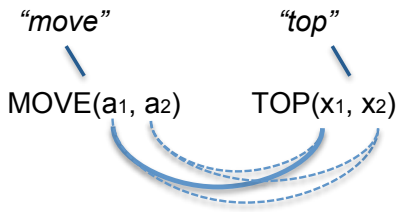
Table 4.2: Summary of features used for second-order decisions, and the resources utilized for extracting those features.

In addition, a set of features indicate which logical symbols are usually composed together, without considering their alignment to text. These feature allow the model to recover in cases where the input sentence contains unknown lexical items. Since no lexical evidence is available, these features allow the model to take an informed guess based on previous decisions. Finally, when using a functional output language (such as in Geoquery) we add a feature indicating that a given function symbol p is the *head* of the formula.

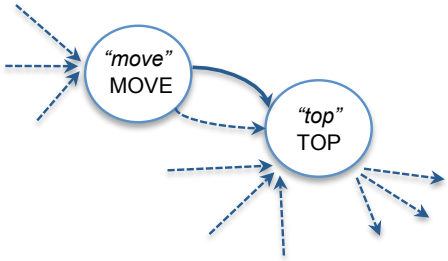
The list of features used for second-order decisions is summarized in table 4.2.



(a) 1^{st}-order decisions. These decisions (denoted as α variables in Eq. 4.1) capture possible mappings between words and domain symbols. In this example, the dashed lines represent possible mappings, and the solid lines represent the variables active in the output formula.



(b) 2^{nd}-order decisions. These decisions (denoted as β variables in Eq. 4.1) describe how to compose the symbols together. In this example the dashed lines represent an identity decision: if the symbols a_i and x_j refer to the same entity.



(c) Flow variables. Unlike the previous variables, these variables are do not appear in the objective function, and are only used to ensure the legality of the output. We restrict the possible output graphs to be connected. We ensure this property by framing it as a flow problem in a graph

Figure 4.1:
An example of inference variables space for a given input. The dashed edges correspond to non-active decision variables and the bold lines to active variables, corresponding to the output structure $\texttt{move}(a_1, a_2)$ $\texttt{top}(a_1, x_2)$. Active variables include 1^{st}-order: $\alpha_{(\text{“}move\text{”},\texttt{move}^1)}$, $\alpha_{(\text{“}top\text{”},\texttt{top}^1)}$, 2^{nd}-order:$\beta_{(\text{“}move\text{”},\texttt{move}^1),(\text{“}top\text{”},\texttt{top}^1)}$, and positive flow:$f_{(\text{“}move\text{”},\texttt{move}^1),(\text{“}top\text{”},\texttt{top}^1)}$

Chapter 5

Learning from Indirect Environment-Generated Supervision

In this work we advocate the idea of instructable computing, in which an automated learning system is taught a concept via direct instructions rather than by examples. The role of traditional machine learning is therefore shifted from learning the target concept to learning to interpret natural instructions, which explain the target concept. In this chapter we explain how can this learning process be done without explicit supervision.

5.1 Response Driven Learning

The learning problem is defined as finding a good set of parameters for the inference function described in Eq. 3.1, such that when applied to NL instructions the corresponding output formula will result in a correct behavior. Typically, such prediction functions are trained in a supervised settings in which the learner has access to training examples, consisting of the input sentences and their corresponding logical forms $\{(\mathbf{x}^l, \mathbf{z}^l)\}_{l=1}^N$ (e.g., Zettlemoyer and Collins [2005]; Wong and Mooney [2007]). However, our learning framework does not have access to this type of data and relies only on feedback obtained from world interaction–by trying out the interpretation. This setup gives rise to our algorithmic learning approach, iteratively performing the following steps- generating rules from instructions, receiving feedback by acting in the world and updating the interpretation function parameters accordingly. We explain the technical aspects of this protocol in the following subsection.

Providing this indication of correctness is typically cheap, alas, it does not exemplify the patterns appearing in the correct prediction, as required by structured learning algorithms. More formally, learning in these settings can be framed as learning a structured predictor[1] using a binary supervision source. Therefore, the key question is **how to use the available binary supervision signal to generate structured feedback signal required for training?**

[1]The learning domains discussed in this paper rely on an intermediate layer, **h**, which is not included in the training set. Our algorithm deals with this variation effortlessly, however when working in a supervised settings a latent variable variation of these algorithms is required. See (Yu and Joachims [2009]; Chang et al. [2010a]) for details.

In this section we aim to answer this question. Our discussion is driven by the observation that the key difficulty of bridging the gap between the binary supervision available and the structured pairs required for learning, stems from the *fault assignment problem*. Supervised learning is guided by a basic principle–minimizing a loss function, defined over the annotated data. In structure learning, the loss function quantifies the incorrect structural decision, penalizing (or assigning fault) only to substructures predicted incorrectly. However in our settings, the learner only has a binary indication of the incorrect prediction, rather than the structural decomposition required. A simple solution (taken for example by (Clarke et al. [2010]; Liang et al. [2011])), focuses on learning only from predictions receiving positive feedback, by extracting structured signal from these predictions and iteratively training the model.

Unlike these approaches, our algorithm is designed to use both types of feedback–by using positive feedback as structural supervision, and negative as binary supervision, thus utilizing the supervision signal more effectively. In Section 5.1.3, we present an online learning algorithm combining both binary learning and structured learning principles. In its most basic form, our algorithm utilizes the negative feedback signal in a coarse way, by penalizing the entire structure (i.e., all the structural decisions) uniformly. In Section 5.1.4 we take an extra step, and show how to amplify the binary signal by approximating the mistakes in the predicted structures leading to the negative feedback, thus allowing the algorithm to use finer grained feedback. We follow the observation that the predicted structure can be naturally decomposed to multiple components, some of which can be predicted more reliably than others. We propose an algorithm that better exploits the structural decomposition; specifically, we suggest to approximate the loss (i.e., the incorrect decisions) associated with structures assigned negative feedback by decomposing the structure into individual components and assessing their correctness probability. We can estimate the correctness probability reliably by using the set of positive predictions to compute their statistics.

5.1.1 Learning Structures from Binary Feedback

Learning to predict structures using binary supervision requires a new kind of learning algorithm, bridging the gap between the available binary supervision, and the required supervision - annotated examples. Since the binary supervision can be used as an indication of prediction correctness, learning algorithms of this type use structures labeled with a binary signal to train the model, by promoting output structures that result in a positive signal and the vice-versa.

In the rest of this section we describe learning in these settings in detail. We begin by providing an overview of existing approaches to this problem, these typically rely only on positive feedback by framing the problem as an incremental supervised learning problem. We then proceed to describe our algorithm,

Algorithm 4 Existing Approaches: High level view

Require: Inputs $\{\mathbf{x}^l\}_{l=1}^N$,
$Feedback : \mathcal{X} \times \mathcal{Y} \rightarrow \{+1, 1\}$,
initial weight vector $\mathbf{w}$

1: **repeat**
2: **for** $l = 1, \ldots, N$ **do**
3: $\hat{\mathbf{h}}, \hat{\mathbf{y}} = \arg\max_{\mathbf{h},\mathbf{y}} \mathbf{w}^T \Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y})$
4: $f = Feedback(\mathbf{x}^l, \hat{\mathbf{y}})$
5: add $(\Phi(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}), f)$ to B
6: **end for**
7: $\mathbf{w} \leftarrow TRAIN(B)$
8: **until** Convergence
9: **return** $\mathbf{w}$

and show how to use both structures assigned positive and negative labels. Finally we show how the negative binary feedback can be better exploited to provide finer-grained feedback to the learner instead of uniformly demoting the negative structure.

5.1.2 Existing Approaches for Learning with Binary Feedback

In Alg. 4 we describe a high level procedure for these settings. The algorithm incrementally samples the space of possible structures for a given input, modifying the model's parameter vector to encourage correct structures and reject incorrect ones. The algorithm repeatedly performs two steps -

1. Predict output structures using current model and receive feedback for them (lines 1-6)
2. Train the model based on that feedback (line 7).

Given the training set collected at the first step, existing algorithms perform the second step by doing one of the following -

Structured update Since positive feedback correspond to correct structural decision, these can be used directly for training a supervised model. This instantiation of the algorithm is in essence an incremental supervised learning algorithm, where at each stage more labeled examples are added to the training set. The choice of structured learning procedure used is left to the system designer, for example previous works used structured SVM (Clarke et al. [2010]) and the Expectation Maximization (EM) algorithm (Liang et al. [2011]).

While this procedure is very intuitive, it can only utilize structures assigned positive feedback and ignores the negative feedback.

Binary update Cast the learning problem as a binary classification problem, where the feature decomposition of correct structures is treated as positive examples and incorrect structures as negative. The weight vector is updated using a binary learning algorithm. This approach allows using both types of feedback, however it is used very coarsely, without taking advantage of structural information available for structures assigned a positive label.

5.1.3 Combined Feedback Perceptron

Our algorithm is designed to make use of both the negative and positive feedback, by combining ideas from both structure and binary learning. Unlike supervised learning algorithms, which take a fixed set of training examples, the algorithm iteratively generates its own training data, using its current set of parameters to generate structures, and the feedback function to label them. The algorithm can be considered as a fusion of the binary perceptron (Rosenblatt [1958]) and structured perceptron (Collins [2002]) and works in an online fashion, performing error driven updates.

Algorithm 5 describes this procedure. The algorithm iteratively samples the space using the current parameter set (line 4) and gets feedback for the chosen structure (line 5), a negative feedback incurs a penalty - if the learner, in any of the previous iterations, has generated a positive structure for that input object, the parameter set is updated towards that structure using the structured perceptron update rule (line 8), and if not, the parameter set is updated away from the selected point by using the binary perceptron update rule (line 10)[2].

The algorithm performs two types of updates, *structural update* in case a positive structure for that example as already been encountered by the algorithm. In this case the algorithm simply performs the structured perceptron updated rule, given as -

Structural Update

$$\mathbf{w} = \mathbf{w} + \Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y}) - \Phi(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}) \tag{5.1}$$

Alternatively, in case the algorithm does not have a positive structure to update towards, the algorithm demotes all active features uniformly, as described by the following equation.

Simple Binary Update

$$\mathbf{w} = \mathbf{w} - \Phi(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}) \tag{5.2}$$

[2]The number of negative structures is likely to overwhelm the learning process, to prevent that we restrict the number of binary updates to the size of the positive training set.

Algorithm 5 Combined Feedback Perceptron

Require: Sentences $\{\mathbf{x}^l\}_{l=1}^N$,
$Feedback : \mathcal{X} \times \mathcal{Y} \rightarrow \{+1, 1\}$,
initial weight vector $\mathbf{w}$

$B_l \leftarrow \{\}$ for all $l = 1, \ldots, N$
repeat
  for $l = 1, \ldots, N$ **do**
    $\hat{\mathbf{h}}, \hat{\mathbf{y}} = \arg\max_{\mathbf{h},\mathbf{y}} \mathbf{w}^T \Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y})$
    $f = Feedback(\mathbf{x}^l, \hat{\mathbf{y}})$
    if $f =$ -1 **then**
      if B_l contains $\Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y})$ **then**
        $\mathbf{w} \leftarrow$ Update using Structure$(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}})$
      else
        $\mathbf{w} \leftarrow$ Binary Update$(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}})$
      end if
    else
      add $(\Phi(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}))$ to B_l
    end if
  end for
until Convergence
return $\mathbf{w}$

The binary update rule treats the entire structure as incorrect. The rationale behind it is that since no gold structure to update towards exists, the "blame" for the negative structure is assigned uniformly. In the following section we describe how to refine this information by approximating which substructures are actually responsible for the negative signal.

5.1.4 Learning to Approximate Structural Loss

The binary update rule targets a simple objective - updating the current set of parameters, such that the incorrect structure will no longer be ranked first by the model. This is achieved by uniformly demoting the weights of all features corresponding to active decisions in the incorrect structure. This contrasts with the structured update rule, which aims to "correct" the decision, rather than just change it, by demoting the weights of features corresponding to incorrect decision, and promoting the weights of those corresponding to correct ones. In the supervised settings, these updates are possible since the gold structure provides this information.

While this information is not available, we try to approximate it by learning a structural loss function based on the set of correct predictions. Our loss function has a probabilistic interpretation, mapping each structural decision in to a probability score. Given an incorrect structure, we use the approximated loss function and update the model according to the score it assigns to structural decisions. Decisions assigned a low probability score are more likely to correspond to the mistakes that have led to the incorrect prediction.

The approximation is done by following the observation that the predicted structure is composed of several components, some more likely to be correct than others based on their occurrence in the positive data. More formally, given an incorrect structural decision, we decompose the structure into a set of sub-structures, and evaluate the conditional probability of each structural decision ($y_i \in \mathbf{y}$) given the input generating it[3]. We use these probabilities to appropriately penalize the model - if the decision is likely to be correct (i.e., it is assigned a high probability) the demotion step will have little effect on the weights of features corresponding to that decision and the vice versa.

Accommodating this procedure requires two changes to Algorithm 5. The first change is to the Binary update rule described in line 13 of Algorithm 5. The uniform penalty is replaced by a procedure described in Alg. 6, which assigns a different penalty to each structural decision based on its confidence score. The second change concerns the computation of the confidence score. This score is computed over the set of examples assigned a positive feedback signal. We change line 13 in Algorithm 5, to consider newly added positive structures in the loss approximation computation. Our algorithm maintains a probability distribution over the individual decisions compromising the positively predicted structures. Given a new positive structure we decompose it into the individual output decisions and the relevant part of the inputs, and the probability distribution over the two is updated.

In order to understand the intuition behind this approach consider the following (sentence,interpretation) pair.

Example 5. Input*:*

"What is the population of the largest city in New York?"

Output*:*

`population( largest (city ( const(new_york_city )))`.

In this example the output structure will receive a negative feedback since its interpretation of the term "New York" refers to New York city, rather than the state of New York. By focusing the update step on that decision we can encourage the model to try a different interpretation of that specific term, rather than penalizing potentially correct substructures in the output formula. Continuing with our example, since "New York" is an ambiguous term, the probability mass is likely to be divided between all the senses of the term, when computed over the positive training data, and therefore the probability score p(`const(new_york_city )`) | "New York") will reflect it, leading to a heavier penalty assigned to this term.

We observe that the success of our loss approximation method is dependent on the structural decom-

[3]Deciding which parts of the input should be considered is determined by observing which parts of the input were considered by the feature functions active for this substructure.

Algorithm 6 Approximated Structural Update

Require: Input $(\mathbf{x}, \mathbf{y})$
1: **for** $(x, y)_i \in (\mathbf{x}, \mathbf{y})$ **do**
2: $\mathbf{w} = \mathbf{w} - \phi(x, y) \cdot (1 - p(y|x))$
3: **end for**

position specific for each domain. In domains where the structural decomposition considers *local* feature functions, computing the conditional probability is likely provide useful information, as these local input-output substructures are likely to appear frequently in the data (as in the example above, where the lexical mapping decision depended only on the term "New York", which is likely to appear in other examples). An extreme opposite scenario could occur when each substructure relies on the entire input. Fortunately, linguistic input decomposes naturally in to smaller units over which the structural decision is defined, thus allowing our loss approximation approach to provide useful information during learning.

In this section we describe our experimental evaluation. We begin this section by describing our experiments on instructional text for several variations of the Solitaire card game. In this domain we evaluated the results of applying our overall approach, teaching an agent the rules to a card game,treating the predicted structure as a classification rule and evaluating the result on card game moves.

In the second part of this section we focus on response based learning, and evaluate our approach on the well known Geoquery domain. In this case the predicted structure is a database query that can be used to query a database. This well studied domain allows us to compare our results to existing work and evaluate different properties of response based learning.

5.1.5 Overall Approach: Teaching Solitaire Card Game Rules

Experimental Setup The decision function described by the text classifies the legality of several Solitaire card game moves given a game state. We consider several games, each having one or more such rules. All the rules describe a similar operation - moving a card from one location to another. The rules differ according to the game and the source and target cards locations. Consider for example two such rules for the famous Freecell solitaire game - FREECELL (move a card to a freecell) and TABLEAU (move a card to a tableau).

Example 6 (FREECELL concept and its description)**.** .

$\texttt{move}(\texttt{a}_1, \texttt{a}_2) \leftarrow \texttt{top}(\texttt{a}_1, \texttt{x}_1) \wedge \texttt{card}(\texttt{a}_1) \wedge \texttt{freecell}(\texttt{a}_2) \wedge \texttt{empty}(\texttt{a}_2)$

- "You can move any of the top cards to an empty freecell"
- "Any playable card can be moved to a freecell if it is empty"

Example 7 (TABLEAU concept and its description). .

$\texttt{move}(\texttt{a}_1, \texttt{a}_2) \leftarrow \texttt{top}(\texttt{a}_1, \texttt{x}_1) \wedge \texttt{card}(\texttt{a}_1) \wedge \texttt{tableau}(\texttt{a}_2) \wedge \texttt{top}(\texttt{x}_2, \texttt{a}_2) \wedge \texttt{color}(\texttt{a}_1, \texttt{x}_3) \wedge \texttt{color}(\texttt{x}_2, \texttt{x}_4) \wedge$
$\texttt{not} - \texttt{equal}(\texttt{x}_3, \texttt{x}_4) \wedge \texttt{value}(\texttt{a}_1, \texttt{x}_5) \wedge \texttt{value}(\texttt{x}_2, \texttt{x}_6) \wedge \texttt{successor}(\texttt{x}_5, \texttt{x}_6)$

- "A top card can be moved to a tableau if it has a different color than the color of the top tableau card, and the cards have successive values"

In order to evaluate our framework we associate with each target concept instructional text describing the target rule, and game data over which the predicted structures are evaluated. Each target concept is associated with 10 different instructions describing the target rule, and 900 relevant game moves sampled randomly. To avoid bias the game moves contain an equal number of positive and negative examples. Note that these examples are used for evaluation only.

We evaluated the performance of our learning system by measuring the proportion of correct predictions for each of the target concepts on the game data. The accuracy for each target concept is measured by averaging the accuracy score of each of the individual instruction interpretations.

The semantic interpreter was initialized using a simple rule based procedure, assigning uniform scores to input constituents appearing in the lexicon (first-order decisions) and penalizing second-order decisions corresponding to input constituents which are far apart on the dependency tree of the input sentence.

Experimental Approach Our experiments were designed to evaluate the learner's ability to generalize beyond the limited supervision offered by the feedback function. Generalization is evaluated along two lines: (1) Evaluating the quality of the learned target concept: the ability of the system to perform well on unseen solitaire game data (2) Evaluating the quality of the learned semantic interpretation model. Our goal goes beyond learning a single task: the semantic interpreter is meant to provide a broader interpretation ability, to the extent it can be applied, after learning one task, to other tasks defined over a similar lexicon. To study this scenario, after the learning process terminates, the system is given a set of new textual instructions belonging to a different game, and its performance is evaluated based on the quality of the newly generated rules. To accommodate this set up we performed a 5-fold cross validation over the data and report the averaged results.

Results Our results are summarized in table 5.1 and 5.2, the first describing the ability of our learning algorithm to generalize to new *game data*, and the second, to *new instructions*.

A natural baseline for the prediction problem is to simply return FALSE (or TRUE) regardless of the input–this ensures a baseline performance of 0.5 . The performance achieved without learning (using the

initialized model) improves over overly simplified baseline, in some cases, considerably, in others– just barely. After learning results consistently improves for all rules.

As can be noticed in the examples above, target concepts have different levels of difficulty–for example, the FREECELL concept is relatively easy compared to the other rules, both in terms of the output structure and the text used to describe it. The results indeed support this observation, and performance for this task is excellent. The other tasks are more difficult, resulting in a more modest improvement; however, the improvement due to learning is still clear.

In the second scenario we tested the ability of our semantic interpreter to generalize to previously unseen tasks. In this case successful learning depends on the different learning tasks sharing similar constructs and lexical items. We can expect performance to drop in this case, even simply for the reason that the semantic interpreter is trained using less data. Nonetheless the question remains - does LNI depend on *developing a reading ability*, or is the improvement after learning due only to the guidance the game data provides. If the latter is the case, we can expect performance to drop significantly as in these new settings the system is evaluated on the classification rule it generated after training, without feedback from the domain while constructing it.

Table 5.2 shows the results for this set up. It can be observed that while there is some performance drop, generally our learning algorithm is also able to learn a good semantic interpreter, which generalizes well to previously unseen instructions. Interestingly, the difference in performance compared to the first setting (table 5.1) differs depending on which rule is tested. Simple rules achieve good score in both cases, and their performance does not change. We hypothesize that this is due to the fact that the text corresponding to these rules is relatively simple and the model can predict the correct rule even without game data. As rules get harder, a specialized learning process is required, this can be observed by the drop in performance. Most notably in the case of ACCORDION THREE CARDS GAP which uses a predicate not used in any of the other rules.

5.1.6 Understanding Learning with Binary Supervision: Geoquery Domain

In order to get a better understanding of our response based learning procedure we used the Geoquery dataset, and compared the performance of several variations of our algorithm, trained over datasets of varying sizes. The dataset we used contains 250 queries used for training, and additional 250 used for testing.

Target Concept	Initial Model	Learned Model
FREECELL FREECELL	0.76	0.948
FREECELL HOMECELL	0.532	0.686
FREECELL TABLEAU	0.536	0.641
ACCORDION CONSECUTIVE LEFT	0.64	0.831
ACCORDION THREE CARDS GAP	0.561	0.724
AGREEMENT	0.74	0.932
ALHAMBRA KING	0.61	0.786
ALHAMBRA ACE	0.632	0.764
ACES UP BETWEEN	0.76	0.924
ACES UP OUT	0.591	0.691

Table 5.1: Results for Solitaire game rules. Each rule is described by the game it belongs to (larger font), and the specific rule name in that game (smaller font). Accuracy was evaluated over previously unseen game moves using the classification rules learned from the instructions used in training. The Initial Model column describes the performance of the rules generated by the initial interpretation model (i.e., before learning).

Target Concept	Initial Model	Learned Model
FREECELL FREECELL	0.76	0.948
FREECELL HOMECELL	0.532	0.679
FREECELL TABLEAU	0.536	0.635
ACCORDION CONSECUTIVE LEFT	0.64	0.817
ACCORDION THREE CARDS GAP	0.561	0.678
AGREEMENT	0.74	0.932
ALHAMBRA KING	0.61	0.763
ALHAMBRA ACE	0.632	0.753
ACES UP BETWEEN	0.76	0.924
ACES UP OUT	0.591	0.631

Table 5.2: Results for Solitaire game rules. Each rule is described by the game it belongs to (larger font), and the specific rule name in that game (smaller font). Accuracy was evaluated over previously unseen game moves using classification rules generated from *previously unseen game instructions*. Semantic interpretation was done using the learned semantic interpreter.

5.1.7 Empirical Results

We compare the results obtained by our algorithm to two natural reference points. The first is the initial model used to bootstrap learning (denoted INITIAL MODEL), improving significantly over this baseline is required to demonstrate effective learning. The second reference point is a supervised model, trained over the same inputs, however using annotated structures rather than binary feedback. We measure performance using the Accuracy score. We refer to our combined learning algorithm as COMBPERCEPT, and when applied with loss approximation we use the term COMBPERCEPT W APRXLOSS.

The results for the Semantic Interpretation domain are summarized in table 5.3. The initial model used for bootstrapping (INITIAL MODEL), resulted in an accuracy score of 0.222. Using our algorithm, without using loss-approximation, resulted in an accuracy score of 0.796, and with loss-approximation, performance was improved to - 0.816.

This model outperforms other systems working under this learning scenario that use the same dataset.

Algorithm	Sup.	Acc.
INITIAL MODEL	—	0.222
BINARY SUPERVISION (SEPARATE)		
CLARKE ET AL. [2010] (BIN)	250 ans.	0.692
CLARKE ET AL. [2010] (ST)	250 ans.	0.732
LIANG ET AL. [2011]	250 ans.	0.789
BINARY SUPERVISION (COMBINED)		
COMBPERCEPT	250 ans.	0.796
COMBPERCEPT W APRXLOSS	250 ans.	**0.816**
SUPERVISED		
CLARKE ET AL. [2010]	250 strct.	0.804
WONG AND MOONEY [2006]	310 strct.	0.6
WONG AND MOONEY [2007]	310 strct.	0.75
ZETTLEMOYER AND COLLINS [2005]	600 strct.	0.793
ZETTLEMOYER AND COLLINS [2007]	600 strct.	0.861
WONG AND MOONEY [2007]	800 strct.	0.866

Table 5.3: Results for the semantic interpretation domain, comparing several models learning with Binary supervision. Our approach outperforms these models when trained over the same datasets and resources. Note that our combined model always outperform competing models using only one type of feedback, and that loss-approximation results in an improved best performance.

We first consider the two models used by (Clarke et al. [2010]) - the first uses both types of feedback, using binary updates only (see Learning section for details) and achieves a score of 0.692, the second, uses structural updates, but utilizes only the positive feedback achieves a score of 0.732. The model presented by (Liang et al. [2011] combines an effective semantic inference procedure with a learning procedure utilizing only structures receiving positive feedback. Their model resulted in a score of 0.789 for this dataset, bootstrapped using the similar lexical resources[4].

There are many works operating in supervised settings. Our model outperforms the supervised model presented in (Clarke et al. [2010]), and other supervised models (Wong and Mooney [2006, 2007]) when trained over dataset of a similar size. This is not surprising, since during training, as the model collects more positive training examples, it converges towards a supervised learning framework.

In order to get a better understanding of the improvement achieved by the loss-approximation framework our algorithm uses, we compared the results of training the two variations of our algorithm (with and without the loss approximation) on datasets of varying sizes. Fig 5.1 presents the results of these experiments and shows consistent improvement when using loss-approximation.

5.1.8 Features and Linguistic Resources Study

The process of semantic interpretation consists of two types of decisions - lexical and compositional. Briefly described, the first type of decisions capture the possible mappings from natural language constituents to

[4](Liang et al. [2011]) presents an additional model which uses a larger manually constructed lexicon, resulting in a considerable improvement. Since we did not have the additional information we did not compare to it.

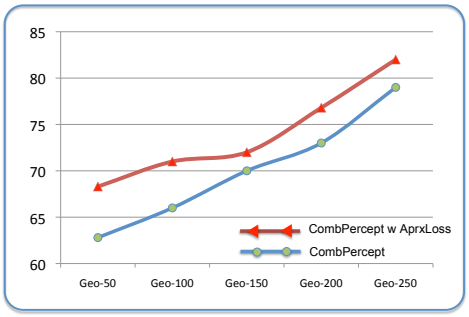


Figure 5.1: Comparing the results of the two variations of our combined-perceptron algorithm - with, and without loss-approximation, over datasets of increasing size in the semantic interpretation domain. Results show consistent improvement when using loss-approximation.

logical symbols, and the second type capture how these symbols should be composed together to create a meaningful statement capturing the meaning of the input.

As described in Chapter 4, the first type of decision relies mostly on lexical information and indeed, the feature decomposition of this type of decision captures different lexical aspects. The second type requires higher level information, such as syntactic information. The representation used for learning in both first and second type of decisions is composed of features that can be extracted directly (e.g., word occurrence for lexical decisions, and symbol bigrams for compositional decisions) and features functions utilizing external linguistic resources such as Wordnet for lexical decisions and a dependency parser for compositional decisions.

In this section we study the contribution of each one of those resources in the overall learning process. The importance of this study is two fold. First, it help us assess the importance of these resources in the learning process. Second, since many of these resources are only available for English, it is therefore useful to understand how much can be learned based on the information appearing in the data alone. Finally, we observe that there are strong dependencies between the two types of decisions. It is useful to get an understanding of how much each component (lexical vs. compositional contributes to the learning process. In order to check this we ran several experiments, testing the performance of different systems, using different representations.

- **Expanding the Lexicon Using WordNet** This scenario tests the importance of a lexical external resource - WordNet. This feature attempts to increase the lexicon derived from data using an external resource. This resource is used to quantify the semantic similarity of an input word to known lexicon words. The hope is that using this resource a small seed of words can be extended to help the system deal with previously unobserved expressions. We denote the system not making use of this resource

as **NoWordNet**.

- **Capturing Relevant Lexical Context Using Part-of-Speech Tagger** We exploit our knowledge of English and a part-of-speech tagger to encode if an input word occurs close to a preposition. We denote the system not making use of this resource as **NoPOS**.
- **Using Syntactic Dependencies to Determine Semantic Composition** Compositional decisions are often projected in the syntactic structure of the input sentence. Hoping to avoid the complexity of constructing a rule based system mapping between syntactic structures and semantic ones, we use a simple distance metric derived from calculating the distance between nodes in the dependency tree of the input sentence. We denote the system not making use of this resource as **NoDependency**.
- **Using Output Bigrams as Features** We observe that some predicate tend to co-occur with other predicates. We capture the preferences using bigram features, encoding which output symbols are composed together, regardless of any input information. We denote the system not making use of this feature type as **NoBigram**.
- **No Compositional Features** This system relies on lexical features only. Non of the compositional decisions features are used. We denote the system not making use of these feature types as **NoCompInf**.
- **No External Resources** This systems does not use any external resources. Specifically, it does not use the part-of-speech tagger, WordNet and dependency parser used by other systems. We denote the system not making use of all resources types as **NoResources**.

The experimental results are summarized in table 5.4. Observing the results revels the following: The importance of using external resources over this dataset is not crucial. This is demonstrated over both lexical and syntactic resources. In the lexical case, not using WordNet features reduces performance slightly. This can be explained by the fact that Geoquery is a closed domain, which uses a restricted vocabulary. Since there are other means to increase the size of the lexicon the WordNet features contribution is limited. We hypothesize the when using a larger dataset, in which the test data can include a substantial number of previously unseen word the contribution of this feature will increase. We tested this hypothesis by removing this feature and not allowing the system to increase its lexicon, in this case it can be observed that the performance loss increases (appearing in parentheses in table 5.4). Features utilizing a part-of-speech tagger did not help improve performance. While part-of-speech is an important source of information, our features use it only to identify prepositions. We leave other ways of exploiting this resource to future work.

The importance of using a dependency parser, syntactic resource, is also not clear. Removing features which use it did not hurt performance significantly. We hypothesize that this is due to redundancy –

Representation	Accuracy
FULL MODEL	0.816
NOWORDNET	0.792 (0.744)
NOPOS	0.816
NODEPENDENCY	0.804
NOBIGRAM	0.668
NOCOMPINF	0.632
NORESOURCES	0.776

Table 5.4: Results of several systems, using different representations, over the Geoquery dataset. The different systems use a weaker representation of the learning domain, either by not using external knowledge resources, or by not encoding different aspects of the input. The different systems are explained in Sec. 5.1.8. The results of these systems are compared to the full model, utilizing all information and resources (denoted **FULL MODEL**)

other features capture this information and are able to make up for it. Indeed, when comparing a system using only dependency information for compositional decisions (**NOBIGRAM**) to a system which uses no compositional information (**NOCOMPINF**), the first system performs better, thus demonstrating that features which rely on this resource contribute when used in isolation. Finally, we tested the system when stripped of all external knowledge sources, this resulted in considerable performance degradation.

When removing the Bigram features the system's performance drops dramatically, this suggests that the learning system is able to capture domain specific preferences.

Chapter 6

Unsupervised Confidence Driven Semantic Parsing

In this chapter we take another step towards alleviating the supervision effort, and present the first unsupervised approach for this task. Our model compensates for the lack of training data by employing a self training protocol based on identifying high confidence self labeled examples and using them to retrain the model.

We base our approach on a simple observation: semantic parsing is a difficult structured prediction task, which requires learning a complex model, however identifying good predictions can be done with a different model, allowing the learner to identify robust patterns appearing in the predicted data and use them to further training the model. We present several simple, yet highly effective confidence measures capturing such patterns, and show how to use them to train a semantic parser without manually annotated sentences.

Our basic premise, that predictions with high confidence score are of high quality, is further used to improve the performance of the unsupervised training procedure. Our learning algorithm takes an EM-like iterative approach, in which the predictions of the previous stage are used to bias the model. While this basic scheme was successfully applied to many unsupervised tasks, it is known to converge to a sub optimal point. We show that by using confidence estimation as a proxy for the model's prediction quality, the learning algorithm can identify a better model compared to the default convergence criterion.

We evaluate our learning approach and model on the well studied Geoquery domain (Zelle and Mooney [1996]; Tang and Mooney [2001]), consisting of natural language questions and their prolog interpretations used to query a database consisting of U.S. geographical information. Our experimental results show that using our approach we are able to train a good semantic parser without annotated data, and that using a confidence score to identify good models results in a significant performance improvement.

Using Confidence Estimation to Compensate for Lack of Training Data Our learning framework takes a self training approach in which the learner is iteratively trained over its own predictions. Successful application of this approach depends heavily on two important factors - **how to select high quality examples**

to train the model on, and **how to define the learning objective so that learning can halt once a good model is found.**

Both of these questions are trivially answered when working in a supervised setting: by using the labeled data for training the model, and defining the learning objective with respect to the annotated data (for example, loss-minimization in the supervised version of our system).

In this work we suggest to address both of the above concerns by approximating the quality of the model's predictions using a confidence measure computed over the statistics of the self generated predictions. Output structures which fall close to the center of mass of these statistics will receive a high confidence score.

The first issue is addressed by using examples assigned a high confidence score to train the model, acting as labeled examples.

We also note that since the confidence score provides a good indication for the model's prediction performance, it can be used to approximate the overall *model* performance, by observing the model's total confidence score over all its predictions. This allows us to set a performance driven goal for our learning process - return the model maximizing the confidence score over all predictions. We describe the details of integrating the confidence score into the learning framework in Sec. 6.1.

Although using the model's prediction score (i.e., $\mathbf{w}^T\Phi(\mathbf{x},\mathbf{y},\mathbf{z})$) as an indication of correctness is a natural choice, we argue and show empirically, that unsupervised learning driven by confidence estimation results in a better performing model. This empirical behavior also has theoretical justification: training the model using examples selected according to the model's parameters (i.e., the top ranking structures) may not generalize much further beyond the existing model, as the training examples will simply reinforce the existing model. The statistics used for confidence estimation are different than those used by the model to create the output structures, and can therefore capture additional information unobserved by the prediction model. This assumption is based on the well established idea of multi-view learning, applied successfully to many NL applications (Blum and Mitchell [1998]; Collins and Singer [1999]). According to this idea if two models use different views of the data, each of them can enhance the learning process of the other.

The success of our learning procedure hinges on finding good confidence measures, whose confidence prediction correlates well with the true quality of the prediction. The ability of unsupervised confidence estimation to provide high quality confidence predictions can be explained by the observation that prominent prediction patterns are more likely to be correct. If a non-random model produces a prediction pattern multiple times it is likely to be an indication of an underlying phenomenon in the data, and therefore more likely to be correct. Our specific choice of confidence measures is guided by the intuition that unlike

Algorithm 7 Unsupervised Confidence driven Learning

Require: Sentences $\{\mathbf{x}^l\}_{l=1}^N$,
initial weight vector $\mathbf{w}$
define $Confidence : \mathcal{X} \times \mathcal{H} \times \mathcal{Y} \rightarrow \mathcal{R}$,
$i = 0,\ S_i = \emptyset$
repeat
  for $l = 1, \ldots, N$ **do**
    $\hat{\mathbf{h}}, \hat{\mathbf{y}} = \arg\max_{\mathbf{h},\mathbf{y}} \mathbf{w}^T \Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y})$
    $S_i = S_i \cup \{\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}\}$
  end for
  $Confidence$ = compute confidence statistics
  S_i^{conf} = select from S_i using $Confidence$
  $\mathbf{w_i} \leftarrow Learn(\cup_i S_i^{conf})$
  $i = i + 1$
until S_i^{conf} has no new unique examples
$best = \arg\max_{\mathbf{i}}(\sum_{s \in S_i} Confidence(s))/|S|$
return $\mathbf{w_{best}}$

structure prediction (i.e., solving the inference problem) which requires taking statistics over complex and intricate patterns, identifying high quality predictions can be done using much simpler patterns that are significantly easier to capture.

In the reminder of this section we describe our learning approach. We begin by introducing the overall learning framework (Sec. 6.1), we then explain the rational behind confidence estimation over self-generated data and introduce the confidence measures used in our experiments (Sec. 6.2). We conclude with a description of the specific learning algorithms used for updating the model (Sec. 6.3).

6.1 Unsupervised Confidence-Driven Learning

Our learning framework works in an EM-like manner, iterating between two stages: making predictions based on its current set of parameters and then retraining the model using a subset of the predictions, assigned high confidence. The learning process "discovers" new high confidence training examples to add to its training set over multiple iterations, and converges when the model no longer adds new training examples.

While this is a natural convergence criterion, it provides no performance guarantees, and in practice it is very likely that the quality of the model (i.e., its performance) fluctuates during the learning process. We follow the observation that confidence estimation can be used to approximate the performance of the entire model and return the model with the highest overall prediction confidence.

We describe this algorithmic framework in detail in Alg. 7. Our algorithm takes as input a set of natural

language sentences and a set of parameters used for making the initial predictions [1]. The algorithm then iterates between the two stages - predicting the output structure for each sentence (line 4), and updating the set of parameters (line 9). The specific learning algorithms used are discussed in Sec. 6.3. The training examples required for learning are obtained by selecting high confidence examples - the algorithm first takes statistics over the current predicted set of output structures (line 7), and then based on these statistics computes a confidence score for each structure, selecting the top ranked ones as positive training examples, and if needed, the bottom ones as negative examples (line 8). The set of top confidence examples (for either correct or incorrect prediction), at iteration i of the algorithm, is denoted S_i^{conf}. The exact nature of the confidence computation is discussed in Sec. 6.2.

The algorithm iterates between these two stages, at each iteration it adds more self-annotated examples to its training set, learning therefore converges when no new examples are added (line 11). The algorithm keeps track of the models it trained at each stage throughout this process, and returns the one with the highest averaged overall confidence score (lines 12-13). At each stage, the overall confidence score is computed by averaging over all the confidence scores of the predictions made at that stage.

6.2 Unsupervised Confidence Estimation

Confidence estimation is calculated over a batch of input ($\mathbf{x}$) - output ($\mathbf{y}$) pairs. Each pair decomposes into smaller first-order and second-order decisions (defined Sec. 4.2). Confidence estimation is done by computing the statistics of these decisions, over the entire set of predicted structures. In the rest of this section we introduce the confidence measures used by our system.

Translation Model The first approach essentially constructs a simplified translation model, capturing word-to-predicate mapping patterns. This can be considered as an abstraction of the prediction model: we collapse the intricate feature representation into high level decisions and take statistics over these decisions. Since it takes statistics over considerably less variables than the actual prediction model, we expect this model to make reliable confidence predictions. We consider two variations of this approach, the first constructs a unigram model over the first-order decisions and the second a bigram model over the second-order decisions. Formally, given a set of predicted structures we define the following confidence scores:

[1] Since we commit to the max-score output prediction, rather than summing over all possibilities, we require a reasonable initialization point. We initialized the weight vector using simple, straight-forward heuristic, similar to the one used in response-based learning

Unigram Score:

$$p(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{|z|} p(s_i|y(s_i))$$

Bigram Score:

$$p(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{|z|} p(s_{i-1}(s_i)|y(s_{i-1}), y(s_i))$$

Structural Proportion Unlike the first approach which decomposes the predicted structure into individual decisions, this approach approximates the model's performance by observing global properties of the structure. We take statistics over the proportion between the number of predicates in $\mathbf{z}$ and the number of words in $\mathbf{x}$.

Given a set of structure predictions S, we compute this proportion for each structure (denoted as $Prop(\mathbf{x}, \mathbf{z})$) and calculate the average proportion over the entire set (denoted as $AvProp(S)$). The confidence score assigned to a given structure $(\mathbf{x}, \mathbf{y})$ is simply the difference between its proportion and the averaged proportion, or formally

$$PropScore(S, (\mathbf{x}, \mathbf{z})) = AvProp(S) - Prop(x, z)$$

This measure captures the global complexity of the predicted structure and penalizes structures which are too complex (high negative values) or too simplistic (high positive values).

Combined The two approaches defined above capture different views of the data, a natural question is then - *can these two measures be combined to provide a more powerful estimation?* We suggest a third approach which combines the first two approaches. It first uses the score produced by the latter approach to filter out unlikely candidates, and then ranks the remaining ones with the former approach and selects those with the highest rank.

6.3 Learning Algorithms

Given a set of self generated structures, the parameter vector can be updated (line 9 in Alg. 7). We consider two learning algorithm for this purpose.

The first is a **binary learning** algorithm, which considers learning as a classification problem, that is finding a set of weights $\mathbf{w}$ that can best separate correct from incorrect structures. The algorithm decomposes each predicted formula and its corresponding input sentence into a feature vector $\Phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$ normalized by the size of the input sentence $|\mathbf{x}|$, and assigns a binary label to this vector[2]. The learning process

[2]Without normalization longer sentences would have more influence on binary learning problem. Normalization is therefore

Model	Description
INITIAL MODEL	Manually set weights (Sec. 6.4)
PRED. SCORE	normalized prediction (Sec. 6.4)
ALL EXAMPLES	All top structures (Sec. 6.4)
UNIGRAM	Unigram score (Sec. 6.2)
BIGRAM	Bigram score (Sec. 6.2)
PROPORTION	Words-predicate prop (Sec. 6.2)
COMBINED	Combined estimators (Sec. 6.2)
RESPONSE BASED	Supervised (binary) (Sec. 6.4)
SUPERVISED	Fully Supervised (Sec. 6.4)

Table 6.1: Compared systems and naming conventions.

is defined over both positive and negative training examples. To accommodate that we modify line 8 in Alg. 7, and use the confidence score to select the top ranking examples as positive examples, and the bottom ranking examples as negative examples. We use a linear kernel SVM with squared-hinge loss as the underlying learning algorithm.

The second is a **structured learning** algorithm which considers learning as a ranking problem, i.e., finding a set of weights $\mathbf{w}$ such that the "gold structure" will be ranked on top, preferably by a large margin to allow generalization.The structured learning algorithm can directly use the top ranking predictions of the model (line 8 in Alg. 7) as training data. In this case the underlying algorithm is a structural SVM with squared-hinge loss, using hamming distance as the distance function. We use the cutting-plane method to efficiently optimize the learning process' objective function.

6.4 Experimental Settings

In this section we describe our experimental evaluation. We compare several confidence measures and analyze their properties. Table 6.1 defines the naming conventions used throughout this section to refer to the different models we evaluated. We begin by describing our experimental setup and then proceed to describe the experiments and their results. For the sake of clarity we focus on the best performing models (COMBINED using BIGRAM and PROPORTION) first and discuss other models later in the section.

In all our experiments we used the Geoquery dataset (Zelle and Mooney [1996]), consisting of U.S. geography NL questions and their corresponding Prolog logical MR. We used the data split described in (Clarke et al. [2010]), consisting of 250 queries for evaluation purposes. We compared our system to several supervised models, which were trained using a disjoint set of queries. Our learning system had access only to the NL questions, and the logical forms were only used to evaluate the system's performance. We report

required to ensure that each sentence contributes equally to the binary learning problem regardless of its length.

the proportion of correct structures (accuracy). Note that this evaluation corresponds to the $0/1$ loss over the predicted structures.

Initialization Our learning framework requires an initial weight vector as input. We use a straight forward heuristic and provide uniform positive weights to three features. This approach is similar in spirit to previous works (Clarke et al. [2010]; Zettlemoyer and Collins [2007]). We refer to this system as INITIAL MODEL throughout this section.

Competing Systems We compared our system to several other systems:

(1) **PRED. SCORE**: An unsupervised framework using the model's internal prediction score ($\mathbf{w}^T\Phi(\mathbf{x},\mathbf{h},\mathbf{y})$) for confidence estimation.

(2) **ALL EXAMPLES**: Treating all predicted structures as correct, i.e., at each iteration the model is trained over all the predictions it made. The reported score was obtained by selecting the model at the training iteration with the highest overall confidence score (see line 12 in Alg. 7).

(3) **RESPONSE BASED**: A natural upper bound to our framework is the approach used in (Clarke et al. [2010]). While our approach is based on assessing the correctness os the model's predictions according to unsupervised confidence estimation, their framework is provided with external supervision for these decisions, indicating if the predicted structures are correct.

(4) **SUPERVISED**: A fully supervised framework trained over 250 $(\mathbf{x},\mathbf{y})$ pairs using structured SVM.

6.4.1 Results

Our experiments aim to clarify three key points:

(1) Can a semantic parser indeed be trained without any form of external supervision? this is our key question, as this is the first attempt to approach this task with an unsupervised learning protocol.[3] In order to answer it, we report the overall performance of our system in table 6.2.

The manually constructed model INITIALMODEL achieves a performance of 0.22. We can expect learning to improve on this baseline. We compare three self-trained systems, ALL EXAMPLES, PREDICTIONSCORE and COMBINED, which differ in their sample selection strategy, but all use confidence estimation for selecting the final semantic parsing model. The ALL EXAMPLES approach achieves an accuracy score of 0.656. PREDICTIONSCORE only achieves a performance of 0.164 using the binary learning algorithm and 0.348 using the structured learning algorithm. Finally, our confidence-driven technique COMBINED

[3]While unsupervised learning for various semantic tasks has been widely discussed, this is the first attempt to tackle this task. We refer the reader to Sec. 6.5 for further discussion of this point.

achieved a score of 0.536 for the binary case and 0.664 for the structured case, the best performing models in both cases. As expected, the supervised systems RESPONSE BASED and SUPERVISED achieve the best performance.

These results show that training the model with training examples selected carefully will improve learning - as the best performance is achieved with perfect knowledge of the predictions correctness (RESPONSE BASED). Interestingly the difference between the structured version of our system and that of RESPONSE BASED is only 0.07, suggesting that we can recover the binary feedback signal with high precision. The low performance of the PREDICTIONSCORE model is also not surprising, and it demonstrates one of the key principles in confidence estimation - the score should be comparable across predictions done over different inputs, and not the same input, as done in PREDICTIONSCORE model.

(2) How does confidence driven sample selection contribute to the learning process? Comparing the systems driven by confidence sample-selection to the ALL EXAMPLES approach uncovers an interesting tradeoff between training with more (noisy) data and selectively training the system with higher quality examples. We argue that carefully selecting high quality training examples will result in better performance. The empirical results indeed support our argument, as the best performing model (RESPONSE BASED) is achieved by sample selection with perfect knowledge of prediction correctness. The confidence-based sample selection system (COMBINED) is the best performing system out of all the self-trained systems. Nonetheless, the ALL EXAMPLES strategy performs well when compared to COMBINED, justifying a closer look at that aspect of our system.

We argue that different confidence measures capture different properties of the data, and hypothesize that combining their scores will improve the resulting model. In table6.3 we compare the results of the COMBINED measure to the results of its individual components - PROPORTION and BIGRAM. We compare these results both when using the binary and structured learning algorithms. Results show that using the COMBINED measure leads to an improved performance, better than any of the individual measures, suggesting that it can effectively exploit the properties of each confidence measure. Furthermore, COMBINED is the only sample selection strategy that outperforms ALL EXAMPLES.

(3) Can confidence measures serve as a good proxy for the model's performance? In the unsupervised settings we study the learning process may not converge to an optimal model. We argue that by selecting the model that maximizes the averaged confidence score, a better model can be found. We validate this claim empirically in table 6.4. We compare the performance of the model selected using the confidence score to the performance of the final model considered by the learning algorithm (see Sec. 6.1 for details).

Algorithm	Supervision	Acc.
INITIAL MODEL	—	0.222
SELF-TRAIN: (*Structured*)		
PRED. SCORE	—	0.348
ALL EXAMPLES	—	0.656
COMBINED	—	**0.664**
SELF-TRAIN: (*Binary*)		
PRED. SCORE	—	0.164
COMBINED	—	0.536
RESPONSE BASED		
BINARY	250 (binary)	0.692
STRUCTURED	250 (binary)	0.732
SUPERVISED		
STRUCTURED	250 (struct.)	0.804

Table 6.2: Comparing our *Self-trained* systems with Response-based and supervised models. Results show that our COMBINED approach outperforms all other unsupervised models.

Algorithm	Accuracy
SELF-TRAIN: (*Structured*)	
PROPORTION	0.6
BIGRAM	0.644
COMBINED	0.664
SELF-TRAIN: (*Binary*)	
BIGRAM	0.532
PROPORTION	0.504
COMBINED	0.536

Table 6.3: Comparing COMBINED to its components BIGRAM and PROPORTION. COMBINED results in a better score than any of its components, suggesting that it can exploit the properties of each measure effectively.

We also compare it to the best model achieved in any of the learning iterations.

Since these experiments required running the learning algorithm many times, we focused on the binary learning algorithm as it converges considerably faster. In order to focus the evaluation on the effects of learning, we ignore the initial model generated manually (INITIAL MODEL) in these experiments. In order to compare models performance across the different iterations fairly, a uniform scale, such as UNIGRAM and BIGRAM, is required. In the case of the COMBINED measure we used the BIGRAM measure for performance estimation, since it is one of its underlying components. In the PRED. SCORE and PROPORTION models we used both their confidence prediction, and the simple UNIGRAM confidence score to evaluate *model* performance (the latter appear in parentheses in table 6.4).

Results show that the over overall confidence score serves as a reliable proxy for the model performance - using UNIGRAM and BIGRAM the framework can select the best performing model, far better than the performance of the default model to which the system converged.

Algorithm	Best	Conf. estim.	Default
PRED. SCORE	0.164	0.128 (0.164)	0.134
UNIGRAM	0.52	0.52	0.4
BIGRAM	0.532	0.532	0.472
PROPORTION	0.504	0.27 (0.504)	0.44
COMBINED	0.536	0.536	0.328

Table 6.4: Using confidence to approximate model performance. We compare the best result obtained in any of the learning algorithm iterations (Best), the result obtained by approximating the best result using the averaged prediction confidence (Conf. estim.) and the result of using the default convergence criterion (Default). Results in parentheses are the result of using the UNIGRAM confidence to approximate the model's performance.

6.5 Related Work

The difficulty of providing the required supervision motivated learning approaches using weaker forms of supervision. The works of (Chen and Mooney [2008]; Liang et al. [2009]; Branavan et al. [2009]; Titov and Kozhevnikov [2010]) ground NL in an external world state directly referenced by the text. The NL input in our setting is not restricted to such grounded settings and therefore we cannot exploit this form of supervision. Recent work (Clarke et al. [2010]; Liang et al. [2011]) suggest using response-based learning protocols, which alleviate some of the supervision effort. This work takes an additional step in this direction and suggest an unsupervised protocol.

Other approaches to unsupervised semantic analysis (Poon and Domingos [2009]; Titov and Klementiev [2011]) take a different approach to semantic representation, by clustering semantically equivalent dependency tree fragments, and identifying their predicate-argument structure. While these approaches have been applied successfully to semantic tasks such as question answering, they do not ground the input in a well defined output language, an essential component in our task.

Our unsupervised approach follows a self training protocol (Yarowsky [1995]; McClosky et al. [2006]; Reichart and Rappoport [2007a]) enhanced with constraints restricting the output space (Chang et al. [2007, 2009]). A Self training protocol uses its own predictions for training. We estimate the quality of the predictions and use only high confidence examples for training. This selection criterion provides an additional view, different than the one used by the prediction model. Multi-view learning is a well established idea, implemented in methods such as co-training (Blum and Mitchell [1998]).

Quality assessment of a learned model output was explored by many previous works (see Caruana and Niculescu-Mizil [2006] for a survey), and applied to several NL processing tasks such as syntactic parsing (Reichart and Rappoport [2007b]; Yates et al. [2006]), machine translation (Ueffing and Ney [2007]), speech (Koo et al. [2001]), relation extraction (Rosenfeld and Feldman [2007]), IE (Culotta and McCallum [2004]), QA (Chu-Carroll et al. [2003]) and dialog systems (Lin and Weng [2008]).

In addition to sample selection we use confidence estimation as a way to approximate the overall quality of the model and use it for model selection. This use of confidence estimation was explored in (Reichart et al. [2010]), to select between models trained with different random starting points. In this work we integrate this estimation deeper into the learning process, thus allowing our training procedure to return the best performing model.

6.6 Conclusions

We introduced an unsupervised learning algorithm for semantic parsing, the first for this task to the best of our knowledge. To compensate for the lack of training data we use a self-training protocol, driven by unsupervised confidence estimation. We demonstrate empirically that our approach results in a high preforming semantic parser and show that confidence estimation plays a vital role in this success, both by identifying good training examples as well as identifying good over all performance, used to improve the final model selection.

In this work we provided several intuitive confidence measures, and showed that combining them help capture different aspects of the correct distribution of the correct predictions. This could be extended to include more aspects which could potentially improve performance.

Chapter 7

Transferring Semantic Knowledge Across Domains

Learning from natural instructions is designed as a ubiquitous communication framework between an automated agent and a human instructor, allowing the human teacher to communicate his knowledge over a broad range of tasks and applications. This emphasizes the need for a semantic interpreter that can interpret instructions for more than a single domain. Since we tie the process of language learning with task learning (e.g., learning to interpret instructions of card game rules as a by-product of learning to play cards) this requirement can be phrased as learning an interpreter that can generalize to inputs beyond the specific task it was trained on. In the previous chapters discussing LNI for card games we test the ability of the agent to generalize along two lines – (1) New game scenarios, which tests whether the learned game-rule can generalize to new game states, and more relevant to this discussion, (2) New tasks, which tests whether learned interpreter can generalize to new tasks *defined over the same set of domain symbols*.

In this chapter we take a first step towards constructing a semantic interpreter that can leverage the model learned in one domain, and use it in a different domain. This is not a straight forward objective – the domain specific nature of semantic interpretation, as described in the current literature, does not allow for an easy move between domains. For example, an agent trained for the task of understanding card game descriptions will not be of any use when it will be given a sentence describing robotic soccer instructions. The obvious question therefore is whether the semantic interpretation process can be changed to accommodate an easier transition between domains.

In order to answer this question a closer look at the interpretation model is needed. When given a natural language sentence, the interpretation process breaks the sentence into a set of interdependent decisions which rely on an underlying representation mapping words to symbols, and syntactic patterns into compositional decisions. This representation takes into account domain specific information and is therefore of little use when moving to a different domain. In order to facilitate an easier transition between domains, domain-independent information has to be represented, such that when learned for one domain, it can be reused in another. We suggest to represent this information by adding a domain-independent intermediate layer capturing shallow simple semantic relations between the input sentence constituents.

The key requirement from such a representation is to capture import semantic information, but without relying on domain specific information. We suggest to use Semantic Roles Labels (SRL) as the hidden representation (section 7.2 below describes the SRL task in more details). This representation assigns labels carrying semantic information, that abstract over domain specific labels. Given an input sentence this representation consists of assigning labels mapping the sentence to a structure capturing the predicate and a list of typed arguments. The following example describes this representation -

Example 8. *[SRL based hidden representation].*

- ***Robocup Soccer Domain***

  "Pink11 kicks to Pink1"

  [Pink11]$_{ARG0}$ *[kicks]*$_{PRED}$ *to* *[Pink1]*$_{ARG1}$

  $\texttt{pass}(\texttt{pink11}, \texttt{pink1})$

- ***Card Game domain***

  "Any top card can be moved to an empty freecell

  Any [top card]$_{ARG0}$ *can be [moved]*$_{PRED}$ *to [an empty freecell]*$_{ARG1}$

  $\texttt{move}(\texttt{c}, \texttt{f}) \leftarrow \texttt{card}(\texttt{c}) \wedge \texttt{top}(\texttt{c}, \texttt{x}) \wedge \texttt{freecell}(\texttt{f}) \wedge \texttt{empty}(\texttt{f})$

Example 8 demonstrates the key idea behind our representation in an idealized settings – two sentences taken from two completely different domains, have a similar intermediate structure. In this case, the constituents of the first sentence, taken from the Robocup domain (Chen and Mooney [2008]), are assigned SRL labels – the word corresponding to a logical function is identified as a SRL predicate, and words associated with the logical formula arguments are labeled as SRL arguments.

The SRL task relies on linguistic information alone, without using domain specific information, for example, the predicate label assigned to the word *"kick"* indicates that this word is the predicate of the sentence, not a specific domain predicate. Since these decisions can rely on domain-independent information, when the same syntactic structures are repeated in a different domain the SRL labels can remain the same – as is the case in example 8. The second sentence taken from a card game domain, however it has the same SRL predicate-argument structure.

Our approach is guided by the observation that if after learning a model for the Robocup domain, the SRL structure can be predicted reliably, then when moving to a different domain, this information can be reused. In an ideal settings this information could be used directly – once the predicate argument structure is identified only domain specific information at the lexical level is needed to complete the prediction and generate the output formula.

In practice however, these idealized settings may not hold as the domain representation may not correspond in to the underlying SRL representation in the same way in the two domains. To clarify the problem consider the following example -

Example 9. *[Arguments mismatch across different domains].*

- ***Robocup Soccer Domain***

  "Pink11 kicks the ball to Pink1"

  [Pink11]$_{ARG0}$ *[kicks]*$_{PRED}$ *[the ball]*$_{ARG2}$ *to* *[Pink1]*$_{ARG1}$

  $\texttt{pass}(\texttt{pink11}, \texttt{pink1})$

- ***Robocup Soccer Domain VERBOSE***

  "Pink11 kicks the ball to Pink1"

  [Pink11]$_{ARG0}$ *[kicks]*$_{PRED}$ *[the ball]*$_{ARG2}$ *to* *[Pink1]*$_{ARG1}$

  $\texttt{pass}(\texttt{pink11}, \texttt{pink1}, \texttt{ball})$

In this case we consider two very close domains - Robocup soccer, and Robocup soccer VERBOSE. The second domain is almost identical to the first, except for one difference, the domain language is more explicit and aligns perfectly with the predicate argument structure described in the text. In this example, the SRL predicate argument structure of the sentence *"Pink11 kicks the ball to Pink1"*, aligns perfectly in the Robocup VERBOSE case, as both the SRL and the logical predicate argument structure contain a predicate and three arguments. The Robocup domain however, omits one of these arguments, since the logical predicate takes only two arguments.

It is clear that the simple strategy suggested above, in which the SRL predictions should be mapped directly will not work in this case, as the two domains differ. **Furthermore, since the intermediate representation is learned as a by product of the learning process for semantic interpretation, it is not clear that an SRL model learned for the first domain will be able to recognize the additional argument that does not correspond to the logical representation.**

The key question that we study in this chapter is whether we can learn such an intermediate layer, identifying shallow predicate-arguments relations in the input sentence, as a by-product of the semantic interpretation learning process, which will generalize to new domains, defined over a different set of logical symbols which may correspond differently to the text. The key requirement from such SRL predictor is to not directly mirror the logical predicate-argument structure in the training domain, but rather learn these relations more generally, to the extent it can assist in making prediction in a new domain.

Note that this phenomenon in which the intermediate representation is strongly coupled with the output prediction is not unique to our SRL model. Similarly, other semantic parsing models which learn a semantic grammar (e.g., Synchronous Grammar Wong and Mooney [2007], or Combinatory Categorical Grammar Zettlemoyer and Collins [2005]) learn an intermediate syntactic layer which does not capture the syntax of the sentence in isolation, but rather syntactic patterns relevant and helpful for the syntactic decision. This happens since the supervision driving grammar construction comes from logical forms, not annotated semantic-parse-trees.

In order to test whether our SRL based intermediate layer suffers from the same problem we introduced the Robocup VERBOSE domain, and test whether learning the SRL classifier on the Robocup domain, which provides supervision for just a subset of the SRL arguments will help identify the arguments for Robocup Verbose, treating success over the final prediction as an indication. In order to test this, when moving to the new domain (Robocup VERBOSE) all Robocup specific information was erased, only the parameters learned from the SRL predictor remained.

For example, the only information carried over between domains for the sentence *"Pink11 passes the ball to Pink2"*, would be a scoring function indicating which constituents are likely to be mapped to a SRL predicate, and arguments, but not to any domain-specific ones. In the new domain, the logical predicate will take three arguments. The contribution of the intermediate layer, if trained appropriately over the previous domain, would be to identify which of the natural language constituents are likely to serve as the arguments, according to their types. This information is acts as a filter over the set of possible logical predicate-argument candidates, giving preference to verbs and noun phrases and taking into account their position in the sentence.

In this chapter we suggest an initial model. We start by describing our working domain, Robocup (Chen and Mooney [2008]) and learning in this domain, which is based on grounded language acquisition. Learning language mimics a human language acquisition scenario – the learner is exposed to a stream of precepts (described symbolically), and a stream of sentences which describe these precepts. We introduce a learning algorithm for these settings in section 7.1. We then move to describe our new intermediate representation in section 7.2.

7.1 Learning Semantic Correspondence in Grounded Settings

In this section we consider the problem of grounded language acquisition. The settings to this problem bears resemblance to its human language acquisition counterpart, in which the language learner is exposed

Dataset	Number of comments	Number of events
Overall	1868	10657
2001 final	671 (520)	4003
2002 final	458 (376)	2223
2003 final	397 (320)	2113
2004 final	342 (323)	2318

Table 7.1: Properties of the Robocup domain taken from Chen and Mooney [2008]. The data consists of pairs of the form (sentence, <event,..,event>), some of these pairs are noisy – not all natural language comments are associated with the correct event. The numbers in parenthesis are the number of comments that are paired with a set of events which includes the correct event.

to a stream of linguistic inputs and perceptual information. Language acquisition is essentially the process of learning to align these two streams. We follow the set up suggested by (Chen and Mooney [2008]) and use the robocup sport casting data for this task. In this section we explain this work from a technical perspective and suggest a discriminative, perceptron-based, learning algorithm for these settings.

7.1.1 Language Learning from Ambiguous Perceptual Data

The challenge of the language learner is to construct a mapping between natural language sentences describing robotic soccer scenes and precepts capturing these scenes, expressed in formal logic. From a technical perspective, language learning in these settings resembles semantic parsing. However, unlike the previously discussed learning settings, there is no feedback function, but rather the learner has access to a noisy training data, consisting of entries of the form $(\mathbf{x}, < \mathbf{y}_1, .., \mathbf{y}_k >)$, where $\mathbf{x}$ is an input sentence and $\mathbf{y}_i$ is an event occurring roughly at the same time as $\mathbf{x}$ was uttered. This learning set up is described in figure 3.1(c).

Initialization We initialize the model by taking statistics over the aligned data and computing $\mathtt{p}(\mathtt{y}|\mathtt{x})$. Given the training set $\{(\mathbf{x}, < \mathbf{y}_1, .., \mathbf{y}_k >)\}$, this is done by aligning every word in $\mathbf{x}$ to every logical symbol appearing in every $\mathbf{y}$ that is aligned with it. The probability of a given matching (x,y) is simply computed as the product of probabilities $\prod_{i=1}^{n} \mathtt{p}(\mathtt{y_i}|\mathtt{x_i})$, where n is the number of entities appearing in the interpretation y, and x_i, y_i is the word level matching to a logical symbol.

Learning Algorithm We propose a perceptron-based learning algorithm that is adapted to these settings. Since the algorithm does not have access to annotated data, it has to generate its own training data. In this learning scenario, the learner has a short list of candidate interpretations for each input sentence. The learner picks the most likely output from that list, and the selected interpretation is used by the algorithm as a gold structure. From that point, the algorithm is simply a latent structure perceptron, performing

Algorithm 8 Grounded-Learning Perceptron

Require: Weakly aligned corpus $\mathcal{C}$= $\{\mathbf{x}^l \rightarrow \{\mathbf{y}_1, .., \mathbf{y}_k\}\}_{l=1}^{N}$,

```
Initialize w using C
repeat
  for l = 1, ..., N do
    ĥ, ŷ = arg max_{h,y∈C(x)} w^T Φ(x^l, h, y)
    h, y = arg max_{h,y} w^T Φ(x^l, h, y)
    if ŷ ≠ y then
      w = Φ(x, ĥ, ŷ) − Φ(x, h, y)
    end if
  end for
until Convergence
return w
```

mistake driven updates. Note that the algorithm runs the semantic inference procedure twice – the first is a constrained form which restricts the space of possible outputs to the list of possible candidates and the second is the regular semantic interpreter, not constrained to the short list of candidates. The training procedure updates the model to take the same decisions as the constrained model without explicitly using the constraints. Algorithm 8 describes the algorithm in more detail.

7.1.2 Evaluation

Given a natural language sentence and a set of several events, we evaluate the performance of this algorithm by predicting an event that best corresponds to the sentence, this is typically reffered to as a *matching task*. We perform 4-fold cross validation as described in Chen and Mooney [2008]. The results are summarized in Table 7.2, where we compare our system to several perviously published results.

The best performing system, Kim and Mooney [2010], uses a generative model to learn in these noisy settings. Their best performing system, appearing in parenthesis in table 7.2, combines two models – a probabilistic model for matching an event to a sentence, and a model for strategic generation, capturing the probability of comment generation given an event. Unlike our model that seeks to capture a deeper representation of the predicate argument structure their generative model focuses on the event type alone, rather than structural information. This approach is shown to be very effective, especially when combined with additional information, the strategic generation model, which helps eliminate candidate events that are not likely to be described at all.

The focus of our system is different; unlike Kim and Mooney [2010] which aim to optimize their model for the event matching task, our model is constructed to capture structural properties, likely to repeat in other domains. The model performs reasonably well, moving from a matching accuracy score of 0.69 offered by the simple probabilistic initialization described in section 7.1.1, to an accuracy score of 0.77 using

System	Score
INIT	0.69
LNI	0.77
CHEN AND MOONEY [2008]	0.67
LIANG ET AL. [2009]	0.757
CHEN ET AL. [2010]	0.735 (0.793)
KIM AND MOONEY [2010]	0.832 (**0.885**)

Table 7.2: Results for the matching task over the Robocup data in Chen and Mooney [2008]. Several systems are evaluated: INIT is the initial model without learning, LNI is our system, CHEN AND MOONEY [2008], LIANG ET AL. [2009] CHEN ET AL. [2010] and KIM AND MOONEY [2010] are previously published systems. The number in parenthesis indicates a score obtained by combining several models. In CHEN ET AL. [2010] the score was obtained by using the matching results of Liang et al. [2009] for initialization and in KIM AND MOONEY [2010] this score was obtained by combining a model for strategic generation. See section 7.1.2 for further discussion.

our learning algorithm. Although our model is not the state-of-the-art, it is able to outperform several published systems as shown in table 7.2. Since our focus is to show an improvement across domains by learning an intermediate layer abstracting over domain information, rather than overall performance, we leave improvements to this algorithm as future work. The main claim supported by these experiments is that the system is able to learn in these noisy settings. This model will serve as a baseline for our experiments, in section 7.3, where we shall test whether learning using a rich intermediate layer contributes to learning, and whether the learned model can assist the semantic prediction when moving to a new domain.

7.2 Semantic Roles as Intermediate Representation

Semantic Roles for Situated Language The task of Semantic Role Labeling (Gildea and Jurafsky [2002]; Carreras and Màrquez [2004]) can be described as answering the question *"who did what to whom"* for a given sentence. Answering this question amounts to making several interdependent decisions - identifying the sentence predicate and its arguments, and classifying the arguments according to their roles. In our settings we are concerned with abstracting over domain specific information and reduce the definition of the SRL task accordingly. Our system classifies the predicate argument relations, where arguments can take up to three labels (A0, A1, A2).

In the original definition of the PropBank and CoNLL shared task (Carreras and Màrquez [2004]), there are six different types of arguments (A0-A5), each label having different semantic based on the specific predicate they are attached to, as defined in the PropBank.

In this work we take a different approach and use the semantics dictated by the domain's ontology to determine the meaning of the different argument labels. Given a domain function symbol, we assume

that each argument corresponds to a part of an external world scene, and its thematic role is determined according to that. For example consider the pair appearing in example 8 - *"Pink11 kicks to Pink1"* and $\texttt{pass}(\texttt{pink11}, \texttt{pink1})$, we assume that the function $\texttt{pass}(\cdot, \cdot)$ takes two arguments, the first describing the player initiating the action, and the second describing the recipient. The semantics of the thematic roles assigned to arguments corresponds to this meaning, and should be consistent along all argument labels for that predicate type. To clarify that point consider the following example - *"Pink1 received the ball from Pink11"*, in this case although the sentence has a different syntactic structure, the argument labels should stay the same, capturing the fact that Pink11 was the initiator of the action (i.e., labeled as A0), and that Pink1 was the recipient (i.e., labeled as A1).

Semantic Role Classifier The SRL classifier takes as input a sentence, its part of speech tags and a shallow parse representation (chunks) of the sentence. The classification process assigns a single label to each chunk - a predicate, argument type or none.

Semantic Roles Features In order to determine the SRL label of a natural language constituent we use the following set of features:

- Part of speech tag (POS(W,L)): the part of speech for a constituent W for a label prediction L.
- Noun ordering (NOUNORDER(S,W,L)): The occurrence (ordinal) number of a noun W in a sentence S, for an argument label L (e.g., first noun out of two nouns).
- Verb position (VERBPOSITION(S,W,L)): The position of a constituent W in a sentence S, relative to the verb (word labeled as a predicate), for an argument label L (e.g., before/after the verb).
- Lemmas of words appearing in the chunk (LEMMA(W,L)): The lemma of words appearing in a constituent W, for a label L.
- First word of the chunk (FIRSTWORD(W,L)): The first word in a constituent W, for a label L (e.g., for "The Pink goalie" FIRSTWORD(W,L) will generate a feature for the word "The").
- Words appearing before and after the chunk (WINDOW(S,W,L)): a window of size one, around the constituent W in sentence S, for a label L.
- Prepositions appearing before the chunk (PRECEDINGPREPOSITION(S,W,L)): activated if there is a preposition preceding constituent W in sentence S, for a label L.(e.g., for a candidate argument label for the word "Pink11", in "... to Pink11", PRECEDINGPREPOSITION(S,W,L) will generate a feature for the preposition "to").

In several works studying the SRL task (e.g., Punyakanok et al. [2008]) features derived from more involved syntactic information are also used. However since we learn the SRL model as a by-product of the learning process, without direct supervision we use a relatively small set of features. This approach has been shown to be effective in recent works which a SRL predictor from ambiguous supervision (Connor et al. [2011]).

Integrating SRL decisions into the model The new semantic inference decision is extended to include this information:

$$\begin{aligned} F_{\mathbf{w}}(\mathbf{x}) = \arg\max_{\alpha,\beta} & \sum_{c\in\mathbf{x}}\sum_{s\in D} \alpha_{cs}\cdot {\mathbf{w_1}}^T\Phi_1(\mathbf{x},c,s) \\ & + \sum_{c,d\in\mathbf{x}}\sum_{s,t\in D}\sum_{i,j} \beta_{cs^i,dt^j}\cdot {\mathbf{w_2}}^T\Phi_2(\mathbf{x},c,s^i,d,t^j) \\ + \sum_{c\in\mathbf{x}} \gamma_c\cdot {\mathbf{w_3}}^T\Phi_3(\mathbf{x},c) + & \sum_{c,d\in\mathbf{x},i\in\{0,1,2\}} \delta^i_{cd}\cdot {\mathbf{w_4}}^T\Phi_4(\mathbf{x},c,d,i) \end{aligned} \tag{7.1}$$

The new variables γ, δ are the new decision variables types capturing the SRL decision. The variable γ is an indicator variable for predicate classification, and δ is an indicator variable for argument classification.

To ensure the consistency of classification we add the following constraints:

- Predicate decision and Argument decision consistency:

  $\forall i, c, d \in \mathbf{x}\ (\gamma_c \rightarrow \neg\delta^i_{c,d})$

  $\forall i, c, d \in \mathbf{x}\ (\delta^i_{c,d} \rightarrow \gamma_c)$

- A sentence must have a predicate

  $\sum_{c\in\mathbf{x}} \gamma_c$=1

- Each label can only appear once

  $\forall c, d \in \mathbf{x}\ (\sum_i \delta^i_{c,d} \leq 1)$

In addition to these variables we add new constraints tying these new variables to semantic interpretation. In the ideal setting described above, we can have the two decision types be strongly coupled, that is, a domain symbol is active if and only if an SRL label is active over the same word span. Unfortunately, since these idealized settings do not hold we use a relaxed version of this constraint to bind the two models together – if a domain symbol is identified in the logical interpretation, there has to be an SRL label active over the same word span. In addition, we require that logical constants and unary relations be mapped to arguments. **It should be noted that this constraint essentially drives learning the SRL intermediate layer in our settings, as the supervision for the logical layer forces an SRL layer decision.**

- $\forall c, d \in x \ (\beta(c, d) \rightarrow \delta(c, d))$

The Robocup soccer VERBOSE domain In order to evaluate the SRL knowledge transfer settings, we create a new domain *Robocup soccer Verbose*. This new domain was artificially created to test the ability of the SRL intermediate layer to adapt to new domains in which the SRL predicate argument structure align differently to the logical predicate argument structure. While in the original Robocup domain SRL arguments were frequently left unmatched with a logical entity, in the *Robocup soccer Verbose* domain we made sure that the two align perfectly. The two domains are described in Appendix B.

We obtain this domain by making a slight change to the Robocup domain, by automatically replacing several predicates with their VERBOSE counter-part:

- **Unary Predicates** In the original Robocup domain several game events that involve several entities were described using an unary predicate, implicitly conveying some information, or ignoring it all together. However, this information was usually expressed in the text describing the game. We replaced these predicates with new predicates in the VERBOSE domain, such that the new predicates will explicitly take all these argument. For example, these include `stop(ball)` in addition to `ballstopped`, `playmode(offside_l, Pink)` instead of `playmode(offside_l)`.
- **Binary Predicates** Binary predicates, describe events involving several participants. In some cases we used several variations of the same predicate, with different number of arguments, this include predicates such as `pass(player1, player2)` that was extended to `pass(player1, player2, ball)` and `steal(player1)`, that had two variants -`steal(player1, player2, ball)` and `steal(player1, ball)`.

7.3 Evaluation

In our evaluation we aim to answer two key question: first, how does the new representation impact learning? The lexical model originally used was designed to construct a simple model in which it is easier to learn in the noisy supervision setting we work in. The basic question is therefore *is this a good representation for learning in the settings?*

In order to answer this question we evaluate and compare the performance of a system trained with a lexical representation to a system trained with the SRL intermediate representation.

Second, can this representation facilitate learning when moving to a new domain? As described in Example 9 the mapping between SRL labels and domain specific labels may differ between domains; in some cases not all SRL arguments will be mapped to a domain symbol. Since the learning the SRL model is

completely driven by feedback obtained by the domain prediction, it is very likely to shape the SRL labels prediction accordingly.

The question therefore is *will the intermediate layer learned in this process be useful for a new domain?*

We compare the performance of several system, The systems use the training procedure described in section 7.1) for the Robocup domain, and the initialization procedure for the Robocup VERBOSE domain.

- R(L): A system trained over Robocup data using a Lexical model. This model is similar to the model used for Geoquery, described in chapter 4.
- R(S) : A system trained over Robocup data using SRL as an intermediate model.
- RV(S_R): A system making predictions over the Robocup VERBOSE domain, using an SRL intermediate layer learned over the Robocup domain.
- RV(L): A system making predictions over the Robocup VERBOSE domain, using a lexical intermediate layer.

The results are summarized in Table 7.3. These results show that using SRL as a hidden representation contributes significantly compared to the lexical representation. This shows that representing shallow semantic information at the sentence level, and integrating this information into the model can help the semantic interpretation decision. Our system now show competitive results to the system presented in (Kim and Mooney [2010]), which uses a much simpler representation. In their work the results of this system are significantly improved when adding additional strategic-generation information, which can be considered as a probabilistic filter for event types, we consider incorporating such information into our model as an interesting direction for future work.

To answer the second question we design the following experiment: we train a system over Robocup which uses the SRL intermediate layer (the R(S) system), and use it to make prediction in a new domain (the RV(S_R) system). In this case the intermediate SRL layer learned when training the R(S) system over the Robocup domain can be separated from the domain specific model. Whem moving to the new domain, Robocup VERBOSE, the parameters learned for the SRL model are saved, while the rest of the information is erased. In order to observe the effect of the learned layer directly we use the system initialized from data without learning, using the SRL predicate-argument information learned over Robocup, to predict these relations for the Robocup VERBOSE domain.

The second result shows that the SRL representation learned over the first domain can assist making prediction in the second domain. This demonstrates that the SRL model learned over very noisy supervision can generalize well. This is not trivial as the supervision is derived from the structure of the logical

System	Score
R(L)	0.77
R(S)	0.82
RV(L)	0.71
RV(S_R)	0.76

Table 7.3: Evaluating SRL as a hidden representation and its use to facilitate knowledge transfer between domains. We compare several systems: Robocup using a lexical representation (R(L)), Robocup using SRL representation (R(S)), Robocup Verbose using SRL trained on Robocup (RV(S_R)) and Robocup Verbose which uses a lexical representation. The parameters of both latter systems are obtained using probabilistic initialization from data.

domain, which differs in the two domains. The key difference is the number of arguments used in each representation. During training the SRL model receives feedback only for a several of the linguistic predicates that appear in the input, the SRL system is able to generalize from the feedback to other argument decisions by identifying the shallow syntactic patterns (such as location compared to the verb, part of speech and proximity to a preposition). As a result the SRL model learned over Robocup improves performance when making predictions over Robocup VERBOSE.

7.4 Conclusions

In this chapter we took a first step towards a new kind of generalization in semantic parsing: constructing a model that is able to generalize to new domain, defined over a different set of predicates.

The proposed solution is to add an abstract layer to the prediction process, one that could be shared by different domains. Although this adds to the complexity of the model, and therefore potentially makes learning more difficult, this representation helps learning. This could be attributed to the fact that the proposed intermediate layer is highly constrained and heavily biased by the semantic parsing decisions.

We show that the learned system produces an SRL predictor that can be used for other domains. The improvement in results when using the SRL intermediate layer to make predictions in a new domain can be attributed to intermediate layer's ability to correctly identify the predicate argument structure. Recall that only natural language constituents marked with SRL labels can be mapped to a logical symbol. The SRL layer serves as a filter for candidates for the logical layer, thus helping to reduce ambiguity in the noisy training data. The improvement achieved when moving to the new domain, Robocup VERBOSE, is therefore an indication of the learned SRL predictor's ability to capture the predict argument structure correctly in the new domain, although several argument types were not observed in the original domain (Robocup) over which the SRL predictor was trained.

The key issue discussed in this section is learning an intermediate representation driven by the logical

output supervision, which may differ when moving between domains. We show that our model offers some robustness, it is able to improve the semantic prediction in the new domain. This study however was done is an idealized setting: the natural language sentences were identical in the two domains. This introduces a different aspect of the difference between domains, the structure of the natural language input. We hypothesize that our model will be robust when moving to a new domain which uses sentences with similar syntactic structures. We leave this problem for future study. Note that in the LNI settings, in which issuing commands or giving instructions are the common cases, moving between domains is likely to use similar syntactic structures.

The proposed approach shares some similarities with *domain adaptation*. Domain adaptation is concerned with finding a set of parameters that will generalize better to different (yet related) tasks, typically by biasing the learning process to take into account the new domain. In this work we focus on representation adaptation, rather than domain adaptation. The connections between the two tasks could still be exploited, for example, by treating the SRL learning task as a separate learning task share between multiple semantic interpretation domains. We leave for future work the study of the connections between the two frameworks.

Chapter 8

Summary and Future Work

This work could be considered a feasibility study of LNI, taking into account both the challenges emerging from such a learning framework, and recent advances in semantic parsing and the application of machine learning techniques to natural language processing problems.

We identified semantic interpretation as the key bottleneck in LNI. This is hardly surprising – semantic interpretation is one of the major problems in all fields involving the study of human languages. In this work we take a much more modest goal than solving the problem of semantic interpretation and limit our research questions to the problem of semantic interpretation in LNI: knowledge communication via natural language, in a restricted, task-oriented domain. Our work studies interpretation through this perspective, and focuses on methods that help facilitate the construction of semantic interpreters and minimize human involvement in this process.

We frame the problem of semantic interpretation as a structured prediction problem, a well studied learning framework, used in many NLP learning domains. Under this definition, our intention to minimize human involvement in the semantic interpretation process becomes concrete – we suggest several machine learning approaches that require considerably less annotation effort, and a semantic interpretation model relying on lexical and shallow semantic knowledge that helps the learner to share information when moving between different semantic interpretation domains. Since current models treat learning a new domain as a completely separate learning problem, requiring the learning process to start from scratch, this model helps minimize the effort of moving between domains.

From a machine learning perspective our work dealing with minimizing the supervision effort falls under the body of work dealing with weakly supervised structured learning. We discuss several learning scenarios.

Response Driven Learning A novel learning framework in which learning is driven by a supervision signal obtained from an external environment. The signal does not provide direct feedback for the semantic interpretation decisions, but rather an overall judgment for the entire semantic prediction. For example,

in the case of database queries, interpreting the natural language question correctly results in the correct answer to the query. Identifying this fact is easy – it only requires a human annotator to look up the answers to a question, a considerably easier task than annotating question with logical forms corresponding to database queries. The remaining question is how to use this type of feedback?

The ease of obtaining this type of supervision signal comes at the cost of the information contained in it. From a machine learning perspective, this can be thought of as a structured learning using a binary feedback instead of the annotated structures required.

We suggest a light-weight interpretation model that is less dependent on extracting information from annotated structures and suggest several algorithms to learn in these settings. In addition, we show how to make up for missing data using external linguistic knowledge resources. Interestingly, we show empirically that using this type of annotation results in a model performing competitively to a fully supervised model.

Unsupervised Learning for Semantic Interpretation We also suggest an unsupervised approach for semantic parsing. Our approach uses a self-training training protocol, selectively using the model's self-predictions for retraining the model. Our training procedure is driven by unsupervised confidence estimation, a measure for assigning scores to the model's predictions. We estimate a confidence score for each prediction and use only the top ranking one to train the model.

The confidence estimation is also done in an unsupervised way, by taking statistics over patterns appearing in the current model predictions. We focus on simple, repeating patterns that can provide reliable estimation of correctness. Note that these patterns are different than the ones used for making the actual prediction, thus providing an additional view of the data, which is not captured by the model making the prediction.

We are able to show the robustness of our confidence estimation in several ways: first, we use it for selecting self-annotated training examples when training the model. We show that using this approach we are able to improve the performance of a self training protocol. Second, we use it to estimate the overall quality of the model: in this case we use the averaged confidence of all the predictions made by the model, and show that a higher score correlates with better performance.

Sharing Information Across Multiple Domains Finally, we follow the intuition that although semantic interpretation is defined over a set of symbols specific to a single domain, it should have domain independent characteristics. These characteristics could be exploited and reused when moving between different domains, each defined over a different set of symbols used as an output language.

In order to pursue this direction, we focus our attention to the question of semantic representation, or

more precisely, how to find a representation that can be adapted to a different domain, other than the one it was trained over.

We follow the intuition that certain aspect of this process are *domain-independent*, and suggest a learning representation that is based on recovering abstract predicate-argument relations in the input sentence. In an idealized setting, the only step required on top of that would be grounding the predicate-argument structure in the domain's set of symbols. We suggest a learning algorithm to learn this abstract structure jointly with the semantic interpretation task, treating it as an extension of the intermediate layer.

There are good reasons for learning this representation as an intermediate layer – it saves the effort of annotating data for learning to make predictions at the intermediate level, and also conceptually, this prediction should not be treated as an independent task, but rather one that supports the final prediction. However, learning it as an intermediate layer, driven by the learning process for a specific domain, biases the intermediate layer to support the final output decisions for that specific domain. This could hinder our efforts to learn an abstract domain-independent layer, since the domain-dependent bias introduced during training might prevent it from generalizing, and supporting decisions in new domains. We show that although the model is sensitive to the specific output domain used it is still reusable by other domains.

8.1 Limitations and Future Work

The key research motivation guiding this work is LNI - a framework for communicating knowledge to computer systems using natural language. We identified semantic interpretation as the bottleneck for this task and discussed machine learning solutions for this problem. We limited our examination of LNI to these settings and showed that it is possible to communicate rules expressed symbolically, with varying degree of success based on their difficulty. Many of the assumptions we took in these settings could be relaxed, resulting in further research directions.

This section reviews several ideas for extending work described in this thesis work. In some sense it serves a dual purpose, discussing both relevant ways to extend this work but also describing the boundaries of this work. This discussion will be done according to the different aspects of LNI, we shall start by explaining the limitations, or boundaries of the current work and suggest possible future directions that could help in pushing those boundaries further.

8.1.1 Scope of Natural Language Interpretation

In this work we followed the body of work on semantic parsing, which assumes that there exists a mapping between natural language constituents and logical symbols, and that the meaning of phrases can be recovered by composing their meaning. This assumption is true for many domains, where the natural language and domain representation align well, such as database systems an game playing scenarios. When moving to more complicated domains, these assumptions may no longer hold.

In order to get a better understanding of the gap between the scope of semantic interpretation discussed in this work and the general settings of natural language semantic interpretation, consider the following examples–

Example 10. ***Open Domain vs. Close Domain***

- *Open Domain Example (Wikipedia excerpt)*

  - NL Input:

  "Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review."

  - Logical Output:

  `born(obama, Honolulu) graduate(obama, columbia) graduate(obama, harvard_law)`

  `president(obama, harvard_law_review)`

- *Closed Domain Example (Geographic Query)*

  - NL Input:

  "How many states does the Colorado river run through?"

  - Logical Output:

  `count(state(traverse(river(const(colorado)))))`

The first example is a Wikipedia excerpt discussing Barak Obama, paired with a meaning representation that captures the facts expressed in the text. The second is taken from the Geoquery domain, consisting of natural language questions about geographical facts, paired with a database query. At a first glance it might not be clear why interpreting sentences of the first kind is more difficult than the second, to a human reader both sentences are immediately clear. Moreover, longer sentences of the second type, expressing a precise information need, could even be harder to interpret, and might require a human reader to backtrack several times until the information requirement expressed in the sentence would be completely clear. As a striking example of that kind of sentences consider the question – *"what is the most populous state out of the states that border the state with the longest river?"* Nonetheless, most current semantic parsers would be able

to handle such questions quite well, while relatively simple sentences such as the one in the above example still present a serious challenge.

There are several reasons why the first sentence is more difficult. The most intuitive one is the set of output symbols used. In the Geoquery domain these symbols are used to define a relational database, which typically consists of a relatively small set entities and relations. In the open domain example, the origin of the output language is not clear; the database of reference could be a list of Wikipedia topics, or even a more comprehensive knowledge base.

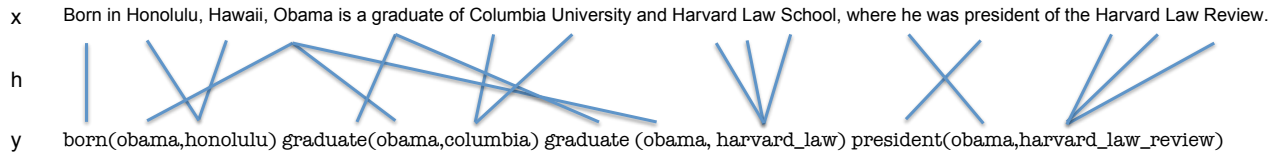
Even if we just restrict the set of output symbols to the ones used for the sentence interpretation, the first example is still more difficult. Consider the alignment between the two logical forms depicted in figure 8.1. In the Geoquery example (figure 8.1(b)) this alignment maps a single word to a single entity. In Geoquery and similar database domains this correspondence occurs frequently. This happens both because the conceptual decomposition of the domain into database entities aligns well with lexical items, but also because this alignment appears explicitly in the text– each logical entity can typically be traced back to a distinct word.

When moving away from such structured settings, this correspondence no longer exists. For example, in the first example (8.1(a)) the entity $obama$ is used four times, however there is a single occurrence of the word *"Obama"* in the text. From a cognitive perspective specifying the entity each time it is used is wasteful and redundant. From an automated semantic interpretation perspective this requires making non-trivial inferences identifying correctly the elliptical constructions (e.g., *"Born in Honolulu"*) and anaphoric expressions (e.g., *"he was the president"*).

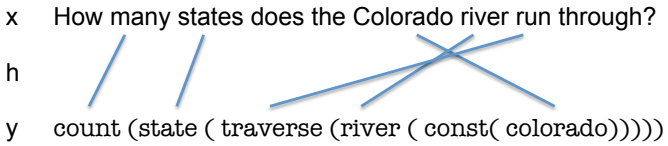
The key question that we try to answer in this section is how to quantify interpretation difficulty. We limit our discussion to LNI-like settings, in which the goal is to communicate knowledge to a computer system. Other currently unsolved aspects of natural language interpretation that require deeper cognitive abilities such as understanding humor and sarcasm, or making pragmatic inferences lie beyond the scope of this discussion.

The difficulty of scaling semantic interpretation to more complex domains could be broken down to three main aspects, a linguistic aspect (input complexity), a formal meaning representation aspect (complexity of the set of output symbols) and the alignment between the two. In the domains described in this work, and in the field in general, all three aspects are relatively simple: the natural language input uses a limited vocabulary, the output language is defined over a closed and well defined set of predicates, and mapping between the two can almost always be done at a lexical level. Each one of these aspects represents a restriction on the degree of freedom of the human-machine interaction.

While finding a generalized solution for semantic interpretation of open-domain language is beyond the scope of a single work, some advances tackling more complicated expressions could be made. In this section we discuss several useful extensions.



(a) Open-domain example, taken from the Wikipedia entry for Barak Obama



(b) A geoquery database language example.

Figure 8.1: Two example of semantic interpretation, of varying difficulty. The first is an example of open-domain interpretation, taken from a wikipedia article about President Obama, the second is an example of a geographical query. The differences between these two examples, and the difficulty they present, are discussed in section 8.1.1

8.1.2 Natural Language Input

Vocabulary The domains described in this work are defined over a limited vocabulary. The first and most immediate obstacle of scaling up semantic parsing is dealing with a larger vocabulary. In the example above (figure 8.1(a)), the relation `graduate(obama, columbia)` is described using a noun pattern, *"a graduate of"*, this could be replaced by a verb *"graduated"*, a completely different term *"his alma mater"* or *"received his degree"*. In general, identifying how relations and entities will be referenced is hard to predict, and since language is a dynamic entity, these references change over time.

From a technical perspective, working with a larger vocabulary results in a larger model with more parameters, requiring substantially more training data. In our work we took a first step towards dealing with this problem by adding an external knowledge resource, WordNet. This resource allowed us to approximate the relations between known words which appeared as input in previous semantic predictions, and previously unseen words. Using this resource is a noisy substitute for annotated data.

Syntactic Complexity and Discourse Structure Representation Making compositional decisions is typically considered to be driven by the syntactic structure of the sentence (e.g., Montague [1970]). However when observing the features and resources study done over the geoquery dataset (section 5.1.8), the contribution of this information to the overall success of the interpretation process is limited. This suggests that the natural language sentences used in this domain are of limited complexity, and as can be seen in figure 8.1(b), the compositional decisions involved in constructing the correct output correspond to word-order, or are heavily constrained by the predicate argument types.

When moving to a sentence with more complex structure this information has to be represented and used, in order to generate the correct output. In example 8.1(a) the syntactic structure is more complex. Handling such sentences correctly require dealing with several phenomena such as ellipsis, anaphora resolution, plural and more. There has been a little work dealing with more complex natural language inputs. In (Zettlemoyer and Collins [2009]) the authors present a context dependent interpretation model, capturing a larger discourse context and using it in the interpretation process.

Non-Grammatical Input and Disfluency The natural language input described in this work is fluent and clear, which is typical of text, as opposed to spoken language which contains repetitions and disfluencies. In the context of LNI, motivated by human computer communication, online speech acts are an important component, these however are often noisy and contain non-grammatical expressions. Consider for example the following sentence - *"Kick the ball to the left player. Hmmm, no, the right one"*. In this case the speaker

corrects the command while speaking. A different example could be *"Kick the ball to the left player, number 12, the one in the back"*. In this case the speaker makes several references to the same player, using different attributes of the player and the field. A simple model, relying on a one-to-one mapping between text and meaning representation will not be able to correctly identify that the expression refers to a single entity.

8.1.3 Domain Meaning Representation Language

Moving Towards Open Domain Interpretation Most semantic parsing domains currently studied map natural language sentences to a small set of symbols. This makes the interpretation process considerably easier as it reduces ambiguity. For example, the word "bank" is a polysemous word, identifying the correct sense of the word requires understanding and representing the context in which it appears, its POS tag etc. However, when working with financial data, the polysemy is essentially nonexistent, as only one sense of the word is used.

In order to allow semantic interpretation to scale to realistic settings, a large comprehensive domain is needed. Several such real world domains exist, for example, Freebase (Krishnamurthy and Mitchell [2012]) and Wikipedia (Ratinov et al. [2011]) were both used as external knowledge bases providing a list of concepts to which NL constituent could be grounded. Using this type of knowledge bases makes parsing more complex, but also provides more information to the interpretation process, as concepts are often organized hierarchically according to specified categories, with connections and links to other relevant concepts.

Identifying and Reusing Structure in Output Domain In this work we discuss cross domain transfer learning at the intermediate layer: a layer dedicated to learning abstract representation of meaning that can be shared across several domains. A different form of information sharing across domains is to use symbols learned in one domain in another domain by altering the original meaning according to the new domain language. Consider for example the sentence - *"The color of the cards should alternate"*, The term *"alternating colors"* refers to a condition about repetitive patterns in the color of cards. The sentence *"The processes should alternate"*, describes a condition about scheduling policies of computer processes. While the meaning of the two sentences is different, the meaning of the term "alternate" is similar in both domains, but rather it is attached to different items.

In this example, a function symbol $\texttt{alternate}(\cdot,\cdot)$ which takes in function symbols as arguments would allow us to capture this similarity: $\texttt{alternate}(\texttt{card}(\cdot),\texttt{color}(\cdot))$ for one domain, and for the other $\texttt{alternate}(\texttt{process}(\cdot),\texttt{runTime}(\cdot))$. In general, using a representation that maps terms to higher order

functions would allow expressing this similarity.

Symbolic Atoms vs. Real World Atoms Our setup for LNI considers only symbolic representations of the domain. The external world is limited to a small set of function and entities, and language understanding is the process of grounding natural language in these set of symbols. In many realistic settings the external environment is not so easily accessible, and moving from a real world sensory input to an internal symbolic representation is not a deterministic and precise step. There has been some work dealing with real world agents taking natural language commands and responding to it, most recently (Matuszek et al. [2012]) has considered learning vision classifiers and grounding them as symbolic atoms in language interpretation, in a learning settings related to response-based learning.

8.1.4 The Correspondence Between Natural and Domain Representation Languages

Representation Languages Mismatch Given a natural language sentence, the assumption guiding interpretation is that there is an almost perfect alignment between the output representation and natural language constituents. This assumption does not hold in many cases, consider for example the sentence - *"Player1 and Player2 passed the ball repeatedly"*, this sentence succinctly describes a series of events, for example – $\texttt{pass}(\texttt{p1}, \texttt{p2}),\ \texttt{pass}(\texttt{p2}, \texttt{p1}),\ \texttt{pass}(\texttt{p1}, \texttt{p2})$. In this case, there is no such alignment, and even worse, the natural language sentence does not describe a specific scene, but rather several possibilities of scenes (in which the two players pass the ball a varying number of times).

A similar scenario will be encountered when the level of granularity in event representation does not align between the natural language and logical output language. For example, when using a term describing several low level actions in the logical output language – "Make the dough" may not correspond directly to a single predicate $\texttt{make}(\texttt{dough})$ in the output representation, but rather involves a series of steps (mixing flour, eggs, etc.), which are not described by this sentence directly.

These scenarios are likely to repeat frequently, as human tend to use language intelligently and succinctly, resorting to short descriptions encoding the relevant information implicitly. Relaxing the strong alignment requirement as described above, would make learning more complex but would allow dealing with such cases. In some cases, such as the first example, the semantic interpretation model needs to make a relatively small accommodation, to capture repetition. The second example call for a deeper representation of the domain and the outcome of interactions.

8.1.5 The Human Factor

In LNI the role of a human teacher was limited to providing response based feedback. The alternative, supervised learning, is far more demanding to the human teacher, a fact motivating our approach. Nonetheless, the role of human teaching could be extended to provide a more meaningful feedback to the learner, both in terms of supervision, the representation used for learning, and the way to information is presented to the automated student. The need for a deeper form of human involvement becomes more important as the learned tasks become more complex, and cannot be expressed directly and succinctly.

Curriculum Learning The notion of curriculum learning (Bengio et al. [2009]) has been studied in the machine learning literature, mostly as a training regime that provides increasingly more difficult training examples to the learner, thus allowing it to learn simple concepts first and then move to more complicated ones. This idea applies in these settings as well. When teaching an agent knowledge about a new domain, a structured hierarchical learning curriculum can be applied. Consider for example the card game domain, a curriculum starting with simple concepts such as cards and their properties could be taught first, along with the relevant lexical information required to describe them. This approach will make learning complicated concepts easier both from an application perspective (more complicated concepts), and a semantic interpretation perspective (more complicated NL rules).

Appendix A

The Robocup and Robocup VERBOSE Domains

A.1 Robocup

This section contains the domain representation language used in the Robocup domain, as published by Chen and Mooney [2008].

*n:Player → (pink1)

*n:Player → (pink2)

*n:Player → (pink3)

*n:Player → (pink4)

*n:Player → (pink5)

*n:Player → (pink6)

*n:Player → (pink7)

*n:Player → (pink8)

*n:Player → (pink9)

*n:Player → (pink10)

*n:Player → (pink11)

*n:Player → (purple1)

*n:Player → (purple2)

*n:Player → (purple3)

*n:Player → (purple4)

*n:Player → (purple5)

*n:Player → (purple6)

*n:Player → (purple7)

*n:Player → (purple8)

*n:Player → (purple9)

*n:Player → (purple10)

*n:Player → (purple11)

*n:Playmode → (kick_off_l)

*n:Playmode → (kick_off_r)

*n:Playmode → (kick_in_l)

*n:Playmode → (kick_in_r)

*n:Playmode → (play_on)

*n:Playmode → (offside_l)

*n:Playmode → (offside_r)

*n:Playmode → (free_kick_l)

*n:Playmode → (free_kick_r)

*n:Playmode → (corner_kick_l)

*n:Playmode → (corner_kick_r)

*n:Playmode → (goal_kick_l)

*n:Playmode → (goal_kick_r)

*n:Playmode → (goal_l)

*n:Playmode → (goal_r)

*n:S → (playmode (*n:Playmode))

*n:S → (ballstopped)

*n:S → (turnover (*n:Player , *n:Player))

*n:S → (kick (*n:Player))

*n:S → (pass (*n:Player , *n:Player))

*n:S → (badPass (*n:Player , *n:Player))

*n:S → (defense (*n:Player , *n:Player))

*n:S → (steal (*n:Player))

*n:S → (block (*n:Player))

A.2 Robocup VERBOSE

This section contains the domain representation language used in the Robocup VERBOSE domain described in this work.

*n:Player → (pink1)

*n:Player → (pink2)

*n:Player → (pink3)

*n:Player → (pink4)

*n:Player → (pink5)

*n:Player → (pink6)

*n:Player → (pink7)

*n:Player → (pink8)

*n:Player → (pink9)

*n:Player → (pink10)

*n:Player → (pink11)

*n:Player → (purple1)

*n:Player → (purple2)

*n:Player → (purple3)

*n:Player → (purple4)

*n:Player → (purple5)

*n:Player → (purple6)

*n:Player → (purple7)

*n:Player → (purple8)

*n:Player → (purple9)

*n:Player → (purple10)

*n:Player → (purple11)

*n:Team → (Pink)

*n:Team → (Purple)

*n:Ball → (Ball)

*n:Playmode → (kick_off_l)

*n:Playmode → (kick_off_r)

*n:Playmode → (kick_in_l)

*n:Playmode → (kick_in_r)

*n:Playmode → (play_on)

*n:Playmode → (offside_l)

*n:Playmode → (offside_r)

*n:Playmode → (free_kick_l)

*n:Playmode → (free_kick_r)

*n:Playmode → (corner_kick_l)

*n:Playmode → (corner_kick_r)

*n:Playmode → (goal_kick_l)

*n:Playmode → (goal_kick_r)

*n:Playmode → (goal_l)

*n:Playmode → (goal_r)

*n:S → (playmode (*n:Playmode))

*n:S → (playmode (*n:Playmode,*n:Team))

*n:S → (ballstopped)

*n:S → (stop (*n:Ball))

*n:S → (turnover (*n:Player , *n:Player))

*n:S → (turnover (*n:Player , *n:Player , *n:Ball))

*n:S → (kick (*n:Player))

*n:S → (kick (*n:Player , *n:Ball))

*n:S → (pass (*n:Player , *n:Player))

*n:S → (pass (*n:Player , *n:Player , *n:Ball))

*n:S → (badPass (*n:Player , *n:Player))

*n:S → (defense (*n:Player , *n:Player))

*n:S → (steal (*n:Player))

*n:S → (steal (*n:Player , *n:Player))

*n:S → (steal (*n:Player, *n:Ball))

*n:S → (steal (*n:Player, *n:Player , *n:Ball))

*n:S → (block (*n:Player))

*n:S → (block (*n:Player , *n:Ball))

*n:S → (block (*n:Player , *n:Player))

*n:S → (block (*n:Player , *n:Player, *n:Ball))

Appendix B

Detailed Review of Features Extraction

In this appendix we review in detail the feature representation used for learning a semantic interpreter. The purpose of this review is to provide the reader with exact definitions of the feature functions used in this work and the way these function were implemented. The starting point of this review is the Learning-Based Java (LBJ) feature extraction script provided in section B.2. The LBJ script defines two *classifiers*, feature extractors, mapping programming objects representing first-order (α) decisions, and second-order (β) decisions, into feature vectors. The specific feature functions called by the LBJ script are implemented in code. We describe the class-relationship in figure B.1, and the specific feature functions used in section B.1. We begin our review by providing a brief description of two resources used in implementing this process: the LBJ framework, and the curator.

Learning Based Java (LBJ) is a modeling language, designed and implemented to allow rapid development of learning and inference systems. The LBJ programming language provides powerful learning and inference tools and a feature extraction language. For further details the reader is referred to (Rizzolo and Roth [2010]).

The Curator is a server framework for managing multiple natural language processing components. The Curator framework provides a unified access to data structures holding the results of NLP processing. When used in conjunction with Edison, an NLP software library, the combined framework offers additional functionality, including such elements as caching of processing results, unified multi-view representation and useful tools to query these views. In our implementation of the feature functions, we use this server based system to provide us with linguistic annotation for feature extractions. For further details the reader is referred to (Clarke et al. [2012]).

B.1 Feature Functions

Lexical Features These feature capture properties of mapping between a word and a logical domain object. Some of these features depend on a lexicon, mapping words to predicate symbols, that is constructed incrementally during training.

- **StemGlossMatch(w,p)** Does the stem of a word w match an entry for p in the initial lexicon? (new feature for every predicate).
- **StemPredicateGlossMatch(w,p)** Does the stem of a word w match a lexicon item for p? (new feature for every word, predicate pair).
- **LearnLexiconGlossMatch(w,p)** Does the stem of a word w match an entry for p in the learned lexicon? (new feature for every predicate).
- **LexiconGlossMatchCho(w,p)** Does the stem of a word w match an entry for p in the learned lexicon? (new feature for every word, predicate pair).
- **StemmedGlossAndContextMatch(w,p,s)** Does the stem of a word w match a lexicon item for p and a word in the window around w in the input sentence s match another lexicon item? (New feature for each predicate pair).
- **NullPredicateMatch(w)** Does the word w match a null predicate? (two features, one for null-predicate mapping to lexicon items, and one for non-lexicon items).
- **WNSimGloss(w,p)** What is the highest WNSim score assigned to w and any of the lexicon items for p? (single numeric feature).
- **PrepositionContext(w,p,s)** Does w match a lexicon entry for p and is a preposition in a window around w in s? (new feature for ever predicate and context word pair).

Example 11. *The following example describes an input sentence and the lexical features that will be extracted for a given output.*
Input: "How many people live in Texas?"
Output: `population(const(texas))`

Assume the following alignments: (*"population"*,`population`), (*"Texas"*,`const(texas)`), and that both words appear in the initial lexicon for their corresponding logical items, and also the mapping (*"many"*, `count`) appears in the lexicon.

The following features will be extracted for the first alignment: StemGlossMatch(`population`), StemPredicateGlossMatch(*"population"*,`population`), StemmedGlossAndContextMatch(*count* ,`population`).

Feature Type	Feature Functions	Feature Instantiation
1) Lexicon features	StemGlossMatch	New feature: each predicate. Initial Lexicon
	StemPredicateGlossMatch	New feature: each predicate+word. Initial Lexicon
	StemGlossAndContextMatch	New feature: each predicate pair.
	LearnLexiconGlossMatch	New feature: each predicate. Learned lexicon
	LexiconGlossMatchCho	New feature: each predicate+word. Learned lexicon
2) NullPredicate	NullPredicateMatch	New feature: lexicon,non-lexicon
3) WNSimilarity	WNSimGloss	Single numeric feature
4) PreposContext	PrepositionContext	New feature: each predicate+context location
5) DepeDistance	DependencyDistance	Single numeric feature
6) TopDistance	HeadDistance	Single numeric feature
	HeadBigram	New feature: each predicate
7) Bigram	BigramPredicatesIO	New feature: each ordered predicate pair

Table B.1: Detailed feature description table. The first column describes the feature type, the second describes the LBJ feature function implementing it, and the third describes how the feature was instantiated.

Compositional Features These features capture the properties of a compositional decision by taking into account properties of the resulting logical structure and the syntactic structure of the words originating the logical symbols.

- **DependencyDistance(**w_1, w_2**,s)** The distance between the nodes of w_1 and w_2 in the dependency tree of s. The distance is normalized by the length of the input sentence. (single numeric feature).
- **HeadDistance(w,s)** If w is aligned with the head predicate, what is the index of w in s? (single numeric feature)
- **HeadBigram(p)** The predicate p is the head predicate. (new feature for each predicate).
- **BigramPredicatesIO(**p_1, p_2**)** The ordered pair of predicates p_1, p_2 appear in a second-order decision. (new feature for every predicate pair).

Continuing with example 11, the following features will be generated for the pair `population(const(texas))`. DependencyDistance(*"population"*,*"Texas"*) with a numeric activation value of 0.333, HeadDistance("population") with a numeric activation value of 0.5, HeadBigram(`population`), BigramPredicatesIO(`population`, `const(texas)`)

B.2 Learning Based Java Feature Extraction Script

```
real% LexiconGlossMatchChoice(ConstituentPredicateAlignment cpa) ← {
   String result =":"+ cpa.getPredicate().getSimpleName() +"!!"+cpa.getConstituent().getSurfaceString().toLowerCase();
if (FeatureFunctions.glossMatch(cpa, true))
    sense result: 1.0; else
    sense result: 0.0;
```

```
}

real% LearnedLexiconGlossMatch(ConstituentPredicateAlignment cpa) ← {
      String result = cpa.getPredicate().isConstantPredicate() ? ”constant” : ”predicate”;
       if (FeatureFunctions.learnedGlossMatch(cpa, true))
            sense result: 1.0;
       else
            sense result: 0.0;
}

real% NoGlossMatch(ConstituentPredicateAlignment cpa) ← {
      String result = cpa.getPredicate().isConstantPredicate() ? ”constant” : ”predicate”;
       if (FeatureFunctions.glossMatch(cpa, true))
            sense result: 0.0;
       else
            sense result: 1.0;
}

real% NullPredicateMatch(ConstituentPredicateAlignment cpa) ← {
       String result = FeatureFunctions.nullPredicate(cpa);
      if (result != null) {
            sense result: 1.0;
      }
}
real% StemmedGlossMatch(ConstituentPredicateAlignment cpa) ← {
      String result = cpa.getPredicate().isConstantPredicate() ? ”constant” : ”predicate”;
       if (FeatureFunctions.glossMatch(cpa, true))
            sense result: 1.0;
       else
            sense result: 0.0;          }

real% WNSimGloss(ConstituentPredicateAlignment cpa) ← {
      String result = cpa.getPredicate().isConstantPredicate() ? ”constant” : ”predicate”;
       double wnsim = FeatureFunctions.wnsimGloss(cpa);
      sense result: wnsim;
```

```
    if (!cpa.getPredicate().isConstantPredicate())
        sense cpa.getPredicate().getSimpleName() : wnsim;
    }

real% PrepositionContext(ConstituentPredicateAlignment cpa) ← {
    String result = FeatureFunctions.prepositionContext(cpa, -1, true);
    if (result != null) {
        sense result: 1.0;
    }
}

real% StemmedPredicateGlossMatch(ConstituentPredicateAlignment cpa) ← {
    String result = FeatureFunctions.predicateGlossMatch(cpa, true);
    if (result != null) {
        sense result: 1.0;
    }
}

real% StemmedGlossAndContextMatch(ConstituentPredicateAlignment cpa) ← {
    String[] left = FeatureFunctions.glossAndContextMatch(cpa, -1, true);
    for (int i = 0; i ¡ left.length; i++)
        sense ”left:”+left[i]: 1.0;
    String[] right = FeatureFunctions.glossAndContextMatch(cpa, 1, true);
    for (int i = 0; i ¡ right.length; i++)
        sense ”right:”+right[i]: 1.0;
}

real DependencyDistance(ArgumentDecision[] ads) ← {
    return FeatureFunctions.dependencyDistance(ads[0].getAlignment(), ads[1].getAlignment());
}

real HeadDistance(ArgumentDecision[] ads) ← {
    return FeatureFunctions.headDistance(ads[0].getAlignment(), ads[1].getAlignment());
}
```

```
real% BigramPredicates(ArgumentDecision[] ads) ← {
    sense FeatureFunctions.bigramPredicates(ads[0].getAlignment(), ads[1].getAlignment()) : 1.0;
}

real% BigramPredicatesIO(ArgumentDecision[] ads) ← {
    sense FeatureFunctions.bigramPredicatesIO(ads[0], ads[1]) : 1.0;
}

real% HeadBigram(ArgumentDecision[] ads) ← {
    String result = FeatureFunctions.headBigram(ads[0].getAlignment(), ads[1].getAlignment());
if (result != null) {
    sense result : 1.0;
    }
}


    real% CPAFeatureGenerator(ConstituentPredicateAlignment cpa) ←
StemmedGlossMatch, StemmedPredicateGlossMatch, NullPredicateMatch, StemmedGlossAndContextMatch, WNSim-
Gloss, PrepositionContext, LearnedLexiconGlossMatch, LexiconGlossMatchChoice
real% SecondOrderCPAFeatureGenerator(ArgumentDecision[] ads) ← DependencyDistance, HeadDistance, Bigram-
PredicatesIO, HeadBigram
end
```

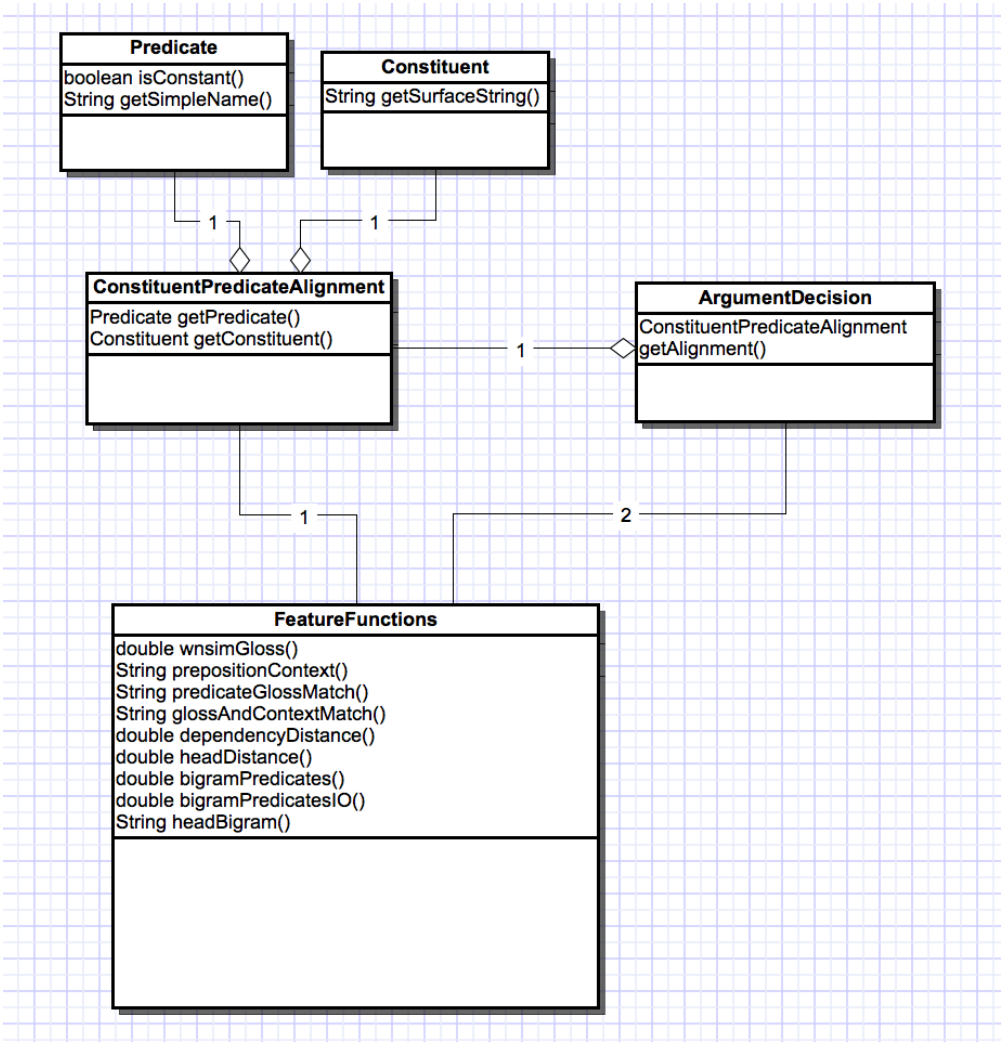


Figure B.1: Class diagram of the feature extraction package and relevant classes as implemented in Java. The feature extraction process is defined using the LBJ file. In this case the file defines two classifiers, one for extracting features from first-order decisions instances and another one for extracting features from second order instances. Each classifier uses a different set of features. From a programmatic point of view, the first-order decision are represented using the *ConstituentPredicateAlignment* class, the second-order decision are represented using a pair of *ArgumentDecision* instances. The feature functions referenced in the LBJ files are implemented in class *FeatureFunctions*, these functions are described in section B.1.

References

E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. In *Proceedings of the 17th International Workshop on Machine Learning*, 2000.

Y. Artzi and L. Zettlemoyer. Bootstrapping semantic parsers from conversations. In *EMNLP*, 2011.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, 2009.

A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, 1998.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, 1992.

S.R.K. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *ACL*, 2009.

S.R.K. Branavan, D. Silver, and R. Barzilay. Learning to win by reading manuals in a monte-carlo framework. In *ACL*, 2011.

X. Carreras and L. Màrquez. Introduction to the CoNLL-2004 shared tasks: Semantic role labeling. In *Proceedings of CoNLL-2004*, 2004.

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised l earning algorithms. In *ICML*, 2006.

M. Chang, L. Ratinov, and D. Roth. Guiding semi-supervision with constraint-driven learning. In *ACL*, 2007.

M. Chang, D. Goldwasser, D. Roth, and Y. Tu. Unsupervised constraint driven learning for transliteration discovery. In *NAACL*, 2009.

M. Chang, D. Goldwasser, D. Roth, and V. Srikumar. Discriminative learning over constrained latent representations. In *NAACL*, 2010a.

M. Chang, D. Goldwasser, D. Roth, and V. Srikumar. Structured output learning with indirect supervision. In *Proc. of the International Conference on Machine Learning (ICML)*, 2010b.

E. Charniak. Towards a model of children's story comprehension. In *AI-TR266, MIT AI Laboratory*, 1972.

D. Chen and R. Mooney. Learning to sportscast: a test of grounded language acquisition. In *ICML*, 2008.

D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, 2011.

D. L. Chen, J. Kim, and R. J. Mooney. Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*, 37:397–435, 2010.

J. Chu-Carroll, J. Prager K. Czuba, and A. Ittycheriah. In question answering, two heads are better than on. In *NAACL*, 2003.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. Driving semantic parsing from the world's response. In *CoNLL*, 7 2010.

J. Clarke, V. Srikumar, M. Sammons, and D. Roth. An nlp curator (or: How i learned to stop worrying and love nlp pipelines). In *LREC*, 2012.

M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

M. Collins and Y. Singer. Unsupervised models for named entity classification. In *EMNLP–VLC*, 1999.

M. Connor, C. Fisher, and D. Roth. Online latent structure training for language acquisition. In *IJCAI*, 7 2011.

A. Culotta and A. McCallum. Confidence estimation for information extraction. In *NAACL*, 2004.

Q. Do, D. Roth, M. Sammons, Y. Tu, and V. Vydiswaran. Robust, light-weight approaches to compute lexical similarity. In *Computer Science Research and Technical Reports, University of Illinois - 2009*, 2009.

J. Eisenstein, J. Clarke, D. Goldwasser, and D. Roth. Reading to learn: Constructing features from semantic abstracts. In *EMNLP*, 2009.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 1999.

D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 2002.

D. Goldwasser and D. Roth. Learning from natural instructions. In *IJCAI*, 2011.

D. Goldwasser, R. Reichart, J. Clarke, and D. Roth. Confidence driven unsupervised semantic parsing. In *ACL*, 2011.

C.L Isbell, M. Kearns, S. Singh, C. Shelton, P. Stone, and D. Kormann. Cobot in lambdamoo: An adaptive social statistics agent. In *AAMAS*, 2006.

R. Kate. Transforming meaning representation grammars to improve semantic parsing. In *CoNLL*, 2008.

R. Kate and R. Mooney. Using string-kernels for learning semantic parsers. In *ACL*, 2006.

R. Kate, Y.W. Wong, and R.J. Mooney. Learning to transform natural to formal languages. In *AAAI*, 2005.

J. Kim and R. J. Mooney. Generative alignment and semantic parsing for learning from ambiguous supervision. In *COLING*, 2010.

D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *NIPS*, 2003.

B. Knox and P. Stone. Interactively shaping agents via human reinforcement. In *KCAP*, 2009.

Y. Koo, C. Lee, and B. Juang. Speech recognition and utterance verification based on a generalized confidence score. *IEEE Transactions on Speech and Audio Processing*, 9(8):821–832, 2001.

J. Krishnamurthy and T. Mitchell. Weakly supervised training of semantic parsers. In *EMNLP*, July 2012.

G. Kuhlmann, P. Stone, R. J. Mooney, and J. W. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *AAAI workshops*, 2004.

T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, , and M. Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *EMNLP*, 2010.

P. Liang, M. I. Jordan, and D. Klein. Learning semantic correspondences with less supervision. In *ACL*, 2009.

P Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *ACL*, 2011.

F. Lin and F. Weng. Computing confidence scores for all sub parse trees. In *ACL*, 2008.

A. Martins, N. A. Smith, and E. Xing. Concise integer linear programming formulations for dependency parsing. In *ACL*, 2009.

C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. A joint model of language and perception for grounded attribute learning. In *ICML*, 2012.

D. McClosky, E. Charniak, and Mark Johnson. Effective self-training for parsing. In *NAACL*, 2006.

G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K.J. Miller. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 1990.

R. Montague. Universal grammar. In *Theoria*, pages 36,373–398, 1970.

F. C. N. Pereira and S. M. Shieber. *Prolog and Natural-Language Analysis*, volume 10 of *CSLI Lecture Notes Series*. Center for the Study of Language and Information, 1987.

H. Poon and P. Domingos. Unsupervised semantic parsing. In *EMNLP*, 2009.

V. Punyakanok, D. Roth, and W. Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 2008.

L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, 2011.

R. Reichart and A. Rappoport. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *ACL*, 2007a.

R. Reichart and A. Rappoport. An ensemble method for selection of high quality parses. In *ACL*, 2007b.

R. Reichart, R. Fattal, and A. Rappoport. Improved unsupervised pos induction using intrinsic clustering quality and a zipfian constraint. In *CoNLL*, 2010.

N. Rizzolo and D. Roth. Learning based java for rapid development of nlp systems. In *LREC*, 2010.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

B. Rosenfeld and R. Feldman. Using corpus statistics on entities to improve semi–supervised relation extraction from the web. In *ACL*, 2007.

D. Roth and W. Yih. Global inference for entity and relation identification via a linear programming formulation. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, 2007.

Xu Sun, Takuya Matsuzaki, , Daisuke Okanohara, and Junichi Tsujii. Latent variable perceptron algorithm for structured classication. In *IJCAI*, 2009.

L. Tang and R. Mooney. Automated construction of database interfaces: integrating statistical and relational learning for semantic parsing. In *EMNLP*, 2000.

L. R. Tang and R. J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*, 2001.

A. L. Thomaz and C. Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI*, 2006.

I. Titov and A. Klementiev. A bayesian model for unsupervised semantic parsing. In *ACL*, 2011.

I. Titov and M. Kozhevnikov. Bootstrapping semantic analyzers from non-contradictory texts. In *ACL*, 2010.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

N. Ueffing and H. Ney. Word-level confidence estimation for machine translation. *Computational Linguistics*, 33(1):9–40, 2007.

A. Vogel and D. Jurafsky. Learning to follow navigational directions. In *ACL*, 2010.

T. Winograd. Understanding natural language. In *Academic Press*, 1972.

Y.W. Wong and R. Mooney. Learning for semantic parsing with statistical machine translation. In *NAACL*, 2006.

Y.W. Wong and R. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *ACL*, 2007.

D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised method. In *ACL*, 1995.

A. Yates, S. Schoenmackers, and O. Etzioni. Detecting parser errors using web-based semantic filters. In *EMNLP*, 2006.

C. Yu and T. Joachims. Learning structural svms with latent variables. In *ICML*, 2009.

J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic proramming. In *AAAI*, 1996.

L. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, 2005.

L. Zettlemoyer and M. Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *CoNLL*, 2007.

L. Zettlemoyer and M. Collins. Learning context-dependent mappings from sentences to logical form. In *ACL*, 2009.