Diplomaterv

Neu Gergely

2008.

Nyilatkozat

Alulírott *Neu Gergely*, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Neu Gergely
hallgató

Kivonat

A diplomaterv célja az inverz megerősítéses tanulás (inverse reinforcement learning, IRL) technikáinak áttekintése, azoknak elméleti és gyakorlati összehasonlítása. Az inverz megerősítéses tanulás egy újszerű tanulási paradigma, amely köthető mind a felügyelt, mind a megerősítéses tanuláshoz. Bár az IRL problémájának megoldására számos algoritmus született az utóbbi években, a megoldások széles körű elméleti és kísérleti összehasonlítása eddig nem került publikálásra. A dolgozatban rámutatok, hogy olyan egységes formalizmus dolgozható ki, amelynek segítségével felismerhetők egyes korábbi megoldások hasonlóságai, és hogy egyes, korábban önállónak tekintett algoritmusok egy nagyobb algoritmusosztály speciális esetei.

A dolgozatban új megközelítést javaslok az IRL probléma megoldásához, amely kvadratikusan korlátozott kvadratikus programozáson alapul. A konstruált kvadratikus programozási feladat megoldására szubgradiens-módszert mutatok be. Az új módszert kísérletileg összehasonlítom néhány korábbi módszerrel. A kísérletek eredményei szerint az új algoritmus a korábbiaknál hatékonyabban használja fel a rendelkezésre álló mintákat, valamint kevésbé érzékeny a probléma hiperparamétereinek bizonyos típusú perturbációira.

A dolgozat második felében bemutatom az IRL egy új alkalmazását a strukturált predikció területén. Analógiát állítok fel a Markov döntési folyamatok és a sztochasztikus környezetfüggetlen nyelvtanok által generált mondatok elemzése között, amely analógia segítségével IRL-alapú algoritmusokat javaslok mondatelemzők tanítására. Megmutatom, hogy a vizsgált korábbi IRL algoritmusok alternatív levezetéssel már korábban is használatosak voltak ezen a területen, azonban a diplomatervben bemutatott új algoritmus alkalmazása teljesen új tanítási eljárást eredményez. A kapott módszereket a területen standardnak számító körülmények között kísérletileg összehasonlítom, valamint egyéb, fontosságuk ellenére kevésbé szokványos vizsgálatokat is elvégzek. Az eredmények szerint az új algoritmus versenyképes a jelenleg használt csúcsteljesítményű algoritmusokkal szemben is.

Abstract

The aim of this thesis is to give a detailed overview on inverse reinforcement learning (IRL) techniques, to compare existing IRL methods both theoretically and empirically. Inverse reinforcement learning is a relatively new learning paradigm situated between supervised learning and reinforcement learning. Although several methods were proposed for solving the IRL problem in the recent years, no theoretical or empirical comparison of these methods is available in the literature. This thesis introduces a unified framework that helps to recognize similarities between previous solutions, and to show that some known methods are specializations of a larger class of algorithms.

In this thesis we propose a new quadratic-programming-based approach for solving the IRL problem. We introduce a subgradient-based solution for this quadratic programming problem. The new method is empirically compared to some previous methods. The results of the experiments suggest that the new algorithm is more sample efficient than the examined previous ones, and that it is less sensitive to some specific perturbations of the hyperparameters of the problem.

In the second part of the thesis we propose a new application of IRL in the field of structured prediction. We introduce an analogy between Markovian decision processes and parsing sentences generated by probabilistic context-free grammars, and exploit this analogy to propose IRL-based parser training algorithms. We show that while the methods derived from previous IRL algorithms are known top-performing discriminative training methods in the structured prediction domain, the method proposed in this thesis yields a completely new training method. Besides experimentally comparing all the IRL-based methods in a standardized setting, we also perform other important new experiments. The results suggest that the new method proposed in the thesis achieves state-of-the-art performance in discriminative parser training.

Contents

Chapter 1

Introduction

This thesis presents a new, unified approach to *inverse reinforcement learning* (IRL) and an application to natural language parsing. Inverse reinforcement learning is a relatively new machine learning technique, situated between traditional reinforcement learning (RL) and supervised learning. The ultimate goal of IRL is learning an optimal behavior to navigate in a *Markovian decision process* as in straightforward RL. However, the learning process is driven by (positive) examples of desired behavior as in supervised learning.

The aim of reinforcement learning algorithms is to learn a behavior that maximizes a long-term performance measure, such as the total rewards. A behavior is a mapping from past observations to actions. The actions influence the state of the environment, which in turn determines the future observations and the rewards. Through influencing the state, actions have long-term effects and agents maximizing a long-term performance measure must take into account such effects. This is a flexible framework: numerous problems can and have been posed as reinforcement learning problems (e.g., see Sutton and Barto, 1998).

The definition of an optimal behavior in IRL is similar to that in RL. The difference is that the agent is given examples from an *expert*, who performs a specific task in a professional way. In many cases, it is reasonable to think that the expert acts optimally with respect to its own reward function – the idea of IRL is to clone the expert's behavior by finding the reward function

that the expert employs to optimize its own behavior.

In the rest of this Chapter, we give further motivation for our research. After giving the necessary background on Markovian decision processes in Chapter 2, the first contribution of the thesis is presented in Chapter 3. In this chapter, we introduce our unifying framework for IRL, describe some previous methods using this formalism, and propose a new IRL algorithm. We experimentally compare our method to the previous methods in an artificial domain. The second part of the thesis (Chapter 4) proposes a novel application of IRL to natural language parser training. We introduce an analogy between Markovian decision processes and parsing sentences of *probabilistic context-free grammars* (PCFGs), and exploit this analogy to propose new parser training algorithms based on IRL. Our experiments (presented in Section 4.4) show that the new parser training algorithm based on our IRL algorithm achieves state-of-the-art performance in parser training.

1.1 Apprenticeship learning

The aim of apprenticeship learning is to estimate a policy of an expert based on samples of the expert's behavior. The relevant literature is too large to survey here; examples include Sammut et al. (1992), Kuniyoshi et al. (1994), Demiris and Hayes (1996), Amit and Mataric (2004), Pomerleau (1988), Atkeson and Schaal (1997), Price and Boutilier (2003), Silva et al. (2006) and Ramachandran and Amir (2007).

In apprenticeship learning (a.k.a. imitation learning) one can distinguish between *direct* and *indirect approaches*. Direct methods attempt to learn the policy (as a mapping from states, or features describing states to actions) by resorting to a supervised learning method. They do this by optimizing some loss function that measures the deviation between the expert's policy and the policy chosen. The main problem then is that in parts of the state space that the expert tends to avoid the samples are sparse and hence these methods may have difficulties learning a good policy at such places.

In an indirect method it is assumed that the expert is acting optimally in the environment, in the precise sense of maximizing the cumulated rewards

in an MDP. Inverse reinforcement learning is clearly an indirect approach to the apprenticeship learning problem. First coined by Ng and Russell (2000), IRL did not receive substantial attention until the paper by Abbeel and Ng (2004) was published[1]. Since then, numerous works emerged in this topic, including Ratliff et al. (2006), Silva et al. (2006), Ramachandran and Amir (2007) Neu and Szepesvári (2007), Syed and Schaphire (2007) and Syed et al. (2008). Although the number of IRL methods is rapidly increasing, there is no known work that aims to compare more than two of them neither theoretically, nor empirically. In Neu and Szepesvári (2007) we compare our methods to those of Abbeel and Ng (2004) experimentally, concluding the superiority of our solution. Syed and Schaphire (2007) proposed a method (MWAL) that has improved convergence guarantees as compared to those of the projection algorithm of Abbeel and Ng (2004). The paper by Syed et al. (2008) proposes a solution based on linear programming, that attains the same performance as MWAL with performing much less computations.

One of the aims of this thesis is to provide common basis to discuss IRL algorithms, to relate some existing methods to each other. We introduce a unified notation to be able to compare the methods of Abbeel and Ng (2004), Ratliff et al. (2006), and Syed and Schaphire (2007). We show that these methods are closely related to each other, and are only special cases of a larger class of algorithms. We present a novel approach to solve the IRL problem, that is, we formulate it as a quadratic programming problem, and give a subgradient-based solution to it (Neu and Szepesvári, 2007). This method can be enhanced by computing the natural gradient of the criterion function, or using the Rprop algorithm (Igel and Hüsken, 2000) to find the optimal solution.

One of the other contributions of this thesis is the extensive empirical comparison documented in it. All IRL methods require two sets as input: a set of samples taken from expert behavior, and a set of *features* encoding our prior knowledge of the task that the expert solves. We have tested the

[1]The only application of IRL that we are aware of, comes from this period: Shiraev (2003) applied the method of Ng and Russell (2000) for routing metric discovery, with no particular followup work.

methods in an artificial domain to examine how they scale with the sample size, and to test their sensitivity to the quality of the given features. We have seen that the performance of our new methods improve more rapidly with the sample size than that of the other methods, and that a natural gradient version of it completely eliminates the dependence of the solution on the scaling of the features. All methods have been seen to be roughly equally robust to using slightly perturbed features instead of the real ones. Adding irrelevant features to the feature set have been seen to have positive effect on some previous methods, but affects our new algorithms adversely. This problem can be alleviated by using regularization.

1.2 Structured prediction as behavior

Structured prediction is a classification problem with structured input and/or output spaces (Bakir et al., 2007). A prevalent idea then is that the structured output is obtained by solving an optimization problem, where the problem depends on the input and a set of tunable weights. The classifier is learnt by tuning the weights so that the outputs match those in the training data. Oftentimes dynamic programming is used to solve the optimization problem, e.g. in the classical examples of sequence labeling using hidden Markov models or parsing using probabilistic context-free grammars (Manning and Schütze, 1999), but also in more complex domains as RNA structure prediction (Rivas and Eddy, 1999), or image segmentation (Elliott et al., 1984).

The connection between inverse reinforcement learning and structured prediction is that the behavior of an expert, a policy, or the trace of a policy, can be viewed as a structured output. The main motivation of the second part of this thesis is to make this connection explicit, allowing one to derive algorithms to train structured prediction models from existing IRL methods. For the sake of specificity we consider the problem of training probabilistic context-free (PCFG) parsers as one of the most studied structured prediction problem, allowing us to make connection between existing parser training algorithms and algorithms derived from IRL algorithms.

The connection between inverse reinforcement learning and parser train-

ing is made explicit by mapping parsing problems into episodic MDPs. This idea is not completely new: The reverse connection was exploited by Ratliff et al. (2006) who derived an algorithm for inferring rewards using the large margin approach of Taskar et al. (2005). Maes et al. (2007) have used reinforcement learning for solving the structured prediction problem of sequence labeling. Here we consider the previously described IRL methods to derive parser training algorithms. Some of the resulting methods are previously known: the max-margin planning method of Ratliff et al. (2006), is optimizing the same criterion as the maximum margin parser of Taskar et al. (2004), but uses a subgradient method as proposed by Ratliff et al. (2007). Our previously introduced notation allows us to show how the perceptron algorithm (Freund and Schapire, 1999; Collins, 2002; Collins and Roark, 2004) can be derived from max-margin planning. We also show that the method of Syed and Schaphire (2007), called MWAL, is only an exponentiated gradient version of the perceptron algorithm.

The new algorithms (and classic Maximum Likelihood parameter estimation) are compared in extensive experiments on parser training using the Penn Treebank WSJ corpus. We test their sensitivity to the selection of the step-size sequence, the regularization parameter and how their performance changes with the size of the training set. We report results when performance is measured with cross-validation. The experiments show that the max-margin and the policy matching algorithms are significantly better for parser training than the other three algorithms, while the performance of these are essentially indistinguishable. Our results show that even simpler parsers are able to achieve results that are competitive with state-of-the-art performance when an appropriate parser training method is used.

We find the connection between IRL and parser training especially fruitful in that it allows one to derive parser training algorithms from any IRL method. Since parser training is not too special amongst structured prediction, we expect that similar mappings can be constructed for other domains, too (e.g., dependency parsing, prediction of tag sequences, labelling, parsing of images, etc.). Then whenever a new IRL algorithm is discovered, the discovery automatically leads to a new training algorithm for structured

predictors. The connection also suggests a number of potential future enhancements. Further robustness might be gained by considering stochastic outcomes of the labelling decisions. By changing the way the rewards depend on the states (partial parses) new, more powerful models can be created that may lead to further performance improvements. The connection to RL could also be exploited by considering value function approximation methods that may yield to significantly faster parsers if one uses the approximate value function with search algorithms. Finally, since as far as the connection between structured prediction and MDPs is concerned, it is only the sequential nature of constructing outputs matters, one can construct significantly different output construction methods.

Chapter 2

Markovian decision problems

Markov decision processes (MDPs) are a widely studied, general mathematical framework for sequential decision problems (e.g., Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998)). The essential idea is that an agent interacts with its environment, changing the state of the environment and receiving a sequence of rewards. The goal of the agent is to maximize the cumulative (discounted) sum of the rewards received. A discounted total cost, infinite-horizon MDP is defined with a 5-tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, T, \gamma, r)$, where

- $\mathcal{X}$ is a finite set of *states*.
- $\mathcal{A}$ is a finite set of *actions*.
- T is a set of *transition probabilities*; $T(x'|x, a)$ stands for the probability of the transition from state x to x' upon taking action a ($x, x' \in \mathcal{X}, a \in \mathcal{A}$).
- $\gamma \in [0, 1)$ is called the *discount factor* of the MDP.
- r is the *reward function*; $r : \mathcal{X} \times \mathcal{A} \mapsto \mathbb{R}$. This function determines the reward upon selecting action $a \in \mathcal{A}$ at state $x \in \mathcal{X}$.

An MDP is called deterministic if all the transitions are deterministic, i.e., if for any $(x, a) \in \mathcal{X} \times \mathcal{A}$, $T(x'|x, a) = 0$ holds for all next states $x' \in \mathcal{X}$ except for one. An MDP with finite $\mathcal{X}$ and $\mathcal{A}$ is called a finite MDP. An MDP when $\mathcal{X}$ or $\mathcal{A}$ are infinite, is called an infinite MDP. In this work, we focus on the

case of finite MDPs to keep things simple. Most results can be generalized to infinite MDPs, however, this is left for future work.

A *stochastic stationary policy* (in short: policy) is a mapping $\pi : \mathcal{A} \times \mathcal{X} \to [0,1]$ satisfying $\sum_{a \in \mathcal{A}} \pi(a|x) = 1, \forall x \in \mathcal{X}$.[1] The value of $\pi(a|x)$ is the probability of taking action a in state x. For a fixed policy, the *value* of a state $x \in \mathcal{X}$ is defined by

$$V_r^\pi(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \middle| \; x_0 = x \right].$$

Here x_t is a random process with $P(x_0 = x) > 0$ that is obtained by following policy π in the MDP. This means that in each time step, x_{t+1} follows the distribution $T(\cdot|x_t, a_t)$, where a_t is a random action drawn from $\pi(\cdot|x_t)$. (The notation $\mathbb{E}_\pi$ is used to signify that the expectations is taken by assuming that the transitions are generated while following π.) The function $V_r^\pi : \mathcal{X} \to \mathbb{R}$ is called the value function corresponding to policy π and reward function r. The *expected return* of policy π is defined by

$$\bar{V}_r^\pi = \mathbb{E}\left[V_r^\pi(x) | x \sim D \right],$$

where D is a distribution from where the starting state is drawn.

A policy that maximizes the values at all states is called an *optimal policy* and is denoted by π_r. Thus, an optimal policy maximizes the cumulated total expected discounted rewards irrespective of where the process starts. The values under an optimal policy define the *optimal value function*, which also satisfies

$$V_r^*(x) = \max_\pi V_r^\pi(x), \;\; \forall x \in \mathcal{X}.$$

The *Q-factors* (or *action-value functions*) can be defined similarly:

$$Q_r^\pi(x, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \middle| \; x_0 = x, a_0 = a \right].$$

[1] Instead of $\pi(a, x)$ we use $\pi(a|x)$ to emphasize that $\pi(\cdot, x)$ is a probability distribution.

In words, the action-value of x under π is the expected total discounted reward assuming that the first action taken in $x \in \mathcal{X}$ is $a \in \mathcal{A}$, and the further actions are sampled using π. Similarly to the optimal state-values, the *optimal action-values* or *optimal Q-factors* are given by maximizing with respect to the policies:

$$Q_r^*(x,a) = \max_\pi Q_r^\pi(x,a),\ \forall (x,a) \in \mathcal{X} \times \mathcal{A}.$$

It is also useful to define *advantage functions*:

$$A_r^\pi(x,a) = Q_r^\pi(x,a) - V_r^\pi(x),\ \forall (x,a) \in \mathcal{X} \times \mathcal{A}.$$

Similarly, $A_r^* = Q_r^* - V_r^*$. In finite MDPs we can represent the mappings π, r, Q_r^π, A_r^π as matrices and V_r^π as a vector. In the future we will use this convention. An action in a state x is called optimal if it maximizes $Q_r^*(x,\cdot)$ or $A_r^*(x,\cdot)$. The next theorem concerning the *Bellman-equations* follows directly from the definition of V_r^π and Q_r^π (taken from Sutton and Barto (1998)).

Theorem 1. *The value functions and Q-factors for any policy π satisfy the Bellman equations*

$$\begin{aligned} V_r^\pi(x) &= \sum_{a \in A} \pi(a|x) \left(r(x,a) + \gamma \sum_{y \in X} P(y|x,a) V_r^\pi(y) \right), \\ Q_r^\pi(x,a) &= r(x,a) + \gamma \sum_{y \in X} P(y|x,a) V_r^\pi(y), \end{aligned}$$

$\forall x \in \mathcal{X}, a \in \mathcal{A}$.

The following result is the fundamental result of the theory of MDPs:

Theorem 2. *A policy $\pi(a|x)$ is optimal if and only if*

$$\pi(a|x) \begin{cases} \geq 0 & , \text{ if } a = \arg\max_{b \in A} Q_r^*(x,b) \\ = 0 & \text{else.} \end{cases}$$

In particular, it follows that there always exist optimal policies amongst the deterministic ones. A policy that chooses the best action with respect to

some function $Q_r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is called *greedy* with respect to Q_r. Hence, greedy policies with respect to the optimal action-value function are optimal. For a proof of these results, see e.g. Sutton and Barto (1998).

Each stationary policy induces a stationary *Markov chain* over the state space of the MDP, i.e., a random process where the probabilities of state transitions only depend on the actual state. For a short overview on Markov chains, see Appendix B.4 in Bertsekas and Tsitsiklis (1996). The *stationary distribution* over the state space generated by the optimal policy π_r will be denoted by μ_r.

One variation of MDPs that we will need is when not all actions are available in each state. In that case we use $\mathcal{A}(x)$ to denote the set of actions permitted in state x. The rest of the definitions need then to be adjusted accordingly (i.e., policies in state x cannot choose actions outside of $\mathcal{A}(x)$). Another particularly interesting version of the MDP framework is when a set of *terminal states* $\mathcal{X}_T$ is also given, and γ is set to 1. When reaching a state $x \in \mathcal{X}_T$, the random process terminates. Then the definitions of V_r^π, Q_r^π, A_r^π are changed straightforwardly by replacing the summation to infinity by summation to the (random) time step H, when the process first enters a terminal state. In such MDPs policies induce non-stationary Markov chains that have absorbing states. We define the distribution μ_r in this case with

$$\mu_r(x) = \mathbb{E}_{\pi_r}\left[\sum_{t=0}^{H} \mathbb{I}(x_t = x) \,\middle|\, x_0 \sim D\right]$$

for all $x \in \mathcal{X}$. Solving an MDP $\mathcal{M} = \{\mathcal{X}, \mathcal{A}, T, \gamma, r\}$ means finding the optimal policy π_r for it. This problem can be solved by dynamic programming techniques, when the dynamics of the MDP (i.e., T) is exactly known. However, in many practical cases, this is not true. The problem of solving MDPs under high uncertainty and large state spaces is attacked by Reinforcement Learning techniques.

Chapter 3

Inverse reinforcement learning

In this chapter we present an overview of current inverse reinforcement learning (IRL) methods. First, we give the definition of the IRL problem and discuss some of its difficulties. We then present three existing algorithms in a unified framework. The unified framework allows us to compare these algorithms and elaborate on their similarities and differences. We point out a particular shortcoming of some previous algorithms in Section 3.3. In Section 3.4 we present a novel approach to inverse reinforcement learning, partly based on our paper Neu and Szepesvári (2007). We compare the presented methods in an artificial domain in Section 3.5. (We will compare the methods in a practical domain in Chapter 4.)

Informally, IRL is a method to clone the observed behavior of an expert by finding the definition of the task that the expert performs. Assuming that the expert acts in an MDP, this can be stated more formally as finding the reward function that generates an optimal behavior that is close enough to the behavior of the expert. This definition still leaves one question open: how do we decide if two particular behaviors are "close enough"? The main difference between the algorithms to be shown is this definition of closeness (that we will call *dissimilarity* furtheron): once this definition is fixed, we are left with the task of finding an algorithm to efficiently minimize it.

The IRL problem is interesting for many reasons. In general, it is very difficult to handcraft a reward function that produces the desired behavior.

In many cases, though, samples from the desired behavior are available, which can be used to define a reward function via IRL. This reward function is not simply used to explain the behavior of the expert, or reproduce it in the situations where the expert has demonstrated his knowledge – it is constructed in a way that *generalization* of the behavior is possible. In other words, IRL is not only a useful tool to help defining reward functions, but also a method that solves the *apprenticeship learning* problem – the problem of imitating and generalizing behaviors.

Besides the dilemma of selecting an appropriate feature set we have to be aware of that the IRL problem is ill-posed: infinitely many reward functions can give rise to a specific behavior. Even worse, this set of solutions contains degenerate solutions as well, such as the reward function that is identically zero in all state-action pairs (this reward makes all policies optimal). One solution is to give preference to reward functions that robustly generate the observed behavior. An even stronger requirement would be that the observed behavior be the *only* optimal behavior with respect to the reward function. The dissimilarity functions should be chosen to encourage such solutions.

3.1 A unified approach

The purpose of this section is to present a unified framework for the design of IRL algorithms. An IRL algorithm receives a rewardless MDP $\mathcal{M} \setminus r$ (an MDP without reward function) and a list of samples, $\mathcal{D} = \{(x_t, a_t)\}_{t=1}^{N}$, representing some trajectories of an expert. Thus, here x_t is the state visited by the expert at time step t and a_t is the action selected by the expert at time step t. The task is then to come up with a reward such that the optimal policy in the given MDP follows closely the observed behavior. In order to specify what we mean by "closely" we define a dissimilarity function $J = J(r; \mathcal{D})$ that maps reward functions and datasets into reals, assigning a higher value to a pair when the optimal behavior with respect to the selected reward function is less similar to the expert's observed behavior as represented by the dataset.

Given J, a good reward function r should minimize the dissimilarity

between r and $\mathcal{D}$. Thus, one might be interested in computing

$$r^* = \arg\min_r J(r; \mathcal{D}) = ?$$

Below we will argue that most IRL algorithms aim to solve an optimization of this form, the difference being the choice of the dissimilarity. Oftentimes, in order to facilitate generalization, the reward function is sought in a parametric family of reward functions. Assuming linear parametrization we have

$$r_\theta = \sum_{i=1}^{n} \theta_i \phi_i = \theta^T \phi, \tag{3.1}$$

where ϕ is a vector-valued mapping from the complete state-action space, and is usually called the *feature vector*. Unless stated otherwise, we will only consider linear parametrizations in the following. [1] Given a parametrization the problem becomes to find a parameter vector θ^* such that $J(\theta; \mathcal{D}) = J(r_\theta; \mathcal{D})$ is minimized at θ^*.

The optimal parameter vector is typically found by incremental algorithms. We will see that these algorithms take the form

$$\theta^{(k+1)} = g(g^{-1}(\theta^{(k)}) + \alpha^{(k)} \Delta^{(k)}),$$

where $\alpha^{(k)}$ is the step-size at iteration k, g is the so-called link-function and $\Delta^{(k)}$ is the parameter update proposed by the actual IRL method. In particular, $g(x) = \exp(x)$ leads to multiplicative, $g(x) = x$ leads to additive updates. The discussion of the relative advantages of the choice of the link function is out of the scope of this thesis and the interested reader is referred to Cesa-Bianchi and Lugosi (2006) where such algorithms are extensively discussed in the on-line learning framework.

Regularization is an efficient tool that we can use to facilitate model selection. Here one switches from minimizing J to minimizing the ℓ^2-regularized

[1] Then the feature vector can be regarded as the third input for the IRL algorithm.

dissimilarity J_λ defined by

$$J_\lambda(\theta; \mathcal{D}) = J(\theta; \mathcal{D}) + \frac{1}{2}\lambda\|\theta\|^2,$$

where λ is some small positive constant. As most of the methods to be presented (except the projection algorithm) can be regarded as steepest descent methods, this regularization factor will appear as an additive term in the update steps:

$$\Delta_\lambda^{(k)} = \Delta^{(k)} - \lambda\theta^{(k)}$$

Before moving on to the particular algorithms, we need to introduce some more notations: the *conditional feature expectation function* with respect to a reward function r is defined by

$$\Phi_r(x, a) = \mathbb{E}_{\pi_r}\left[\sum_{t=0}^{\infty}\gamma^t\phi(x_t, a_t)\middle| x_0 = x, a_0 = a\right], \forall(x, a) \in \mathcal{X} \times \mathcal{A},$$

where π_r is an optimal policy w.r.t. r. The (unconditional) feature expectations are defined by taking the expectation of the condition feature expectations:

$$\bar{\Phi}_r = \mathbb{E}\left[\Phi_r(x, a)\middle| x \sim D, a \sim \pi_r(\cdot|x)\right]$$

For the sake of brevity, but at the price of slightly abusing the notation the optimal policy w.r.t. $r = r_\theta$ will be denoted by π_θ and the corresponding state visitation frequencies will be denoted by μ_θ. Similarly, V_θ, Q_θ, A_θ and Φ_θ will stand for the value function, the action-value function, the advantage function and the feature expectations generated by policy π_θ.

3.2 Previous solutions

In this section we shall examine three different algorithms from the aspects of what dissimilarity they use and what update step they have. We do not deal with the derivation of the algorithms or their convergence properties.

3.2.1 Projection

The projection algorithm was proposed in Abbeel and Ng (2004). This is the earliest IRL algorithm discussed in this thesis.[2]

Dissimilarity

Behaviors are represented here with their generated feature expectations, and the selected dissimilarity is the ℓ^2-metric on $\mathbb{R}^n$.

The training samples are needed in the form of N_{traj} trajectories, the length of trajectory i is l_i. The estimate of the expert's feature expectation vector is then

$$\bar{\Phi}_E := \frac{1}{N_{traj}} \sum_{i=0}^{N_{traj}} \sum_{j=0}^{l_i} \gamma^j \phi(x_{t_{i,j}}, a_{t_{i,j}}),$$

where $t_{i,j} = j + \sum_{k=0}^{i-1} l^{(k)}$.

Using this notation the dissimilarity is

$$J(\theta; \mathcal{D}) = \|\bar{\Phi}_\theta - \bar{\Phi}_E\|^2. \tag{3.2}$$

Update step

The parameter updates are done additively at each step ($g(x) = x$), the value of the update at the k-th step is

$$\Delta^{(k)} = \beta^{(k)}(\bar{\Phi}_E - \bar{\Phi}_{\theta^{(k)}}) - \beta^{(k)}\theta^{(k)} \tag{3.3}$$

where we use a special step size parameter, $\beta^{(k)}$ (and the global step size parameter $\alpha^{(k)}$ is kept constant 1). To compute this step size, we need to maintain a vector $\Psi^{(k)}$ throughout the training steps. By setting $\Psi_0 = \bar{\Phi}_{\theta_0}$, the values of $\beta^{(k)}$ and $\Psi^{(k)}$ are computed for all following steps incrementally

[2]The term "inverse reinforcement learning" was first used by Ng and Russell (2000), but that paper did not present a practical IRL algorithm, its aim is mainly to characterize the solution set for the IRL problem.

in the following way:

$$\beta^{(k)} = \frac{\left(\bar{\Phi}_{\theta^{(k)}} - \Psi^{(k-1)}\right)^T \left(\bar{\Phi}_E - \Psi^{(k-1)}\right)}{\left(\bar{\Phi}_{\theta^{(k)}} - \Psi^{(k-1)}\right)^T \left(\bar{\Phi}_{\theta^{(k)}} - \Psi^{(k-1)}\right)} \tag{3.4}$$

$$\Psi^{(k)} = \Psi^{(k-1)} + \beta^{(k)} \left(\bar{\Phi}_{\theta^{(k)}} - \Psi^{(k-1)}\right) \tag{3.5}$$

The original algorithm includes a post-processing step when a mixed policy is constructed that produces feature expectations with minimal distance to the expert's observed feature expectations. As in many practical cases we are only interested in optimal policies that are deterministic, we omit this step.

3.2.2 MWAL

This *multiplicative weights algorithm for apprenticeship learning* (MWAL) was proposed in Syed and Schaphire (2007). The authors follow a game-theoretic approach, aiming to improve the projection algorithm of Abbeel and Ng (2004).

Dissimilarity

In this case the dissimilarity is

$$J(\theta; \mathcal{D}) = \theta^T(\bar{\Phi}_\theta - \bar{\Phi}_E), \tag{3.6}$$

where θ is restricted to nonnegative values, corresponding to the assumption that the features are positively correlated with the rewards.[3]

The rationale underlying this criterion is that $\theta^T \bar{\Phi}_{r_\theta}$ can be shown to be the average expected discounted reward under the optimal policy corresponding to r_θ and if the states are initialized from the distribution $\mathcal{D}$. Further, $\theta^T \bar{\Phi}_E$, where $\bar{\Phi}_E$ represents the empirical feature expectations (measured using the data D) is an approximation to the average reward of the expert. The minimization problem corresponds to a robust (minimax) approach: The

[3] Abbeel and Ng (2004) also propose a max-margin algorithm that targets to minimize the same criterion.

optimal choice of θ makes the performance of the optimal policy the least favorable compared with that of the expert. Syed and Schaphire (2007) show that the value of J at the minimum is positive. Due to von Neumann's min-max theorem they also show that the found behavior will be better than the expert's behavior even when the least favorable parameters are taken. This buys an extra robustness to the algorithm.

Update step

The updates proposed to solve this optimization are multiplicative, i.e., $g(x) = \exp(x)$. Further,

$$\Delta^{(k)} = \bar{\Phi}_E - \bar{\Phi}_{\theta^{(k)}}. \tag{3.7}$$

As it is well-known, multiplicative weights algorithms can be sensitive to step-sizes. In this work we will use several choices, but do not follow the theoretically motivated step-size sequences of Syed and Schaphire (2007) as these are known to yield suboptimal results.

3.2.3 Max margin planning

This algorithm was published in Ratliff et al. (2006) and is derived from the maximum margin algorithm of Taskar et al. (2005). Although Ratliff et al. (2006) argue that their problem is distinct from IRL since it concerns a series of planning problems, by assuming that the state spaces are disjoint we can take the union of these state spaces and define a single planning problem over the joined state space.

Dissimilarity

Let the state-action visitation frequencies, μ_E, and a loss function, $\ell : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^+$, be defined as follows:

$$\mu_E(x,a) := \frac{\sum_{t=1}^{N} \mathbb{I}(x_t = x \wedge a_t = a)}{N}, \quad \forall (x,a) \in \mathcal{X} \times \mathcal{A}$$
$$\ell(x,a) := c_\ell \mu_E(x,a), \quad \forall (x,a) \in \mathcal{X} \times \mathcal{A}$$

In the above formula, c_ℓ is a positive *loss constant*, a parameter of the algorithm. Then the dissimilarity is defined as

$$J(\theta; \mathcal{D}) = \left(\sum_{x,a} (r_\theta(x,a) - \ell(x,a)) \mu_{\theta,\ell}(x,a) - \sum_{x,a} r_\theta(x,a) \mu_E(x,a) \right) + \frac{\lambda}{2} \|\theta\|^2 ,$$

where $\mu_{\theta,\ell}$ is the stationary distribution (visitation frequencies/counts in episodic MDPs) generated by the policy that is optimal w.r.t. $r_\theta - \ell$, and $\lambda \geq 0$ is a regularization constant that can be used for complexity reguralization. The role of the loss function is to enforce that the solution found is better (cf. Equation 3.6) than other solutions by at least a margin proportional to this loss. Accordingly, here the average expected payoff of the optimal policy corresponding to $r_\theta - \ell$ (and not to r_θ) is compared with the average payoff of the expert. By choosing the loss proportional to the state-visitation frequencies we force rewards of highly visited state-action pairs to take on larger values, encouraging the learnt policy to visit such states. This also has an effect of enforcing meaningful solutions to the IRL problem, as the degenerate solution $\theta = 0$ does not minimize the criterion.

Update step

The update of the subgradient algorithm of Ratliff et al. (2006) uses $g(x) = x$ and

$$\Delta^{(k)} = \sum_{x,a} \phi(x,a)(\mu_E - \mu_{\theta^{(k)},\ell}) - \lambda\theta^{(k)} = \bar{\Phi}_E - \bar{\Phi}_{r_{\theta^{(k)}} - \ell} - \lambda\theta^{(k)}. \qquad (3.8)$$

This algorithm connects the previous two in some sense. It can be proved that $\sum_{x,a} r_\theta(x,a)\mu_\theta(x,a) = (1-\gamma)\mathbb{E}[V_\theta(x)|x \sim D]$, see e.g. Theorem 1 in Baxter and Bartlett (1999). When setting $\ell_E = 0$ and $\lambda = 0$, the dissimilarity is the same as in the MWAL method up to a multiplicative constant, as $\sum_{x,a} r_\theta(x,a)\mu_\theta(x,a) = \theta^T \sum_{x,a} \phi_{x,a}\mu_\theta(x,a) = (1-\gamma)\theta^T\bar{\Phi}_\theta$. Then the update of the max-margin algorithm becomes the same as the perceptron update of Freund and Schapire (1999). If we apply these updates multiplicatively,

and assume that the signs of the optimal parameters are known, we get the MWAL algorithm. Furthermore, if we set $\lambda = 1$, $\ell = 0$, and use the special step sizes computed using Equations 3.4 and 3.5, we get the update step of the projection algorithm.

3.3 The feature rescaling problem

Our main concern in this section is the algorithm of Abbeel and Ng (2004). This algorithm returns policies that come with the guarantee that their average total discounted reward computed from the expert's *unknown* reward function is in the ε-vicinity of the expert's performance. We claim that this guarantee will be met only when the scaling of the features in the method and the "true" scaling match each other. Actually, this observation led us to the algorithm to be proposed in the next section.

A major underlying hidden assumption (implicit in the formalism of Abbeel and Ng (2004)) is that the scaling of the features is known. To see this assume that the number of features is 2, the optimal parameters are $\theta_1^* = \theta_2^* = \sqrt{2}/2$, $\|\bar{\Phi}_\theta - \bar{\Phi}_E\|_2 \leq \varepsilon$ and in particular $\bar{\Phi}_{E,1} = 0$, $\bar{\Phi}_{E,2} > 0$, $\bar{\Phi}_{\theta,1} = -\varepsilon$, $\bar{\Phi}_{\theta,2} = 0$. Further, assume that the features are rescaled by $\lambda = (\lambda_1, \lambda_2)$. In the new scale the expert's performance is $\bar{V}_E(\lambda) = \sqrt{2}/2\lambda^T\phi_E$ and π's performance is $\bar{V}_\theta(\lambda) = \sqrt{2}/2\lambda^T\bar{\Phi}_\theta = \sqrt{2}/2(\lambda^T\phi_E - \lambda_1\varepsilon)$. A natural requirement is that for any scaling λ, $\bar{V}_\theta(\lambda)/\bar{V}_E(\lambda)$ should be lower bound by a positive number (or rather a number close to $1-\varepsilon$). By straightforward calculations, $\bar{V}_\pi(\lambda)/\bar{V}_E(\lambda) = 1 - (\lambda_1/\lambda_2)\varepsilon/\phi_{E,2} \to -\infty$, hence although $\|\phi_\pi - \phi_E\|_2 \leq \varepsilon$, the actual performance of π can be quite far from the performance of the expert if the scaling of the features does not match the scaling used in the algorithm.

3.4 A novel approach

In this section we present a novel approach to IRL, which is based on minimizing the mismatch to the expert's policy. We can define different dissimilarity

functions for policies, such as the ℓ^2-distance, the Kulback-Liebler (KL) divergence or the Rényi divergence. The advantage of using the ℓ^2-metric is that the problem simplifies in this case to a quadratic optimization problem.

3.4.1 QP formulation

Here, we present our algorithms for a weighted ℓ^2-distance. The weighted ℓ^2-distance of policies π and π_E is defined as

$$\|\pi - \pi_E\|_\mu = \sum_{x \in \mathcal{X}} \mu(x) \sum_{a \in \mathcal{A}} (\pi - \pi_E)^2,$$

where μ is some distribution over $\mathcal{X}$. A suitable choice for μ is the estimate of the stationary distribution generated by the expert, which we will denote with μ_E. This weighting expresses our belief that the states that are most frequently visited by the expert have a key role in the task that the expert performs. Initially, we assume that $\mu_E(x) > 0$ for all $x \in \mathcal{X}$.

The problem then can be stated as the quadratically constrained quadratic program

$$\min_{\pi,\theta} \frac{1}{2} \|\pi - \pi_E\|^2_{\mu_E} \tag{3.9}$$

subject to:

$$\begin{aligned}
Q(x,a) &= \theta^T \phi(x,a) + \gamma P_{x,a} V && \forall x, a && (3.10)\\
V(x) &= \textstyle\sum_a \pi(a|x) Q(x,a) && \forall x, a && (3.11)\\
\textstyle\sum_a \pi(x|a) &= 1 && \forall x && (3.12)\\
\pi(x|a) &\in [0,1] && \forall x, a && (3.13)
\end{aligned}$$

Constraint 3.10 is linear, thus causes no problems. The problem is that constraint (3.11) is quadratic. If we assume that the policy π is deterministic, that is, we replace constraint (3.13) by $\pi(a|x) \in \{0,1\}$, this can be overcome by the so-called big-M method. Using this trick, constraint (3.11) can be

rewritten as the following two linear constraints:

$$V(x) \geq -M(1-\pi(a|x)) + Q(x,a) \qquad \forall x, a \tag{3.14}$$

$$V(x) \leq M(1-\pi(a|x)) + Q(x,a) \qquad \forall x, a, \tag{3.15}$$

where M is some large positive constant. It is easy to see that these two constraints are satisfied if and only if constraint (3.11) is satisfied. This is a *mixed integer quadratic problem* (MIQP). It is known that solving a mixed integer problem can be NP-hard in the worst case, so we present a relaxation of the problem in the following.

Proposition 1. *Assume that the policy of the expert is deterministic and the reward function of the expert is in the span of the features. Then the solution of the following quadratic program is in the feasible set determined by constraints (3.10)–(3.13):*

$$\min_{\pi,\theta} \frac{1}{2}\|\pi - \pi_E\|^2_{\mu_E} \tag{3.16}$$

subject to:

$$V(x) \geq -M(1-\pi(a|x)) + Q(x,a) \qquad \forall x, a \tag{3.17}$$

$$V(x) \leq M(1-\pi(a|x)) + Q(x,a) \qquad \forall x, a \tag{3.18}$$

$$\textstyle\sum_a \pi(x|a) = 1 \qquad \forall x \tag{3.19}$$

$$\pi(x|a) \in [0,1] \qquad \forall x, a \tag{3.20}$$

Proof. If the criterion $\frac{1}{2}\|\pi - \pi_E\|^2_{\mu_E}$ can be forced to be zero while satisfying the above constraints, then π is also deterministic. This means that the transformed constraints (3.17) and (3.18) are equivalent to constraint (3.11). Hence, the found solution lies in the feasible set described by the original constraint. All we need to show is that the criterion can be forced to be zero. As the true reward function of the expert lies in the span of the features, the expert's policy can be fully reconstructed by some parameter vector. □

This means that we have successfully reduced our problem to minimizing

a continuous, convex quadratic function on a convex domain, which is known to be tractable, see e.g. **?**. We need to have a closer look at the case when the expert's reward function lies outside of the region that can be reached by our parametrization.

Let us fix a state x. It is clear that if we represent $\pi(\cdot|x)$ with an $|\mathcal{A}|$-dimension vector, than the possible values of this vector lie on the surface on the standard $|\mathcal{A}|$-simplex. This simplex has $|\mathcal{A}|$ corners, each of them representing a deterministic choice of actions, each of them lies to the same distance from the others. If at this fixed state $\frac{1}{2}\|\pi(\cdot|x) - \pi(\cdot|x)\|^2$ cannot be made zero, we are left with the following question: is the next closest policy deterministic? The answer is negative: the next closest policy is represented by the point on the surface of the $|\mathcal{A}|-1$ dimension simplex that is closest to the origin—which is a stochastic one. To be more exact, it gives probability $\frac{1}{|\mathcal{A}|-1}$ to all the actions that can be reached under the actual parametrization. Which means that the constraint transformation is *invalid* in the case when there is no parameter vector that can perfectly explain the expert's behavior.

Another important case is when there are states where $\mu(x) = 0$, which means that minimizing the criterion does not enforce the policy π to be completely deterministic. Again, this invalidates the constraint transformation.

We have to conclude that linearizing the constraints can only be done in the impractical case when the expert's true reward function lies in the span of our features and when we have an example of the expert's behavior in all of the states.

3.4.2 A subgradient method

This algorithm (first published in Neu and Szepesvári (2007)) aims to solve the optimization problem with nonlinear constraints using a subgradient method (Shor et al., 1985). We satisfy constraints 3.10 and 3.11 by explicitly computing the optimal policy π_θ w.r.t. r_θ, and compute the subgradient of the criterion. Using our unified notation, the dissimilarity metric of our algorithm will be

$$J(\theta; \mathcal{D}) = \frac{1}{2}\|\pi_\theta - \pi_E\|^2_{\mu_E}, \tag{3.21}$$

and the update step will be

$$\Delta^{(k)} = -\nabla_\theta J(\theta^{(k)}; \mathcal{D}),$$

i.e., a subgradient of J w.r.t. θ. By using the chain rule, we get that $\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial \pi_\theta}\frac{\partial \pi_\theta}{\partial \theta}$, thus we are left with calculating the gradient of π_θ.

It is clear that if π_θ is a greedy policy with respect to Q_θ, than it is piecewise constant because of the involved arg max function, hence $\frac{\partial \pi_\theta}{\partial \theta} = 0$ almost everywhere. Therefore we select a smooth mapping $B(Q_\theta)$ that returns near-optimal policies with respect to its argument. One possibility is to use *Boltzmann-policies*, i.e.

$$B(Q)(a|x) = \frac{\exp\left(\frac{1}{\eta}Q(x,a)\right)}{\sum_{b\in\mathcal{A}}\exp\left(\frac{1}{\eta}Q(x,b)\right)},$$

where $\eta > 0$ is a temperature parameter that controls how close $B(Q)$ is to greedy action selection. With this choice

$$\begin{aligned}\frac{\partial \pi_\theta}{\partial \theta_i}(a|x) &= \pi_\theta(a|x)\frac{\partial \ln[\pi_\theta(a|x)]}{\partial \theta_i} \\ &= \pi_\theta(a|x)\frac{1}{\eta}\left(\frac{\partial Q^*_\theta(x,a)}{\partial \theta_i} - \sum_{b\in\mathcal{A}}\pi_\theta(b|x)\frac{\partial Q^*_\theta(x,b)}{\partial \theta_i}\right).\end{aligned} \tag{3.22}$$

Hence, all that remains is calculating $\partial Q^*_\theta(x,a)/\partial\theta_i$. We have shown that these derivatives can be calculated almost everywhere in the parameter space by solving some fixed-point equations similar to the Bellman-optimality equations. The following proposition holds:

Proposition 2. *Assume that the reward function r_θ is linearly parametrized by θ with uniformly bounded features: $\sup_{(\theta,x,a)\in\mathbb{R}^n\times\mathcal{X}\times\mathcal{A}}\|\phi(x,a)\| < +\infty$. The following statements hold:*

(1) *Q^*_θ is uniformly Lipschitz-continuous as a function of θ in the sense that for any (x,a) pair, $\theta,\theta' \in \mathbb{R}^n$, $|Q^*_\theta(x,a) - Q^*_{\theta'}(x,a)| \le L'\|\theta - \theta'\|$ with some $L' > 0$;*

(2) *Except on a set of measure zero, the gradient, $\nabla_\theta Q^*_\theta$, is given by the solution of the following fixed-point equation:*

$$\Phi_\theta(x,a) = \phi(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) \sum_{b \in \mathcal{A}} \pi(b|y) \Phi_\theta(y,b),$$

where π is any policy that is greedy with respect to Q_θ.

For a detailed proof of a more general version of this proposition, see Neu and Szepesvári (2007). In fact, Equation (3.23) can be solved componentwise: the i-th component of the derivative can be obtained computing the action-value function for the policy π using the i-th component of the feature vector in place of the reward function. This means that if we want to use our favorite value-based reinforcement learning algorithm for finding an optimal policy for the actual reward function, we can compute the conditional feature expectations function without extra effort. The only change we need to make is to maintain an extra "action-value function" for each feature, and update these extra functions similarly to the actual action values. The convergence results for computing the conditional feature expectations vector carry through from the case of computing the action values.

As an example, if we use Sarsa (Sutton and Barto, 1998) to learn the action-value function, then the proposed update step at time step t is

$$Q^\pi_r(x_t,a_t) = Q^\pi_r(x_t,a_t) + \alpha_t \Big(r(x_{t+1},a_{t+1}) + \gamma Q^\pi_r(x_{t+1},a_{t+1}) - Q^\pi_r(x_t,a_t) \Big).$$

Then the i-th component of the feature expectations vector is updated as follows:

$$\Phi^\pi_i(x_t,a_t) = \Phi^\pi_i(x_t,a_t) + \alpha_t \Big(\phi_i(x_{t+1},a_{t+1}) + \gamma \Phi^\pi_i(x_{t+1},a_{t+1}) - \Phi^\pi_i(x_t,a_t) \Big)$$

For certain problems, the computation of the optimal advantage function A^*_r is easier than that of the optimal action values. In these cases, we can use a Boltzmann policy with respect to A^*_r, as $B(Q^*_r) = B(A^*_r)$ holds by elementary arguments. In such problems, computing $\Phi_\theta(x,a) - \sum_{b \in \mathcal{A}} \pi_\theta(a|x) \Phi_\theta(x,b)$ can also be easier for all state-action pairs, than computing $\Phi_\theta(x,a)$, which is

clearly an advantage if we want to compute the gradient as in Equation (3.22).

Putting all this together, we get the subgradient of the dissimilarity metric:

$$\nabla_\theta J(\theta^{(k)};\mathcal{D}) = \sum_{x\in\mathcal{X}} \mu(x) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_{\theta^{(k)}}(a|x)(\pi_{\theta^{(k)}}(a|x) - \pi_E(a|x)), \qquad (3.23)$$

where $\nabla_\theta \pi_{\theta^{(k)}}(a|x)$ is computed using Equation (3.22).[4]

Our original aim with constructing this new algorithm was to create a method that is insensitive to the scaling of the features. In order to achieve this, we have used an extension of the *natural gradient* technique (Amari, 1998) to the subdifferentials of nonconvex functions. For a detailed explanation of the idea, the reader is referred to the paper Neu and Szepesvári (2007).

3.5 Experimental comparison

In this section, we compare the presented methods in an artificial grid world MDP, as in Neu and Szepesvári (2007). We used a 10×10 grid world, where each state is a grid square and the four actions correspond to moves in the four compass directions with 70% probability of success. If an action is unsuccessful, the agent is randomly moved to one of the other neighboring grid squares. We constructed the reward function as a linear combination of 10 features ($\phi_i : \mathcal{X} \to [-0.5; 0.5]$, $i = 1, \ldots, 10$). The features were essentially randomly constructed and dense (i.e., nonzero in most states). The optimal parameter vector θ_* consists of evenly distributed random values from $[0, 1]$. In general we try to approximate the reward function with the use of the same set of features that has been used to construct it, but we also examine the situation of unprecisely known features. Value iteration was used for finding the optimal policy (or gradients) in all cases.

[4]Using the KL-divergence as dissimilarity metric between the policies, $\pi_{\theta^{(k)}} - \pi_E$ is replaced in Equation (3.23) by $-\pi_E(a|x)/\pi_{\theta^{(k)}}(a|x)$.

The performance is measured with the relative loss, which is defined by:

$$\frac{\bar{V}_{r^*}^{\pi_E} - \bar{V}_{r^*}^{\pi_\theta}}{\bar{V}_{r^*}^{\pi_E}},$$

where $\bar{V}_{r^*}^{\pi_E}$ is the expected return of the expert's policy with respect to the expert's real reward function r^*, and similarly $\bar{V}_{r^*}^{\pi_\theta}$ is the expected return of the policy π_θ returned by the algorithm with respect to the same reward function.

Unless otherwise stated the data consists of 100 independent trajectories following the optimal policy with respect to the real reward function, each having a length of 100 steps. The step sizes were constant for all algorithms except projection, which has a special step size parameter. If not stated otherwise, the regularization parameter is set to 1 for max margin planning, and 0 for all other methods. The maximal number of iterations is kept at 100. All results are averages of 25 runs, and we also show the standard error of the performance.

For all methods, we selected the best solution with respect to its own dissimilarity function, e.g., the best solution given by policy matching minimizes the dissimilarity (3.21), and the one given by projection minimizes the dissimilarity (3.2). For the MWAL algorithm, information on the sign of the optimal parameter vector was always given, while all the other methods were initialized with all zero parameter vectors.

We examined the algorithms' behavior when *(i)* the number of the training samples was varied (Figure 3.1), *(ii)* the features were linearly transformed (Figure 3.2, Table 3.1), *(iii)* the features were perturbed (Figure 3.3), and when *(iv)* irrelevant features were added to the original feature set (Figure 3.4).

Our results show that versions of policy matching produced the best results when using the standard features for all investigated sample sizes. The difference between the performances of projection, MWAL and max margin are insignificant, however, max margin seems to perform better than the other two for smaller sample sizes. Perhaps surprisingly, policy matching using plain gradients outperforms the natural gradient and Rprop versions

Figure 3.1: Relative loss versus the number of observed expert trajectories with 1 s.e. error bars. Note logarithmic scale on both axes.

in this setting. In the case of using 100 expert trajectories, the difference between the performance of policy matching and the other methods becomes very large. We wanted to examine whether this outstanding result is robust with respect to the selection of the features, thus we have used a fixed sample set of this size for our following experiments. The results given by the different methods for this sample set are the following:

PM (policy matching): $2.18 \cdot 10^{-11}$

PM + N (policy matching using natural gradients): $2.18 \cdot 10^{-11}$

PM + R (policy matching using Rprop): 0

MM (max margin): $1.23 \cdot 10^{-2}$

PR (projection): $1.84 \cdot 10^{-3}$

MWAL : $1.10 \cdot 10^{-3}$

This sample set is somewhat better than the average sample sets of this size, as all of the algorithms reach slightly better solutions. Rprop performs remarkably well for this training set: it attains a performance that is exactly as good as that of the expert.

Figure 3.2: Relative loss versus the number of iterations with 1/5 s.e. error bars, under random rescaling of the features.

First, we have applied nonsingular random linear transformations to the features, so that the true reward function remained in the span of the features. The errors versus the number of iterations are plotted on Figure 3.2, the numerical results are shown on the second part of Table 3.1. As we expected, we have seen that the performance of the natural gradient version of our policy matching algorithm remained nearly invariant to this rescaling, while the performance of other methods significantly dropped. This means that our efforts to construct an IRL algorithm that does not suffer from the rescaling problem described in Section 3.3 were successful. All the other methods suffer badly under these adverse conditions.

In practice, it is not realistic to assume that a subspace containing the

	Original		Transformed	
	Mean	S.E.	Mean	S.E.
PM	$8.49 \cdot 10^{-8}$	$2.35 \cdot 10^{-8}$	$2.15 \cdot 10^{-1}$	$7.11 \cdot 10^{-2}$
PM + N	$4.23 \cdot 10^{-7}$	$3.65 \cdot 10^{-7}$	$\mathbf{1.05 \cdot 10^{-11}}$	$\mathbf{2.23 \cdot 10^{-12}}$
PM + R	$\mathbf{4.51 \cdot 10^{-8}}$	$\mathbf{1.52 \cdot 10^{-8}}$	$6.96 \cdot 10^{-2}$	$4.07 \cdot 10^{-2}$
MM	$2.21 \cdot 10^{-2}$	$1.12 \cdot 10^{-3}$	$2.49 \cdot 10^{-1}$	$8.21 \cdot 10^{-2}$
PR	$3.14 \cdot 10^{-2}$	$9.27 \cdot 10^{-3}$	$3.44 \cdot 10^{-2}$	$1.85 \cdot 10^{-2}$
MWAL	$3.21 \cdot 10^{-2}$	$8.57 \cdot 10^{-3}$	$3.83 \cdot 10^{-1}$	$9.59 \cdot 10^{-2}$

Table 3.1: Means and standard error of relative errors in performance. The section marked "original" gives results for the original features, the section marked "transformed" gives results when features are linearly transformed.

reward function is known. To test how the algorithms behave without this assumption we perturbed the features by adding uniform, zero-mean random numbers to them. We have varied the magnitude of the perturbation between 0.02 and 5 to see how the algorithms scale with it (a magnitude of 5 means that the random numbers perturbing the features come from the interval $[-2.5; 2.5]$ – remember that the features take their values from $[-1; 1]$). The results are plotted on Figure 3.3. As predicted in Section3.4, using these distorted features affect policy matching negatively: if the reward function of the expert does not lie in the span of the features, then a completely random policy minimizes the criterion in some states. The methods that are trying to match the feature expectations are less sensitive to this perturbation, as the zero-mean random numbers appear as zero-mean bias in the estimated feature expectations. Note that all against these arguments, the leading place of policy matching is only lost at the largest tested perturbation.

Finally, we have investigated the case of adding irrelevant features to the used feature set. For this reason, we doubled the size of the feature set by adding some randomly constructed features (similar to the original ones) with their respective parameters set to zero. We expected to see the methods trying to match the expert's feature expectations to perform somewhat better, as more aspects of the expert's behavior can be captured by these augmented feature expectations. On the other hand, this extra information

Figure 3.3: Relative loss versus the magnitude of feature perturbation with 1 s.e. error bars.

might also confuse the learning algorithm, as these new aspects might have very little to do with the task that the expert is performing. We have examined the performance of the methods when adding 5, 10, 20, and 50 irrelevant features to the feature set. Our experiments have shown that these methods are indeed helped by adding irrelevant features, but policy matching reacted negatively to this disturbance. The problem caused by irrelevant features can be alleviated using regularization. We have tested the algorithms with different settings of the regularization constant λ, the relative losses against the value of the regularization constant are plotted on Figure 3.5. We see that certain values of λ yield improved performance, however, these values are different for all of the algorithms.

Figure 3.4: Relative loss versus the number of added irrelevant features with 1 s.e. error bars.

Figure 3.5: Relative loss versus λ with 1 s.e. error bars.

Chapter 4

Application to parser training

In this part of the thesis we focus on the problem of sentence parsing. After informally describing the sentence parsing task, we present the mathematical framework for it in Section 4.1. Afterward, we present the idea of formalizing the parsing problem as a Markovian decision process in Section 4.2. We derive parser training methods from the previously presented IRL algorithms in Section 4.3, and present the results of empirical comparison of these algorithms in Section 4.4.

In parsing one is given a sequence of words forming a sentence, and the task is to determine the roles of the words within the sentence and their interrelationships. A subset of words can form grammatical structures called *constituents*, which can form further constituents and finally a grammatically correct sentence. Ambiguity can arise on more levels: trivially on the word-level, on the constituent-level, and also on the sentence-level. In some languages such as English, Chinese or Spanish, the word order in constituents and the order of constituents in sentences is fixed, thus some ambiguity can be filtered out. Computerized processing of these languages largely benefit from the theory of generative grammars introduced by Chomsky (1956). Approximating the grammar of such languages with a *context-free grammar* can lead to surprisingly good results. Using this formalism, the syntactical structure of a sentence can be represented as a *parse tree*. Figure 4.1 shows the parse tree for the sentence "It was love at first sight".

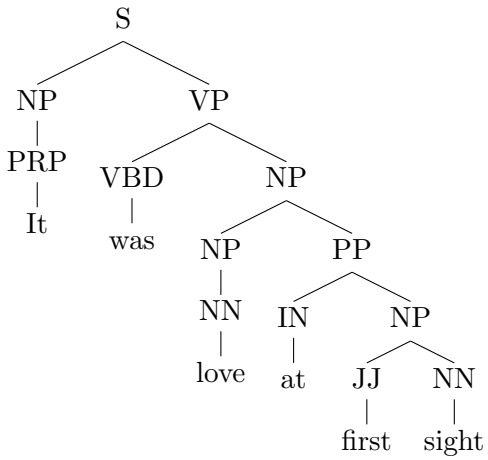

Figure 4.1: Parse tree for the sentence "It was love at first sight".

The flexibility of the context-free model can be substantially increased by using one of its stochastic extensions. *Probabilistic context-free grammars* are the most common of these extensions. In a PCFG one assigns "costs" to each of the production rules and then the task during parsing becomes that of finding a sequence of productions with the minimal total cost. The costs are often derived from a hand-annotated corpus of parsed sentences via simple statistical methods such as computation of relative frequencies of the particular productions. This may work fine for actual PCFGs, but we have to keep in mind that natural languages are not one of these by any means! Thus, there is no guarantee that a theoretical result on PCFGs will carry through natural languages. This brings us to the statement that classical generative parameter-tuning methods for PCFGs do not necessarily give rise to the best parsing results.

In the following we investigate whether it is possible to construct an improved parser training algorithm by using inverse reinforcement learning to train the cost map of a PCFG. The main idea of this work is that the parsing problem is posed as a problem of finding a shortest path from an initial state to a terminal state with unknown costs and the training examples are considered as observed traces of an optimal controller. This allows us to apply IRL techniques for learning the costs.

4.1 PCFG parsing

In this section we present the formalism that we use in parsing with probabilistic context-free grammars (PCFGs). The material presented here is based on Manning and Schütze (1999).

A PCFG is a 5-tuple $G = (\mathcal{W}, \mathcal{N}, S, \mathcal{R}, \mathcal{P})$, where

$\mathcal{W} = \{w_i\}_{i=1}^{n_t}$ is a *terminal vocabulary* (i.e. a set of terminal symbols).

$\mathcal{N} = \{N_i\}_{i=1}^{n_{nt}}$ is a *nonterminal vocabulary* (i.e. a set of nonterminal symbols).

$S \in \mathcal{N}$ is a *start symbol*.

$\mathcal{R} = \{R_i : N_{j_i} \rightarrow \xi_i\}_{i=1}^{n_R}$ is a set of *production rules*, where ξ_i is a string of terminals and nonterminals and $N_{j_i} \in \mathcal{N}$ is some nonterminal.

$\mathcal{P} : \mathcal{R} \rightarrow [0, 1]$ satisfies

$$\sum_{\{R \in \mathcal{R} : \mathrm{lhs}(R) = N\}} \mathcal{P}(R) = 1, \ \forall N \in \mathcal{N}.$$

From now on we will focus on grammars that are in Chomsky normal form (CNF), i.e. all rules are either of the form $R : N \rightarrow N_{left}\ N_{right}$ (N_{left}, $N_{right} \in \mathcal{N}$) or of the form $R : N \rightarrow w$ ($w \in \mathcal{W}$).

A PCFG generates an infinite set of *parse trees* or *parses* that we denote by $\mathcal{T}$. A parse tree $\tau = \{c_i\}_{i=1}^{n_\tau}$ is a set of *constituents*, i.e. triples of form $c = (N_c, start_c, end_c) \in \mathcal{N} \times \mathbb{N} \times \mathbb{N}$ (where $start_c$ and end_c are the respective indices of the first and last words forming the constituent) that meet the following requirements:

1. The *root* constituent, $(S, 1, n)$, is in the parse tree, and there is no constituent c in the tree with $end_c > n$.

2. The tree is linearly ordered, binary and consistent, i.e., for every $(N, i, k) \in \tau$, $i \neq k$, there is exactly one j, N_{left}, N_{right} and R such that $i \leq j < k$ and $(N_{left}, i, j) \in \tau$ and $(N_{right}, j{+}1, k) \in \tau$ and $R : N \rightarrow N_{left}\ N_{right} \in \mathcal{R}$.

3. For all i $(1 \leq i \leq n)$ a constituent of the form (w_i, i, i) is in the parse tree, where $w_i \in \mathcal{W}$.

A tree is called a *partial parse tree* if it satisfies all the previous properties except item 3. A constituent c is called *unexpanded* if $end_c \neq start_c$, and there is no constituent c' for which $start_{c'} = start_c$ and $end_{c'} < end_c$. If $end_c = start_c$, then the constituent c is called a *tagged word* constituent.

Each parse tree $\tau \in \mathcal{T}$ represents the grammatical structure of a valid sentence in the language generated by the grammar. This sentence is called the *yield* of the tree and will be denoted by y_τ. We will use the notation w_{ab} for the sequence of words $w_a\, w_{a+1}\, \ldots\, w_b$.

A PCFG defines a probability distribution p over all generated parse trees:

$$p(\tau|G) = \prod_{R \in \mathcal{R}} \mathcal{P}(R)^{f(R,\tau)}, \tag{4.1}$$

where $f(R, \tau)$ is the number of occurrences of rule R in parse tree τ. This way we can also define the probability of a sentence $s = w_{1m}$ as

$$p(s|G) = \sum_{\{\tau \in \mathcal{T} : y_\tau = s\}} p(\tau|G) = \sum_{\tau \in \mathcal{T}} p(\tau, s|G),$$

where $p(\tau, s|G) = \mathbb{I}(y_\tau = s)p(\tau|G)$. The probability that τ is a correct parse for sentence s is given by:

$$p(\tau|s, G) = \frac{p(\tau, s|G)}{p(s|G)}$$

The problem of parsing is to find the most probable parse for a sentence s, given a grammar G. This can be formalized as finding the parse

$$\begin{aligned} \tau^* &= \arg\max_{\tau \in \mathcal{T}} p(\tau|s, G) \\ &= \arg\max_{\tau \in \mathcal{T}} \frac{p(\tau, s|G)}{p(s|G)} = \arg\max_{\tau \in \mathcal{T}} p(\tau, s|G), \end{aligned}$$

which can be further written as

$$\begin{aligned}\tau^* &= \arg\max_{\tau\in\mathcal{T}} \left\{ \mathbb{I}(y_\tau = s) \prod_{R\in\mathcal{R}} \mathcal{P}(R)^{f(R,\tau)} \right\} \\ &= \arg\max_{\tau\in\mathcal{T}} \left\{ \log \mathbb{I}(y_\tau = s) + \sum_{R\in\mathcal{R}} f(R,\tau) \log \mathcal{P}(R) \right\}.\end{aligned} \tag{4.2}$$

The log probabilities of the rules introduced in the above equation are usually called the *scores* of the rules. We will denote the score of rule R by $\sigma(R)$. Then, the maximization problem can be interpreted as finding the parse tree yielding s with maximum total score. The tree τ^* is called the *maximum scoring tree* or the *candidate parse*.

4.2 PCFG parsing as an MDP

The purpose of this section is to present the reduction of parsing to solving a deterministic, episodic Markovian decision process. The idea of formulating the parsing problem as a sequential decision problem has already been discussed before in e.g. Turian and Melamed (2006) propose a method that builds parse trees sequentially in a bottom-up fashion, while Collins and Roark (2004) considers a setting where parse trees are built from left to right. The PhD thesis Daumé III (2006) (and the unpublished work of the same author) presents the more general idea of producing structured outputs by making sequential decisions by decomposing the structured outputs to variable length vectors.

In fact, solving a deterministic, episodic MDP from a given initial state is equivalent to finding a path that connects the initial state to a terminal state such that the total reward along the path is maximal. The connection to PCFG parsing then is that there the aim is to construct a parse with maximal total score, where the scores of the individual rules are additively combined. The idea is that this parse tree can be constructed in a sequential manner, i.e., starting from the sentence symbol and then expanding the obtained partial parse trees by applying some appropriate rule until the sentence is

obtained in the leaf nodes of the tree. This process corresponds to a *top-down* construction of the parse tree.

Definition 1. *The* top-down parsing MDP *for sentence w_{1K} and grammar $G = \{\mathcal{N}, \mathcal{W}, S, \mathcal{R}, \sigma\}$ is a 5-tuple $\mathcal{M}^G_{w_{1K}} = (\mathcal{X}, \{\mathcal{A}(x)\}, T, \mathcal{X}_T, r)$ defined as follows: $\mathcal{X}$ is the state space consisting of states that represent partial parse trees and is constructed recursively from $\{(S, 1, K)\}$ by applying actions in the obtained states and following the transitions. In particular, in state x the set of admissible actions is $A(x) = \{(c, R, split) : c \in x, R \in \mathcal{R}, start_c \leq split < end_c\} \cup \{(c, R) : c \in x, start_c = end_c = p, R \equiv N_c \to w_p\}$. The transition model is deterministic: In case of an action of the form $(c, R, split)$ the next state x' is obtained by adding the new constituents $c_{left} = (N_{left}, start_c, split)$ and $c_{right} = (N_{right}, split{+}1, end_c)$ to x, where R is the rule $N_c \to N_{left}N_{right}$. When the action is of the form (c, R) with $R \equiv N \to w$, the new state x' is obtained by adding the new constituent $(w, start_c, start_c)$. The set of terminal states are those states x where the set of admissible actions, $A(x)$, is empty. The reward function, r is defined as follows: It is a function $r : \mathrm{Dom}_r \to \mathbb{R}$, where $\mathrm{Dom}_r = \{(x, a) : x \in \mathcal{X}, a \in A(x)\}$, where the reward of a transition to a terminal state x_T is $-\infty$ if this terminal state does not correspond to a full parse tree of w_{1K}. Such failure states will be denoted by X_F.*

It is easy to see that any policy in these MDPs terminates after a finite number of steps. When needed we subindex the various elements of $\mathcal{M}_{w_{1K}}$ (state space, action-space, etc.) by w_{1K}. Let us now define the MDP corresponding to a grammar:

Definition 2. *Let $G = \{\mathcal{N}, \mathcal{W}, S, \mathcal{R}, \sigma\}$ be a grammar. Then the top-down parsing MDP corresponding to G, $\mathcal{M}_G = (\mathcal{X}, \{A(x)\}, T, \mathcal{X}_T, r)$ is the disjoint union of the MDPs corresponding to all sentences generated by G: This MDP is obtained by first annotating all the states of the top-down parsing MDP $\mathcal{M}_{w_{1K}}$ by w_{1K} and then joining the resulting disjoint state-space MDPs.*

For the sake of simplicity, when it is clear from the context which MDP we are in, we will use the unannotated symbols.

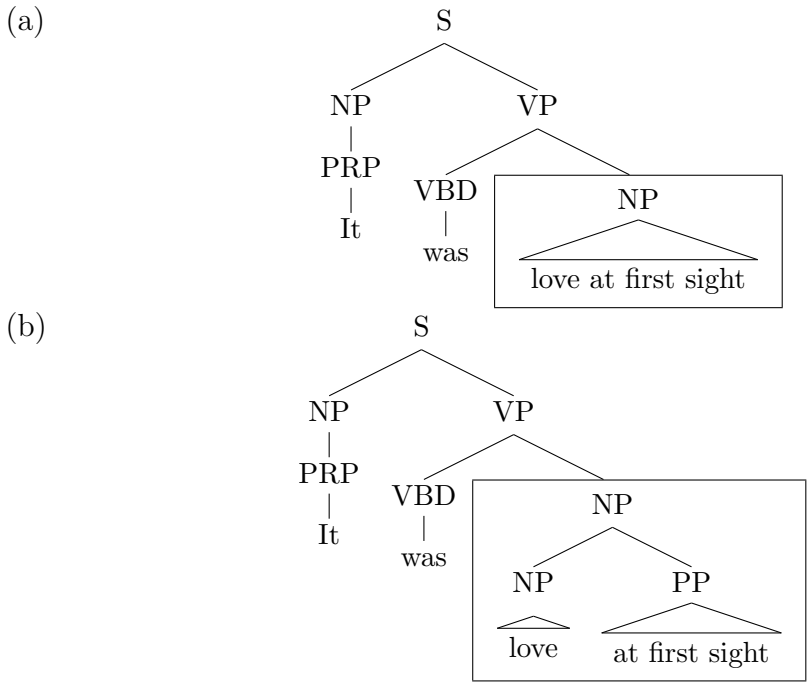

Figure 4.2: Partially parsed tree (a) before and (b) after expanding constituent $(\text{NP}, 3, 6)$.

The idea of state construction is illustrated below: Figure 4.2.a shows the tree representing state

$$x = \{\text{“It was love at first sight”}, (\text{S}, 1, 6), (\text{NP}, 1, 1), \\ (\text{PRP}, 1, 1), (\text{VP}, 2, 6), (\text{VBD}, 2, 2), (\text{NP}, 3, 6)\},$$

while Figure 4.2.b shows the tree after applying action $((\text{NP}, 3, 6), \text{NP} \rightarrow \text{NP PP}, 3)$.

In order to get a representation that is not equivalent in power to PCFGs one can take the reward function to depend on the action only, except for the transitions to the “invalid” terminal states. The following result follows immediately from the above construction and hence its proof is omitted:

Proposition 3. *Let G be a grammar and $\mathcal{M}_G$ be the corresponding top-down parsing MDP. Assume that for any non-failure state x of the top-down parsing MDPs that $\mathcal{M}_G$ is composed of, it holds that $r(x, a) = \log(\mathcal{P}(R^a))$. Let π^* be an optimal policy in $\mathcal{M}_G$. Pick some sentence $w_{1K} \in \mathcal{W}^*$ and let*

$x^*_{w_{1K}}$ be the terminal state reached by π^* when started in $\mathcal{M}_{w_{1K}}$ from state $x_0 = (w_{1K}, (S, 1, K))$. Then if w_{1K} is in the grammar G then $x^*_{w_{1K}}$ will not be a failure state and $V^*_r(x_0) = \log p(\tau, s|G)$, otherwise $x^*_{w_{1K}}$ will be a failure state and $V^*_r(x_0) = -\infty$.

For knowing the optimal policy it is necessary to calculate the optimal action-values (or advantages). The next statement shows how this can be reduced to the calculation of optimal state values in a problem with a larger state space. For this for a given MDP $\mathcal{M}_{w_{1K}}$ we construct the new MDPs $\mathcal{M}_{w_{ij},N,i,j}$, where the states of $\mathcal{M}_{w_{ij},N,i,j}$ are generated from (N, i, j) with the same construction that is used to define $\mathcal{M}_{w_{1K}}$, N is a non-terminal of G and $1 \leq i \leq j \leq K$. The larger problem is obtained as the union of these MDPs. It shall be denoted by $\mathcal{M}'_{w_{1K}}$. Then the following holds:

Proposition 4. *Let $w_{1K} \in \mathcal{W}^*$ and consider $\mathcal{M}'_{w_{1K}}$. Pick x in the state space of $\mathcal{M}'_{w_{1K}}$ and $a \in A(x)$. Assume that $a = (c, R, split)$, where $R \equiv N \to N_{left}N_{right}$. Let $x_{left} = (N_{left}, start_c, split)$, $x_{right} = (N_{right}, split + 1, end_c)$, $x_N = (N, start_c, end_c)$. Let the optimal value function of $\mathcal{M}'_{w_{1K}}$ be V^*_r. Then*

$$A^*_r(x, a) = r(x, a) + V^*_r(x_{left}) + V^*_r(x_{right}) - V^*_r(x_N). \tag{4.3}$$

Proof. We have $A^*_r(x, a) = r(x, a) + V^*_r(x') - V^*_r(x)$, where x' is the state obtained by applying a in x. Further, $V^*_r(x') - V^*_r(x) = V^*_r(x_{left}) + V^*_r(x_{right}) - V^*_r(x_N)$ thanks to the effects of actions being local and the construction of x_{left}, x_{right} and x_N. □

Note that $V^*_r(x_{left})$, $V^*_r(x_{right})$, $V^*_r(x_N)$ are just the inside Viterbi-scores of the respective constituents and they can be computed efficiently using dynamic programming methods as the CKY algorithm or the Earley algorithm (Manning and Schütze, 1999).

4.3 Using Inverse Reinforcement Learning to Learn to Parse

In this section we first present the common ideas underlying applying IRL techniques to parser training, followed by the description of the resulting parser training algorithms.

4.3.1 Common Ideas

This subsection collects all the common ideas needed to apply IRL to parsing. One crucial question is how to set up the features of the reward function. Although in theory at any stage the rewards could depend on the full partial parse tree, in order to facilitate comparison with standard PCFG training where scores are assigned to probabilities we choose to set the features at (x, a), where $a = (c, R, split)$ is an action to depend on the identity of the rule applied: If $\{R_i\}_{i=1}^{n_R}$ is the list of all rules, the features are n_R dimensional, binary and

$$\phi_i(x, a) = \mathbb{I}(R_i = R^a), \quad i = 1, 2, \ldots, n_R.$$

Trees from the treebank Λ will be used as traces of expert behavior. A single treebank tree will be denoted by $\tau_E \in \Lambda$. Although the trees do not allow us to know the exact sequence of "actions" taken by the human parser (what nonterminal to expand first), luckily this information is not needed by the algorithms, since *any* ordering of the same expansions gives rise to the same tree and since the features (hence, the immediate rewards) do not depend on the state of the parse tree. One simple approach then is to assume that the expert always chooses the leftmost unexpanded nonterminal and this is indeed the approach that we take. This yields a series of state-action pairs, allowing us to apply IRL algorithms. Formally, we take $\mathcal{D}_{\tau_E} = \{(x_t, a_t)\}_{t=1}^{H_{\tau_E}}$ and choose $\mu_{\tau_E}(x) = \frac{1}{H_{\tau_E}}$ as the state visitation frequencies for the states in the computed sequence. The situation would change if the features were nonlocal in the sense that they would depend on parts of the parse tree other than the path from the root to the node to be expanded.

In order to apply the presented IRL algorithms, we will need to compute

feature expectations. The (approximate) computation of the expert's unconditional feature expectations $\bar{\Phi}_E$ is straightforward: it is the average feature count given the treebank trees:

$$\bar{\Phi}_E^i = \frac{1}{|\Lambda|} \sum_{\tau_E \in \Lambda} f(R_i, \tau_E), \quad i = 1, 2, \ldots, n_R, \tag{4.4}$$

where $f(R, \tau_E)$ means the total count of rule R in the parse tree τ_E (cf. Equation (4.1)). The feature expectation for the policy optimal w.r.t. the reward function r ($\bar{\Phi}_r$) can be computed for all MDPs $\mathcal{M}^G_{y_{\tau_E}}$ as the feature counts in the respective maximum scoring tree τ^*. The feature expectations in the final MDP $\mathcal{M}^G$ is then computed by averaging over all trees again.

The computation of conditional feature expectations is a bit more difficult. Take a single MDP representing the parsing process for some sentence. Then up to an additive constant vector $\Phi_r(x, a)$ equals[1] the feature counts in tree $\tau(x, a)$, which is the solution of the MDP $\mathcal{M}_{w_{ij},(N,i,j)}$, assuming that $c^a = (N, i, j)$. This can be computed relatively fast as the precomputed Viterbi scores can be used to select the maximum scoring subtrees.

4.3.2 Parser Training Algorithms

In this section we present the four parser training algorithms resulting from the respective IRL algorithms.

Projection

In the learning algorithm derived from the projection algorithm of Abbeel and Ng (2004), behaviors are represented with the total count of rules used during parsing the treebank. This way the dissimilarity between the treebank tree τ_E and the tree τ (see Equation (3.2)) is directly related with the difference of specific rule counts in τ_E and in τ. In other words, the dissimilarity of two trees reflect the number of rules that appear in one tree, but does not appear in the other tree.

[1]This additive constant vector will drop out when we are substituting into Equation 3.22.

The update step at the k-th step is (see Equation (3.3))

$$\sigma(R) = \sigma(R) + \beta^{(k)}(f(R, \tau_E) - f(R, \tau_k^*)) - \beta^{(k)}\theta^{(k)},$$

where τ_k^* is the maximum scoring tree w.r.t. the rewards at iteration k, $\beta^{(k)}$ is the actual step size parameter given by substituting into Equations 3.4 and 3.5. The pseudocode of the resulting algorithm is displayed as Algorithm 1.

Algorithm 1 Projection algorithm for parsing

Input: corpus Λ, initial scores σ, iteration limit k_{max}
for k **in** $1 \dots k_{max}$ **do**
  for $\tau_E \in \Lambda$ **do**
    $\Delta \leftarrow 0$
    for $\tau_E \in \Lambda$ **do**
      $V \leftarrow \text{insideScores}(y_{\tau_E}, G)$
      $\tau^* \leftarrow \text{viterbiParse}(y_{\tau_E}, V)$
      for i **in** $1 \dots n_R$ **do**
        compute $\beta^{(k)}$ using Equations 3.4 and 3.5
        $\Delta_i \leftarrow \Delta_i + \beta^{(k)}(f(R_i, \tau_E) - f(R_i, \tau^*)) - \beta^{(k)}\sigma(R_i)$
      end for
    end for
    for i **in** $1 \dots n_R$ **do**
      $\sigma(R_i) \leftarrow \sigma(R_i) + \frac{1}{|\Lambda|}\Delta_i$
    end for
  end for
end for

MWAL/Perceptron With Multiplicative Updates

As previously shown in Section 3.2.3, this algorithm only differs from the perceptron algorithm of Freund and Schapire (1999) in that the parameters are replaced by their exponentials. This implies that the resulting parser training method will be closely related to the structured perceptron algorithm of Collins and Roark (2004). There must be a small change made to our definitions if we want to use this algorithm for parser training: this algorithm requires strictly positive parameters. In our setting, all parameters have strictly *negative* values (as they are logarithms of probabilities), thus, if we

absorb this change of signs into the features, we meet the aforementioned assumptions. This means that the feature expectations is given by (compare with Equation (4.4))

$$\bar{\Phi}_i = -f(R_i, \tau_E), \quad i = 1, 2, \ldots, n_R.$$

The update step for this algorithm will be (when selecting $c_1 = 1$, see Equation (3.7))

$$\sigma(R) = \sigma(R) e^{\alpha^{(k)}(f(R,\tau^*) - f(R,\tau_E))}, \ \forall R \in \mathcal{R}.$$

The resulting algorithm is shown as Algorithm 2.

Algorithm 2 MWAL algorithm for parser training

Input: corpus Λ, initial scores σ, iteration limit k_{max}, step size sequence α
for k **in** $1 \ldots k_{max}$ **do**
  $\Delta \leftarrow 0$
  for $\tau_E \in \Lambda$ **do**
    $V \leftarrow \text{insideScores}(y_{\tau_E}, G)$
    $\tau^* \leftarrow \text{viterbiParse}(y_{\tau_E}, V)$
    for i **in** $1 \ldots n_R$ **do**
      $\Delta_i \leftarrow \Delta_i - (f(R_i, \tau_E) - f(R_i, \tau^*))$
    end for
  end for
  for i **in** $1 \ldots n_R$ **do**
    $\sigma(R_i) \leftarrow \sigma(R_i) e^{\frac{\alpha^{(k)}}{|\Lambda|}\Delta_i}$
  end for
end for

Max-Margin Parsing

As this method emerges from the structured prediction community, it is no surprise that applying it to the parsing problem, we get a previously known method. The performance measure is essentially the same as that of the Max-Margin Parsing algorithm proposed by Taskar et al. (2004), but the optimization method is different as we follow Ratliff et al. (2007) who propose to use a subgradient method. According to Shalev-Shwartz et al. (2007) (see also the references therein) subgradient methods can be faster

and more memory efficient than interior point or decomposition methods for max-margin problems. As a concrete example, an exponentiated gradient descent in the dual variables can perform better than sequential minimal optimization (Bartlett et al., 2005; Globerson et al., 2007). Note that the MWAL algorithm can be regarded as exponential gradient descent in the primal variables, so it is distinct from these methods.

In our implementation we chose $\ell(a) = c_\ell f(R^a, \tau_E)$ $(a \in \mathcal{A})$ to be the loss function for the treebank tree τ_E. This loss thus encourages giving high reward to rules that are frequently used in the corpus. The resulting algorithm is shown as Algorithm 3.

Algorithm 3 Max-margin algorithm for parser training

Input: corpus Λ, initial scores σ, iteration limit k_{max}, loss constant c_ℓ, step size sequence α, regularization constant λ
for k **in** $1 \dots k_{max}$ **do**
  $\Delta \leftarrow 0$
  for $\tau_E \in \Lambda$ **do**
    $\mathcal{R}_E = \{R \in \mathcal{R} : f(R, \tau_E) \neq 0\}$
    for $R \in \mathcal{R}_E$ **do**
      $\sigma(R) \leftarrow \sigma(R) - c_\ell f(R, \tau_E)$
    end for
    $V \leftarrow \text{insideScores}(y_{\tau_E}, G)$
    $\tau^* \leftarrow \text{viterbiParse}(y_{\tau_E}, V)$
    for i **in** $1 \dots n_R$ **do**
      $\Delta_i \leftarrow \Delta_i + (f(R_i, \tau_E) - f(R_i, \tau^*)) - \lambda\sigma(R_i)$
    end for
    for $R \in \mathcal{R}_E$ **do**
      $\sigma(R) \leftarrow \sigma(R) + c_\ell f(R, \tau_E)$
    end for
  end for
  for i **in** $1 \dots n_R$ **do**
    $\sigma(R_i) \leftarrow \sigma(R_i) + \frac{\alpha^{(k)}}{|\Lambda|}\Delta_i$
  end for
end for

It is clear that setting $c_\ell = 0$ yields a regularized version of the perceptron algorithm of Collins and Roark (2004).

Policy matching

The dissimilarity measure for this algorithm (see Eq. 3.21) can be interpreted in the parsing domain in the following way. The value of $\mu_E(x)$ is constant and sums to one over all states that the expert visited. In all such states x, the following holds:

$$\sum_{a \in \mathcal{A}(x)} (\pi_\theta(a|x) - \pi_E(a|x)) = \begin{cases} 1, & \text{if } a = a_E(x) \\ 0, & \text{else} \end{cases},$$

where $a_E(x)$ is the action selected by the expert at state x. In other words, this expression indicates if the new constituents induced by the optimal action in this state are in the gold standard tree or not. Summing up over all the states, we get the ratio of unmatched decisions. Note that this does not equal the number of unmatched rules in the maximum scoring parse tree, as the states visited by the expert may not be present in the derivation of this parse tree.

The pseudocode for this algorithm is shown as Algorithm 4. Note that the computation of A^*_r can be easily done once the Viterbi scores have been computed for all spans and all nonterminal symbols. These scores are available without extra computation if we are using a Viterbi parser (e.g., a CKY parser). The subtree $\tau(x, a)$, can also be easily found by looking up these precomputed scores.

4.4 Empirical evaluation

We experimentally evaluated our algorithms on the Penn Treebank WSJ corpus. The settings that we used were the same as those used by Taskar et al. (2004), Titov and Henderson (2007) and Turian and Melamed (2006)—i.e., we have trained and tested all of our methods on the sentences not longer than 15 words, and unless otherwise mentioned we used sections 2-21 for training, 22 for development, and 23 for final testing. The baseline that we were using is a CNF transformation of a simple parent-annotated grammar extracted from the training set. This grammar is much simpler than the one

Algorithm 4 Policy matching algorithm for parser training

Input: corpus Λ, initial scores σ, iteration limit k_{max}, step size sequence α
for k **in** $0 \dots k_{max}$ **do**
  $\Delta \leftarrow 0$
  for $\tau_E \in \Lambda$ **do**
    $V \leftarrow \text{insideScores}(y_{\tau_E}, G)$
    for $\tau_x \in \text{subtrees}(\tau_E)$ **do**
      for $a \in \mathcal{A}(x)$ **do**
        compute $A^*_r(x, a)$ as in Eq. 4.3
        $\pi(a|x) \leftarrow B(A^*_r(x, a))$
        $\tau(x, a) \leftarrow \text{viterbiParseStartingWith}(x, a, V)$
        $\Phi(x, a) \leftarrow (f(R_i, \tau(x, a)))_{i=1}^{n_R}$
        $\delta\pi(a|x) \leftarrow \pi(a|x)\frac{1}{\eta}\left(\Phi(x, a) - \sum_{b \in \mathcal{A}(x)} \pi(b|x)\Phi(x, b)\right)$
        $\Delta \leftarrow \Delta + (\text{isCorrect}(x, a) - \pi(a|x))\delta\pi(a|x)$
      end for
    end for
  end for
  for i **in** $1 \dots n_R$ **do**
    $\sigma(R_i) \leftarrow \sigma(R_i) + \frac{\alpha^{(k)}}{|\Lambda|}\Delta_i$
  end for
end for

used by Taskar et al. (2004): it contains 639 nonterminal symbols, which is approximately six times less than the number (3975) reported by Taskar et al. (2004). During the training process, we have only trained the parameters for the binary rules contained in the grammar and used the default lexicon of the Berkeley Parser[2] that scores word-tag pairs with a smoothed estimate of $\log\left(\frac{P(tag|word)}{P(tag)}\right)$. We had a total of 3392 weights to train.

We had multiple aims with our experiments. First, we were interested in comparing the results of some previously published algorithms (max-margin parsing and the perceptron algorithm) with some new ones (projection, MWAL, policy matching) that have never been tried on parser training. Second, we were interested in the sensitivity of the algorithms to the step-size choice, how performance changes with the size of the training set and the effect of regularization. Finally, we are interested in the robustness of the results. Therefore, we decided to compare the algorithms with cross-validation.

[2] `http://nlp.cs.berkeley.edu/Main.html#Parsing`, Petrov and Klein (2007)

We also want to draw attention to the importance of refined parameter training methods. We can directly compare our results to those of Taskar et al. (2004), as our training and test setup is the same. Although our model is six times smaller than theirs, its performance is comparable. Note that finding the optimal parse is of $\mathcal{O}(n^3k^3)$ complexity in the worst case, where n is the number of nonterminal symbols, and k is the length of the sentence to parse. Thus a sixfold decrease in the number of nonterminals can potentially represent a $6^3 = 216$-fold speedup in parsing of which we expect to realize at least a factor of 2-4.

We have run our algorithms for 100 iterations, and measured performance on the training, development and test sets at all iterations. After 100 iterations, we select the parser that attained maximal F_1 score on the development set during the learning process. Whenever we report a result for a specific hyperparameter setting (e.g., a specific step size setting, a specific regularization setting, etc.), we mean the result that is given by the best parser selected this way. We have initialized the rule scores using Maximum Likelihood estimation, i.e., the standard training method for the Berkeley Parser. The gradients have been normalized before adding the regularization factors to them or multiplying them with the step sizes. This helps with flat areas of the optimization surface, while (with decreasing step-sizes) it does not hurt much close to the optimum. Max-margin and policy matching both have a single hyperparameter that needs to be selected. We have set the loss constant $c_\ell = 0.5$ for max-margin, and the temperature parameter to $\eta = 0.1$ that worked well in other experiments. Unless stated otherwise, we set the regularization constant $\lambda = 0$ for all the methods.

We have implemented our methods in Java, using code pieces from the Stanford Parser[3] and the Berkeley Parser, and run our experiments on the "condor" cluster of the Computer and Automation Research Institute of the Hungarian Sciences. We have used a total of ca. 30,000 hours of CPU time on PCs with 3 GHz processors and 2 gigabytes of RAM, for developing and testing our methods. We have decided to use a simpler structure to be able to examine the different training methods themselves from more aspects

[3] `http://nlp.stanford.edu/software/lex-parser.shtml`

than usual. 100 training iterations of max-margin, perceptron, and MWAL took approximately 8 hours of running time, policy matching was proven to be approximately 10% slower. This is an extremely short training time as compared to the relatively fast training methods of Turian and Melamed (2006) (5 days) and Titov and Henderson (2007) (6 days). The training of the parser of Taskar et al. (2004) took several months, as mentioned in Turian and Melamed (2006).

First, we want to find out which step sizes are the most suitable for this particular problem. We try constant step sizes, step sizes proportional to $\frac{1}{k}$, $\frac{1}{\sqrt{k}}$, and the Rprop step size selection method (Igel and Hüsken, 2000). The results for different step size selections can be seen in Table 4.1. As projection uses fixed step size parameters, we do not present the results for it on this table. The first thing that we can observe is that the two top performing methods are Max-margin and policy matching. Policy matching performs remarkably well when measuring the performance on the training set. The standard selection of $\alpha^{(k)} \sim \frac{1}{k}$ leads to the least improvement in the parsing performance. We see that $\alpha^{(k)} \sim \frac{1}{\sqrt{k}}$ produces particularly good results for all of the methods – this is due to the fact that this step size selection is known to improve robustness. Constant step sizes also perform good for similar reasons. Using the Rprop does not improve performance on the training set as much as using $\frac{1}{\sqrt{k}}$ or constant step sizes, but it produces excellent results on the training set.

Another interesting question is the dependence of the performance of our methods on the size of the training set. For this experiment we used 1, 2, 5, 10 and 20 sections following Section 2 from the Penn Treebank WSJ corpus, and measured performance on Section 23. Results can be seen on Table 4.2. We see that for small training sets, perceptron and MWAL do a good job in reproducing the training examples, but generalize worse than the other two methods. However, as the training size increases, policy matching gradually takes over them in the sense of achieving good performance on the training set. On the test set, max-margin and policy matching produce the best results, irrespective of the size of the training set. Projection produces very poor results.

	Test error [%]				Training error [%]			
$\frac{1}{k}$	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	90.39	81.95	85.97	49.25	**92.98**	91.43	92.20	58.55
MW	90.43	82.69	86.38	50.41	92.91	91.92	92.41	58.77
MM	**90.57**	82.41	86.3	**50.74**	**92.98**	91.78	92.38	**59.23**
PM	90.1	**83.77**	**86.82**	47.59	92.44	**92.87**	**92.66**	55.4
α	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	89.53	82.26	85.74	50.24	92.70	91.96	92.32	58.24
MW	89.87	83.45	86.54	51.57	92.02	92.32	92.17	55.57
MM	91.91	**84.47**	**88.03**	52.4	93.67	92.90	93.28	61.27
PM	**92.02**	84.25	87.96	**53.39**	**94.59**	**93.43**	**94.01**	**64.41**
$\frac{1}{\sqrt{k}}$	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	90.66	83.06	86.69	51.4	92.87	92.18	92.53	58.54
MW	90.18	83.19	86.54	50.91	92.63	92.23	92.43	58.52
MM	91.64	**84.38**	87.86	52.07	93.63	93.02	93.33	61.65
PM	**92.13**	84.27	**88.03**	**54.22**	**94.51**	**93.22**	**93.86**	**63.94**
Rp	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	89.24	82.82	85.91	47.42	92.52	91.84	92.18	57.80
MW	89.16	81.82	85.33	48.42	92.42	91.50	91.96	57.23
MM	91.09	83.34	87.04	53.23	94.11	92.86	93.48	62.36
PM	**91.75**	**83.92**	**87.66**	**54.22**	**95.05**	**93.64**	**94.34**	**66.85**

Table 4.1: Results for different step size selections. Abbreviations: ML=Maximum Likelihood, PE=Perceptron, MW=MWAL, MM=Max-Margin, PM=Policy Matching. Every part of the table is labeled by step sizes (α means constant step size, Rp stands for Rprop).

	Test error [%]				Training error [%]			
1	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	86.70	57.61	69.22	32.17	93.19	91.39	92.28	63.31
PE	86.60	58.37	69.73	30.51	**97.15**	96.01	96.57	**78.19**
PR	79.02	56.37	65.80	23.88	85.37	88.96	90.07	40.81
MW	86.46	58.52	69.80	31.67	97.02	**96.30**	**96.66**	76.93
MM	**88.21**	**59.32**	**70.94**	**34.32**	96.82	95.69	96.25	74.63
PM	87.50	59.28	70.67	32.50	95.73	95.32	95.52	72.74
2	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	87.65	63.95	73.95	**40.46**	93.02	90.24	91.61	59.79
PE	87.19	65.06	74.52	36.65	95.19	94.28	94.73	68.09
PR	73.55	61.20	66.81	25.04	84.67	86.73	88.65	36.97
MW	86.81	64.60	74.08	39.13	95.96	**95.31**	**95.63**	**70.85**
MM	**88.81**	**65.71**	**75.53**	**40.46**	95.54	94.72	95.13	70.47
PM	88.34	65.32	75.10	39.30	**96.30**	94.56	95.42	69.34
5	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.32	75.04	81.56	47.26	92.35	90.06	91.19	57.79
PE	89.43	77.38	82.97	45.93	93.28	93.53	93.40	60.60
PR	80.97	69.98	75.07	30.84	85.10	83.56	86.80	32.38
MW	90.08	76.95	83.00	46.76	94.26	93.29	93.77	64.34
MM	90.49	77.34	83.40	47.59	94.47	93.56	94.01	64.61
PM	**91.39**	**77.75**	**84.02**	**49.58**	**95.24**	**93.89**	**94.56**	**65.80**
10	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	88.64	78.27	83.13	47.59	92.30	89.99	91.13	58.50
PE	89.23	80.98	84.90	49.58	92.96	93.14	93.05	59.85
PR	81.08	74.29	77.54	30.18	85.70	84.86	86.39	39.21
MW	89.25	79.85	84.29	48.92	93.92	92.57	93.24	62.37
MM	90.29	80.78	85.27	51.07	94.08	93.24	93.66	63.30
PM	**92.02**	**81.93**	**86.68**	**53.23**	**95.30**	**93.60**	**94.44**	**66.15**
20	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	90.66	83.06	86.69	51.40	92.87	92.18	92.53	58.54
PR	80.58	73.38	76.81	34.16	85.79	82.37	84.12	39.28
MW	90.18	83.19	86.54	50.91	92.63	92.23	92.43	58.52
MM	91.64	**84.38**	87.86	52.07	93.63	93.02	93.33	61.65
PM	**92.13**	84.27	**88.03**	**54.22**	**94.51**	**93.22**	**93.86**	**63.94**

Table 4.2: Results for different training set sizes. Step size was set to $\frac{1}{\sqrt{k}}$. Abbreviations: ML=Maximum Likelihood, PE=Perceptron, PR=projection, MW=MWAL, MM=Max-Margin, PM=Policy Matching. Every part of the table is labeled by numbers of chapters used for training.

In the next phase, we examined the effects of regularization to our methods. We expect that the methods which performed very well on the training set (policy matching, particularly with Rprop) will take the greatest advantage of regularization, as we can trade off between training set performance and holdout set performance when using regularization. The regularization constant is fixed in the projection algorithm, so results are not shown for it. We report results with Rprop and $\frac{1}{\sqrt{k}}$ step sizes. Although using step sizes proportional to $\frac{1}{\sqrt{k}}$ by itself has some regularization effect, using explicit regularization improves performance. The dependency of the exact match ratio on the choice of the regularization constant can be seen on Figure 4.3a. The same result for Rprop step sizes are shown on Figure 4.3b. Matching our expectations, we see that regularization has a positive effect for the policy matching algorithm, but it does not improve performance of the other methods. The performance of max-margin decreases monotonically as the regularization parameter increases, the behavior of MWAL and perceptron is irregular. For policy matching, we see that there is an interval of regularization values that improve performance, however, this interval is not the same for all step size selections. For $\alpha^{(k)} \sim \frac{1}{\sqrt{k}}$, the best choice is $\lambda = 0.005$. With this choice, the performance of the trained parser is as follows: LP=92.86%, LR=84.68%, F_1=88.58%, EX=55.80%. This is our best result that was produced, which means a 26.46% error reduction in F_1-measure, as compared to our baseline. The best parser reached with Rprop is produced by using $\lambda = 0.0001$. The performance is then LP=91.99%, LR=83.58%, F_1=87.58%, EX=55.38%. In this case regularization only helps in the exact match ratio performance measure.

We believe that the practice used in the parsing community to report results only on the standard training and test setup is dangerous in the sense that we do not get any information about the reliability of the methods tested. Therefore we performed 10-fold cross-validation on Sections 2–22 of the corpus. We report results for Rprop and $\alpha^{(k)} = \frac{1}{\sqrt{k}}$ step sizes, with and without regularization on Table 4.3. We have performed paired t-tests to see whether the measured differences are significant or not. Again, we see that max-margin and policy matching perform significantly better than percep-

(a)

(b)

Figure 4.3: Exact match ratio measured on the test set vs. the regularization constant when (a) $1/\sqrt{k}$ step sizes (b) Rprop is used.

	Test error [%]							
	LP		LR		F_1		EX	
$\lambda = 0$	μ	σ	μ	σ	μ	σ	μ	σ
ML	90.21	0.61	82.38	**1.12**	86.12	0.80	48.45	1.80
PE	90.74	0.58	85.16	1.18	87.86	**0.61**	49.68	2.09
MW	90.91	0.69	84.98	1.13	87.84	0.73	49.88	2.25
MM	**91.89**	**0.57**	85.27	1.14	**88.45**	0.78	**52.05**	**1.78**
PM	91.24	1.39	**85.49**	1.20	88.26	1.07	48.94	5.42
	LP		LR		F_1		EX	
$\lambda = 0.005$	μ	σ	μ	σ	μ	σ	μ	σ
ML	90.21	0.61	82.38	1.12	86.12	0.80	48.45	**1.80**
PE	90.29	0.77	84.43	1.40	87.25	0.75	46.71	2.16
MW	89.28	0.71	83.14	1.61	86.09	0.95	45.90	2.46
MM	91.56	0.67	84.55	1.22	87.91	0.87	50.59	2.47
PM	**92.23**	**0.55**	**84.98**	**1.07**	**88.46**	**0.64**	**52.80**	2.01

Table 4.3: Cross-validation results for different step sizes and regularization settings. Abbreviations: ML=Maximum Likelihood, PE=Perceptron, MW=MWAL, MM=Max-Margin, PM=Policy Matching. In the table μ denotes the estimated performance and σ is the estimated standard deviation. Every part of the table is labeled by applied step sizes and regularization constant.

tron and MWAL. The differences between perceptron and MWAL, and those between max-margin and policy matching are not significant. However, the effect of regularization to policy matching is significant in the exact match ratio and the labeled precision, at the confidence level of 95%. We can also conclude that what seems to be the current practice of measuring performance only on the last chapter may yields to false conclusions. In our case, policy matching looks as if it had a definitely better performance than max-margin training, while the result of cross-validation predicts no significant differences. However, when the performance differences are larger it seems less problematic to measure performance only on the last chapter.

Our results suggest that the max margin planning and policy matching algorithms are particularly useful for PCFG parameter training. Our best parser was trained using regularized policy matching, which achieved 88.58% F_1 measure on the testing set. This means a 26.46% error reduction in F_1-

measure, as compared to our baseline. We can compare this result to that of Taskar et al. (2004), whose training algorithm achieves only 1.74% error reduction in the same measure without introducing extra features. With the introduction of lexical features and using an auxiliary POS-tagger, they report a 9.4% error reduction over the baseline (however, they do not report results for their generative baseline model using this POS-tagger).

Chapter 5

Conclusions

In this thesis we discussed a number of IRL methods in a unified framework which allowed us to observe the relations between these algorithms. There are a number of methods that aim to solve the IRL problem, however, there are no published works that would compare them. In particular, we have seen that the projection, MWAL and the maximum margin algorithms are very close to each other, while the policy matching algorithm is in a disjoint class of methods. Further, by setting the loss function to the identically zero one in the max-margin algorithm, we get a perceptron-like algorithm. By using multiplicative updates in this perceptron method, we get the MWAL algorithm. The similarities between these methods can help us to gain more insight to why and how they work. For example, it is known that incremental methods that use exponential weights, such as MWAL, perform best on problems where the feature set is dense and the weight vector is sparse (Cesa-Bianchi and Lugosi, 2006). Least-mean square methods usually work better in the opposite setting. One idea would be to use p-norm algorithms that interpolates between exponential weight and LMS algorithms.

We proposed a new approach to inverse reinforcement learning problems, that is based on minimizing ℓ^2-distance to the expert's policy. We have formalized this task as a quadratically constrained quadratic programming problem, and have shown a way to linearize its constraints. We have proposed a simple subgradient-based minimization method for the optimization

problem. We have shown that the updates for the subgradient method can be effectively computed if we have access to an efficient value-based reinforcement learning algorithm.

We have experimentally compared versions of our method to some previous methods. Our algorithm was seen to be much more sample-efficient than the others in the standard setting where the searched reward function lies in the span of the feature set that we use. We have examined the effect of using corrupted feature sets and have seen that although theoretically our algorithm is more likely to fail under these conditions than the other methods, its performance still remains superior. We have tested the methods when irrelevant features were added to the actual feature set, and have seen that this has a positive effect on other methods, while degrading the performance of our method. We have shown that this problem can be overcome by using a regularized version of our method.

In the second part of the thesis, we proposed an application to inverse reinforcement learning in natural language processing. We have introduced an analogy of PCFG parsing and MDPs and exploited this concept to derive some parser training algorithms from existing methods for IRL. We have seen that while most of these methods already existed in structured prediction, our policy matching algorithm is a completely new algorithm for parser training.

We provided a much needed, objective empirical comparison of existing solutions for parser training. We have seen that the performance of the policy matching algorithm is comparable with that of the now-popular max-margin method. Interestingly, we have found that the policy matching algorithm tends to produce especially good results on the training set. With regularization this was then converted into an improved performance on the test set.

Although we derived our algorithms for training PCFG models, the resulting methods can easily be generalized to training weights in other structured prediction models. If it is possible to construct the structured output for a specific problem in a sequential manner then the problem can be rewritten as an MDP and IRL methods can be applied for training the the predictors. This has the benefit that whenever a new IRL algorithm is produced, this

can be turned into a training algorithm for structured prediction problems. Thus, in the future we expect to see IRL algorithms to be used in training structured predictors and vice versa.

Bibliography

Abbeel, P. and Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML'04*, pages 1–8.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Amit, R. and Mataric, M. J. (2004). A correspondence metric for imitation. In *AAAI*, pages 944–945.

Atkeson, C. and Schaal, S. (1997). Robot learning from demonstration. In *ICML'97*, pages 12–20.

Bakir, G. H., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. V. N. (2007). *Predicting Structured Data (Neural Information Processing)*. The MIT Press.

Bartlett, P. L., Collins, M., Taskar, B., and McAllester, D. (2005). Exponentiated gradient algorithms for large-margin structured classification. In *Advances in Neural Information Processing Systems 17*, pages 113–120, Cambridge, MA. MIT Press.

Baxter, J. and Bartlett, P. (1999). Direct gradient-based reinforcement learning. Technical report, Research School of Information Sciences and Engineering, Australian National University.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IEEE Transactions on*, 2(3):113–124.

Collins, M. (2002). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA. Association for Computational Linguistics.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 111–118, Morristown, NJ, USA. Association for Computational Linguistics.

Daumé III, H. (2006). *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California, Los Angeles, CA.

Demiris, J. and Hayes, G. (1996). Imitative learning mechanisms in robots and humans. In *EWLR-5*, pages 9–16.

Elliott, H., Derin, H., Cristi, R., and Geman, D. (1984). Application of the Gibbs distribution to image segmentation. In *Proc. 1984 Int. Conf. Acoust., Speech, Signal Processing, ICASSP'84*, pages 32.5.1–32.5.4.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296.

Globerson, A., Koo, T. Y., Carreras, X., and Collins, M. (2007). Exponentiated gradient algorithms for log-linear structured prediction. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 305–312, New York, NY, USA. ACM.

Igel, C. and Hüsken, M. (2000). Improving the Rprop learning algorithm. In *Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000)*, pages 115–121. ICSC Academic Press.

Kuniyoshi, Y., Riekki, J., Ishii, M., Rougeaux, S., Kita, N., Sakane, S., and Kakikura, M. (1994). Vision-based behaviors for multi-robot cooperation. In *IEEE/RSJ IROS*, pages 925–931.

Maes, F., Denoyer, L., and Gallinari, P. (2007). Sequence labeling with reinforcement learning and ranking algorithms. In *ECML*, pages 648–657.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts.

Neu, G. and Szepesvári, C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 295–302.

Ng, A. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *ICML-2000*, pages 663–670.

Petrov, S. and Klein, D. (2007). Learning and inference for hierarchically split PCFGs. In *AAAI 2007 (Nectar Track)*.

Pomerleau, D. (1988). ALVINN: An autonomous land vehicle in a neural network. In *NIPS*, pages 305–313.

Price, B. and Boutilier, C. (2003). A Bayesian approach to imitation in reinforcement learning. In *IJCAI*, pages 712–720.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In *IJCAI*, pages 2586–2591.

Ratliff, N., Bagnell, J., and Zinkevich, M. (2006). Maximum margin planning. In *ICML'06*, pages 729–736.

Ratliff, N., Bagnell, J. D., and Zinkevich, M. (2007). (Online) Subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*.

Rivas, E. and Eddy, S. R. (1999). A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J Mol Biol*, 285(5):2053–2068.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. (1992). Learning to fly. In *ML*, pages 385–393.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 807–814, New York, NY, USA. ACM.

Shiraev, D. E. (2003). Inverse reinforcement learning and routing metric discovery. Master's thesis, Department of Computing Science, Virginia Tech.

Shor, N., Kiwiel, K., and Ruszczynski, A. (1985). *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc., New York, NY, USA.

Silva, V. F. d., Costa, A. H. R., and Lima, P. (2006). Inverse reinforcement learning with evaluation. In *IEEE/ICRA'06*, pages 4246–4251.

Sutton, R. and Barto, A. (1998). Reinforcement learning: An introduction. *Bradford Book*.

Syed, U. and Schaphire, R. (2007). A multiplicative weights algorithm for apprenticeship learning. In *Advances in Neural Information Processing Systems 21*.

Syed, U., Schaphire, R., and Bowling, M. (2008). Apprenticeship learning using linear programming. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML)*. To appear.

Taskar, B., Chatalbashev, V., Koller, D., and Guestrin, C. (2005). Learning structured prediction models: a large margin approach. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 896–903, New York, NY, USA. ACM.

Taskar, B., Klein, D., Collins, M., Koller, D., and Manning, C. (2004). Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Titov, I. and Henderson, J. (2007). Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic. Association for Computational Linguistics.

Turian, J. and Melamed, I. D. (2006). Advances in discriminative parsing. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 873–880, Morristown, NJ, USA. Association for Computational Linguistics.

List of Figures

List of Tables