Variational Methods for Reinforcement Learning

by

Sajaysurya Ganesh

under the guidance of

David Barber

Yap Pau Ching

This report is submitted as part requirement for the

MSc Degree in **Computational Statistics and Machine Learning**

at

University College London

September, 2018

Abstract

Traditional methods for Reinforcement Learning (RL) differ from the vast majority of conventional machine learning techniques as they forgo the established workflow of fitting and subsequently inferring from a well defined model. Tracing their history to optimal control theory, traditional value-based RL techniques and the newer policy-based RL techniques rely on the concept of value functions, which is usually estimated iteratively based solely on the agent's experience in the environment. Although impressive, these techniques typically neglect the uncertainty in the value function estimates and struggle to achieve the eternally elusive balance between exploration and exploitation.

This work explores and attempts to improve an uncommon method of using approximate machine learning algorithms like variational Expectation-Maximization (EM) in RL, where the concept of value functions emerge implicitly in the form a variational distribution, which is iteratively fitted and subsequently inferred to find the optimal policy. Its usefulness is motivated by highlighting the potential advantages over other RL techniques (in certain scenarios) using experiments. Unlike value based techniques, EM can be systematically extended for use with Bayesian model-based RL, which are capable of formally quantifying uncertainty of estimates, possibly leading to a better exploration-exploitation balance. Existing works that employ EM to solve Bayesian model-based RL by leveraging the large body of work done in approximate variational inference techniques to deal with the inevitable computational intractabilities are reviewed, and their current limitations are highlighted using experiments. Possible improvements are discussed and experimental results testing their viability are reported.

Impact Statement

This is an academically oriented preliminary work with some potential impact in academia in the near future. The techniques illustrated and experimental evidence reported is expected to provide background and motivate further research in alternative methods for reinforcement learning.

Though the techniques discussed here are currently not mature enough for commercial, real-world applications, it provides context for further research, which might lead to the development of more efficient reinforcement learning techniques that could be adopted by industries in the future.

Contents

List of Figures

Notations

$p(x, y, \ldots)$	The true (discrete or continuous) probability distribution on a set of random variables.
$p(x \mid y = b)$	The conditional probability distribution of x given that the random variable y takes the value b.
$q(x, y, \ldots)$	The (discrete or continuous) variational probability distribution on a set of random variables. Usually, this is an approximation to the true distribution and its parameters are varied by the learning algorithm during the course of learning.
$p(x = a, y = b, \ldots)$	The probability (value) that the random variables take the indicated values.
$\langle f(x) \rangle_{p(x)}$	Expected value of function $f(x)$ as per distribution $p(x)$. In the subscript, p is written instead of $p(x)$ if the random variables are obvious from context.
$\sum_{\setminus x} p(\cdots)$	Marginalizing out all variables except x from the distribution p.
s_t	Random variable for environmental state occupied by the agent at time t. The subscript is sometimes omitted to indicate temporal stationarity.
a_t	Random variable for action taken by agent at time t. The subscript is sometimes omitted to indicate temporal stationarity.

r_t	Random variable for reward received by the agent at time t. The subscript is sometimes omitted to indicate temporal stationarity.
t, τ	(Random) Variables indicating the time-step, usually constrained such that $\tau \leq t$.
π	Set of parameters that parameterized the policy distribution $p(a\|s) = \pi_{a,s}$.
$\boldsymbol{\theta}$	Set of parameters that parameterized the transition distribution $p(s'\|s,a) = \theta^{s'}_{s,a}$. Sometimes the subscripts or superscripts of θ are replaced with the wild-card character '.' which stands for any valid state or action.
$\bar{r}_t = \bar{r}(s_t, a_t)$	The expected reward function that maps state-action pairs to rewards. It is a time-independent function. It is sometimes abbreviated as $\bar{r}_t$.
$f_t = f(s_t, a_t)$	The value function for state-action pairs as introduced in policy iteration. It is a time-independent function. It is sometimes abbreviated as f_t.
$g_t = g(s_t, a_t)$	The message function for calculating the total expected reward as defined while introducing the gradient descent technique. It is sometimes abbreviated as g_t. This can also be considered as a value function for stochastic policies.
$d(s)$	Deterministic Policy, a mapping from a state to an action.
B	Generally used to indicate a bound.
L	Generally used to indicate the Lagrangian.
ω	Step-size parameter for moving average updates.
$\tilde{r}$	Sampled (or observed) reward.
$\tilde{s}$	Sampled (or observed) state.

$\alpha_{s,a}^{s'}$	Prior number of counts of transition from state s to state s' by taking action a. This is used to parameterize the Dirichlet prior. Sometimes the subscripts or superscripts of θ are replaced with the wild-card character '.' which stands for any valid state or action.
$c_{s,a}^{s'}$	Observed number of counts of transition from state s to state s' by taking action a. This is used to parameterize the Dirichlet posterior. Sometimes the subscripts or superscripts of θ are replaced with the wild-card character '.' which stands for any valid state or action.
ζ	Observations made in the Reinforcement Learning environment. Includes the trajectory of states, actions taken and resulting rewards.

Abbreviations

RL	Reinforcement Learning
MDP	Markov Decision Process
ML	Maximum Likelihood
MDP-PI	Policy Iteration for solving MDP
MDP-GD	Gradient Ascent Algorithm for solving MDP
EM	Expectation Maximization
E-step	Expectation Step
M-step	Maximization Step
MDP-EM	EM Algorithm for solving MDP
ML-EM	EM for model-based RL with Maximum Likelihood Estimates
Semi-EM	ML-EM with just one ML and EM update after each time-step
VF-EM	EM for Bayesian model-based RL with a factorized approximation for the variational distribution in E-step
SGVB	Stochastic Gradient Variational Bayes
VAE	Variational Auto-Encoder
M-EM	EM algorithm for Bayesian model-based RL with Monte-Carlo approximation for the marginalized variational distribution in E-step
Semi-M-EM	M-EM with just one Bayesian and EM update after each time-step

Chapter 1

Introduction

Reinforcement Learning (RL) is the machine learning problem, where an agent learns to interact with an environment using its own experience, so that it could maximize the total numerical reward it gets through such interactions. Such an agent constantly faces the choice to either gather more experience (explore) to make better decisions or to use the limited experience to greedily reap (exploit) as much reward as possible. Achieving the right balance between exploration and exploitation is hard and is generally considered as the holy grail of RL.

The common techniques used in RL trace their history to the theory of optimal control and have evolved in a way different from other machine learning techniques. They depend on the concept of value functions [1] [2], which quantizes the potential of a policy –that guides the agent's actions in different environmental states– to result in high numerical reward. The traditional value-based RL techniques evaluates policies using estimates of value functions, which are iteratively improved (e.g Q-learning) to find the value function of the optimal policy that could guide an agent to get the highest possible rewards. The advancements in automatic differentiation and stochastic gradients based optimization led to the development of policy-based RL techniques (e.g actor-critic) that search for (locally) optimum policies in a policy-space while iteratively updating an estimate of a value function using observations from experience. In both value-based and policy-based techniques, the uncertainty associated with the value estimates is usually not explicitly accounted for, which could potentially lead to policies with an unbalanced focus on exploration and exploitation. However, they scale well to larger problems and have led to impressive results in game playing scenarios [3].

Alternatively, RL can also be performed using mainstream machine learning techniques like variational Expectation-Maximization (EM). Notably, in the late nineties, [4] used the EM algorithm to solve RL problems. EM has also been used to solve a variety of Markov Decision Processes (MDP) [5] [6], which is a special case of the general RL problem. Strictly speaking, EM is not drastically different from value and policy based approaches as once again the concept of value functions arise implicitly in the guise of the variational distribution in the E-step. However, EM has some advantages over the established policy and value based approaches, and is naturally extensible to Bayesian model-based RL, which can systematically include the uncertainty in estimates made from the agent's experience, possibly leading to a better balance between exploration and exploitation than that is achieved by the previous techniques. The computational intractabilities that typically characterize the Bayesian approaches can fortunately be tackled using the pre-existing (and quickly expanding) repertoire of approximate variational inference techniques, and works like [7] have already leveraged this advantage to search for better RL algorithms.

This work explores the applicability of EM in RL and attempts to motivate its use by highlighting the potential advantages, and tries to review and improve upon the pre-existing extensions of EM to Bayesian model-based RL. The subsequent chapters are organized as follows.

- Chapter 2 introduces RL and MDPs. Three techniques (Policy Iteration, Gradient Ascent and EM) for solving MDPs are introduced, and the advantages of EM are empirically highlighted using a set of experiments.

- Chapter 3 extends the three techniques introduced in chapter 2 for solving the complete RL problem. Policy Iteration is extended to the value-based Q-learning. Gradient Ascent is extended to policy-based actor-critic technique and the EM algorithm is extended to model-based RL with maximum likelihood estimates. The similarity between EM and the other approaches is detailed. Inspired by the similarities, a semi-converging variant of EM is proposed and its performance is analyzed.

- Chapter 4 again extends the basic EM algorithm to model-based RL with Bayesian estimates of the environment model. The nature of computational intractability in such an extension is discussed and the effectiveness of a factorized approximation (proposed in [7]) for the variational distribution is reviewed. A Monte-Carlo approximation for the marginals of the variational distribution is proposed and its performance is analyzed.

Chapter 2

Markov Decision Processes

This chapter introduces Markov decision processes in the context of reinforcement learning. Three techniques for solving MDPs are discussed: policy iteration, gradient ascent and expectation-maximization. The scenarios in which EM has an advantage over the other two techniques are highlighted using experiments.

2.1 Reinforcement Learning

Reinforcement learning is a paradigm of machine learning in which an "agent" interacts with a stochastic "environment" in real-time in discrete time-steps and learns from past experiences, where "learning" corresponds to developing a "policy" by which the agent could take an "action" based on its current "state" so as to maximize its total numerical "reward" over a given period of time [1]. The interface between the agent and the environment is shown in figure 2.1

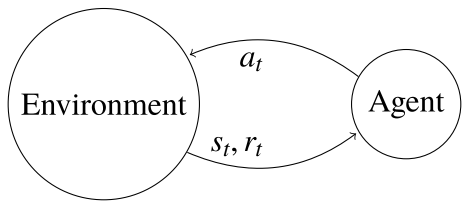


Figure 2.1: Agent-Environment Interface

s_t is the state at time-step t. A state is an abstract entity that may refer to a configuration of the environment or the agent which essentially serves as a context for the agent to select subsequent actions. This configuration can either be absolute or simply the agent's perception. In most cases the state refers to the location of the agent in the environment or to the content of an adequately specified visual field of the agent.

a_t is the action taken by the agent at time-step t. The action is the primary mechanism by which the agent interacts with the environment. Performing an action usually amounts to selecting an action from a predefined list.

r_t is the numerical reward (or penalty, negative-reward) given by the environment to the agent based on the current state s_t and action a_t.

The following are the source of randomness in the environment and the agent:

- $p(s_{t+1}|s_t, a_t)$: **Transition Distribution** gives the probability distribution over the next state given the current state and action.

- $p(r_t|s_t, a_t)$: **Reward Distribution** gives the probability distribution over all possible rewards given the current state and the action performed at that state.

- $p(a_t|s_t)$: **Policy** gives the probability distribution over all actions given the current state. The agent samples an action from this distribution given in the current state.

All the above distributions are (in the context of the current discussion) stationary i.e. the distribution's parameters are independent of time. Note that, this definition includes the cases of deterministic agents and environments as any of the above distributions can be made one-hot[1].

In the general RL problem, all three distributions (transition, reward and policy) are unknown. The transition and reward distribution parameters are learnt from agent's experience in the environment and then the optimal policy is identified to maximize the total numerical reward over a given period of time.

Traditionally, algorithms to identify the optimum policy were developed for a simpler setting where the transition and reward distributions are known and are later extended to the general RL setting. This simpler problem with known transition and reward distribution is called as a Markov Decision Process (MDP).

[1]One-hot distributions are those where all the probability mass (totals to 1) is assigned to just one among all the possible values that a random variable can take. As a result, any sample from the distribution will always (deterministically) be that value with all the probability mass.

2.2 Solving Markov Decision Processes

Solving an MDP amounts to finding the policy that maximizes the total expected reward $U(\pi)$, where π is the set of parameters that define the policy distribution $p(a_t|s_t)$ such that $\pi_{a,s} = p(a_t = a|s_t = s)$ in the case of discrete states and actions. The total expected reward is quite literally the expected reward (expectation taken over different state-action pairs) in each time-step summed over all time steps and can be written as

$$U(\pi) = \sum_{t=1}^{H} \gamma^{t-1} \sum_{r_t, s_t, a_t} r_t p(r_t|s_t, a_t) p(s_t, a_t) \tag{2.1}$$

where, H is the horizon, which can either be finite or infinite. To make the total expected reward well-defined in the case of infinite horizon, we introduce a discount factor γ. As long as $0 < \gamma < 1$, the infinite series converges and the total expected reward is well-defined.

The relation to policy π can be made explicit by realizing that

$$p(s_t, a_t) = \sum_{\backslash s_t, \backslash a_t} p(s_{1:H}, a_{1:H}) \tag{2.2}$$

where the joint distribution factorizes as in equation 2.3 owing to the independence statements implicitly associated (e.g. by having separate transition and reward distributions, we implicitly assumed that for a given state-action pair, the resulting reward and transition are mutually independent) with the previously defined sources of randomness in the environment and the agent. These independence assumptions are not always true, but are true (or assumed to be true) in the scenarios considered in this discussion.

$$p(s_{1:H}, a_{1:H}) = p(s_1) \prod_{t=1}^{H-1} \left[p(a_t|s_t) p(s_{t+1}|s_t, a_t) \right] p(a_H|s_H) \tag{2.3}$$

$$= p(s_1) \prod_{t=1}^{H-1} \left[\pi_{a_t, s_t} p(s_{t+1}|s_t, a_t) \right] \pi_{a_H, s_H} \tag{2.4}$$

Here $p(s_1)$ is the distribution over the starting state of the agent, which we have ignored till now.

Now that the relation between the total expected reward and the policy is established, the problem of finding the policy has been reduced to an optimization problem and any optimization algorithm can be used. To provide comparison-context and motivation to find the optimum policy using EM, two other techniques are presented and their limitations are discussed.

2.3 Policy Iteration

Policy Iteration is a common and computationally light-weight approach for finding the optimum policy. It is built upon the crucial assumption that the optimal policy is deterministic and searches for the best policy among the deterministic policies. Combining equations 2.1 an 2.2, we have

$$U(\pi) = \sum_{t=1}^{H} \gamma^{t-1} \sum_{r_t, s_t, a_t} r_t p(r_t|s_t, a_t) \sum_{\backslash s_t, \backslash a_t} p(s_{1:H}, a_{1:H}) \tag{2.5}$$

As the policy and transition distributions are stationary, the expected utility can be written as,

$$U(\pi) = \sum_{s_{1:H}, a_{1:H}} p(s_{1:H}, a_{1:H}) \sum_{t=1}^{H} \gamma^{t-1} \sum_{r_t} r_t p(r_t|s_t, a_t) \tag{2.6}$$

Using simpler notations for expected reward, we have

$$U(\pi) = \sum_{s_{1:H}, a_{1:H}} p(s_{1:H}, a_{1:H}) \sum_{t=1}^{H} \gamma^{t-1} \bar{r}_t \tag{2.7}$$

As the policy is deterministic, the policy distribution terms in 2.3 like $p(a_t|s_t)$ is 0 for all actions and is 1 for the only action that we would be deterministically choosing for a given state. Lets $d(s)$ be the function that deterministically maps states to those actions. Considering this change, the maximum possible total expected reward can be written as

$$\sum_{s_1}\sum_{s_2}\cdots\sum_{s_{H-1}}\sum_{s_H} p(s_1)\prod_{t=2}^{H} p(s_t|s_{t-1}, d(s_{t-1}))\sum_{t=1}^{H}\gamma^{t-1}\bar{r}_t \tag{2.8}$$

The summations can be distributed to efficiently calculate the total expected reward; however, trying to distribute the sums reveals a recurrent pattern as observed in the following equation.

$$\begin{aligned}\sum_{s_1}\cdots\sum_{s_{H-2}} p(s_1)\prod_{t=2}^{H-2} p(s_t|s_{t-1}, d(s_{t-1}))\Bigg(\sum_{t=1}^{H-2}\gamma^{t-1}\bar{r}_t+ \\ \gamma^{H-2}\Big(\sum_{s_{H-1}} p(s_{H-1}|s_{H-2}, d(s_{H-2}))\Big(\bar{r}_{H-1}+\sum_{s_H}\gamma\bar{r}_H p(s_H|s_{H-1}, d(s_{H-1}))\Big)\Big)\Bigg)\end{aligned} \tag{2.9}$$

By defining $f(s_H, a_H) \equiv \bar{r}_H$, the recursive pattern can then be seen as

$$f(s_t, a_t) = \bar{r}(s_t, a_t) + \gamma\sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) f(s_{t+1}, d(s_{t+1})) \tag{2.10}$$

The function $f(s, a)$ is referred to as the value function and can be interpreted as the sum of immediate reward of taking action a at state s and the subsequent discounted rewards assuming that the subsequent actions are chosen as per the deterministic policy given by the function $d(s)$. So, it is also referred to as the value of the policy given by $d(s)$. Note that the value function is stationary as the transition and reward functions are assumed to be stationary.

Now, expressing the total expected reward in terms of the (recursive) value function,

$$U(\pi) = \sum_{s_1} p(s_1) f(s_1, d(s_1)) \tag{2.11}$$

it can be seen that the total expected reward can be maximized by changing $d(s)$ such that

$$d(s) = \arg\max_a f(s, a) \tag{2.12}$$

Now the value of this new policy given by the updated $d(s)$ can be found by iterating equation 2.10 till convergence. Again, the total expected reward can be further increased by updating the policy using equation 2.12 with the new value function. This iterative procedure of starting from a random policy or value function and then alternating between equations 2.10 and 2.12 is called as policy iteration. Policy iteration can be shown to converge at the optimal deterministic policy [1].

Combining equations 2.10 and 2.12 gives the Bellman's optimality condition, which is satisfied at the convergence of policy iteration. This can easily be seen as the max operator in the Bellman's condition won't change the value function f as the converged $d(s)$ would give the same maximum value yielding action (optimal policy).

$$f(s_t, a_t) = \bar{r}(s_t, a_t) + \max_{a_{t+1}} \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) f(s_{t+1}, a_{t+1}) \tag{2.13}$$

The policy iteration updates and the Bellman optimality condition shown here is slightly different form those in textbooks like [1] and [2], but is essentially the same.

Further, it is not absolutely necessary to iterate 2.10 till convergence to find the true value of a policy given by $d(s)$. Just one sweep of update of equation 2.10 for all state-action pairs is sufficient to find good policy updates using arg max function in 2.12. This significantly reduces the computation requirements and the upcoming experiments use this simpler version of policy iteration. It would be referred to as MDP-PI in the rest of the discussion.

2.4 Gradient Ascent

Gradient ascent is the typical choice of optimization these days and can very well be used to maximize the total expected reward by ascending using gradients in the policy space. Starting again from equation 2.7, we have the expression for the total expected reward as,

$$U(\pi) = \sum_{s_{1:H}, a_{1:H}} p(s_{1:H}, a_{1:H}) \sum_{t=1}^{H} \gamma^{t-1} \bar{r}_t \tag{2.14}$$

The calculation can be performed more efficiently by distributing sums and defining recursive functions called “messages” [2]. Defining the function g as follows,

$$g(s_H, a_H) = \bar{r}(s_H, a_H) \tag{2.15}$$

$$g(s_t, a_t) = \bar{r}(s_t, a_t) + \gamma \sum_{a_{t+1}} \sum_{s_{t+1}} p(a_{t+1}|s_{t+1}) p(s_{t+1}|s_t, a_t) g(s_{t+1}, a_{t+1}) \tag{2.16}$$

total expected reward can then be calculated as

$$U(\pi) = \sum_{a_1} \sum_{s_1} p(s_1) p(a_1|s_1) g(s_1, a_1) \tag{2.17}$$

Note that the function g is very similar to the previously defined value function f and is also referred to as the value function. The difference is that, f is the value function for a deterministic policy, whereas g is the value function for just about any policy.

Equation 2.17 lends itself to be represented as a compute graph and packages like `tensorflow` can be used to calculate gradients using automatic differentiation. These gradients can then be used to perform gradient ascent, which usually converges at a local optimum. Policy must be re-parameterized such that it can take values in the entire real space (and later explicitly rectified and normalized) to make it amenable for optimization using gradient ascent. This simple technique to solve MPDs will be referred to as MDP-GD for the rest of the discussion.

Note that the implementation of gradient ascent discussed here is extremely naive and won’t scale well to large problems. Nevertheless it shares similar theoretical characteristics with other state-of-the-art methods based on gradient ascent and hence can serve as a good baseline for comparison in the upcoming experiments. A more efficient method based on stochastic gradient ascent with sampled observations, which can scale well, is discussed in chapter 3 while extending this technique to reinforcement learning.

2.5 Expectation-Maximization

One of the biggest challenges in machine learning is to efficiently calculate (or approximate) integrals and summations. Distributing the sums to efficiently calculate the total expected reward led to the concept of value functions, and subsequently aided the development of policy iteration and gradient ascent techniques. An alternative is to work with a lower bound of the total expected reward instead of the actual value. The lower bound can be expressed in terms of a variational distribution, which paves way for the application of efficient approximation techniques from the variational approximate inference literature. The standard technique for optimizing such a lower bound is the EM algorithm. This section is based on [6] and [7], but is motivated and explained is a slightly different way.

Combining equations 2.1 and 2.2, we have the total expected utility expressed in terms of expected reward function,

$$U(\pi) = \sum_{t=1}^{H} \sum_{s_{1:H}, a_{1:H}} \gamma^{t-1} \bar{r}_t p(s_{1:H}, a_{1:H}) \tag{2.18}$$

Marginalizing out the random variables representing the future states and actions that have no bearing over the reward at time-step t, we have

$$U(\pi) = \sum_{t=1}^{H} \sum_{s_{1:t}, a_{1:t}} \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}) \tag{2.19}$$

The key to finding the lower bound is to introduce a new variational distribution $q(s_{1:H}, a_{1:H}, t)$ in the numerator and denominator and then treat the summation as an expectation based on the q distribution. Note that the variables in the q distribution include states and actions taken in all time-steps and also includes the time-step parameter t. The summations are also adjusted to include the extra (seemingly useless future states and actions) random variables in the newly introduced q distributions. This is done to subsume the concept of summing over all time-steps into the unifying concept of taking expectations. We now have,

$$U(\pi) = \sum_{t=1}^{H} \sum_{s_{1:H},a_{1:H}} \gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}) \frac{q(s_{1:H}, a_{1:H}, t)}{q(s_{1:H}, a_{1:H}, t)} \tag{2.20}$$

$$= \left\langle \frac{\gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t})}{q(s_{1:H}, a_{1:H}, t)} \right\rangle_q \tag{2.21}$$

As log is a monotonically increasing function, we can optimize the total expected reward in log space. So we have

$$\log U(\pi) = \log \left\langle \frac{\gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t})}{q(s_{1:H}, a_{1:H}, t)} \right\rangle_q \tag{2.22}$$

$$\geq \left\langle \log \frac{\gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t})}{q(s_{1:H}, a_{1:H}, t)} \right\rangle_q \tag{2.23}$$

The above equation follows from Jensen's inequality. This lower bound can be maximized using the EM algorithm [8]. One possible disadvantage of this approach is that rewards cannot be negative (as log transformation is not defined for negative values). So, this technique is only suitable for scenarios with positive reward values.

The two unknowns π and q are updated iteratively by the EM algorithm. The step in which q is updated is termed as the E-step and the step in which π is updated is called as the M-step. The two steps are performed alternatively till convergence.

M-step corresponds to the calculation

$$\pi_{a,s} \propto \sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t) \tag{2.24}$$

π is normalized so that it becomes a distribution.

E-step corresponds to the calculation

$$q(s_{1:H}, a_{1:H}, t) \propto \gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}) \tag{2.25}$$

Actually, this calculation is not explicitly performed. Only the un-normalized marginals of q required in the M-step are calculated by "passing messages".

The detailed derivations for this E-step and M-step are in section A.1. This algorithm shares the limitations of EM algorithm and could end up converging at a local optimum like gradient ascent. However, it also shares the advantage of EM in that, in each iteration, the increase in total expected reward is guaranteed [8]. For the rest of this discussion, this application of EM algorithm for solving MDPs would be referred to as MDP-EM algorithm.

2.6 Motivating Expectation-Maximization

Among the three techniques discussed, MDP-EM is the least common method of solving MDPs. However there are certain arenas in which MDP-EM has an advantage over the other techniques. The following experiments are specifically selected to exhibit such advantages of MDP-EM. It must be noted that the advantages are not absolute and only apply to scenarios similar to those discussed here.

2.6.1 Experiment 1: Finding Shortest Path

This experiment compares MDP-PI, MDP-GD and MDP-EM algorithms based on their capacity to completely change the policy, which is initialized at a local minima.

2.6.1.1 Experimental Setup

This experiment is conducted in a discrete deterministic grid-world as shown in the figure 2.2. Each grid in the grid-world corresponds to a state and the state is represented as the absolute location of the agent in the grid-world. There are 4 actions available to the agents: move-north, move-south, move-east and move-west. Each action moves the agent in the corresponding direction by one state deterministically and only one action can be performed in a time-step. Walls hinder motion and attempting to move into a wall will result in no change to the state of the agent. The reward for moving into the goal state is 100. For every other action, irrespective of the state in which it is performed, the reward is zero. The problem is episodic with episodes of length 100 time-steps and all episodes begin with the agent positioned in the start state. Whenever the agent reaches the goal state, it is teleported to the start state and the episode continues seamlessly till the time-step count reaches 100. A discount of 0.9 is assumed while solving the MDP using all three methods.

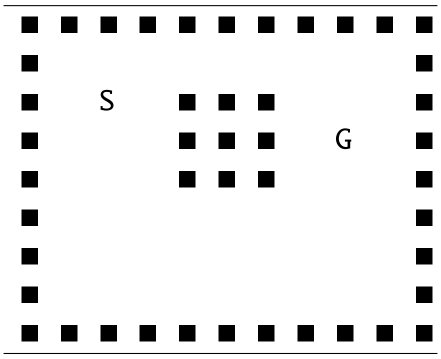


Figure 2.2: Room Grid-World. ■ represents walls. S represents the start state. G represents the goal state.

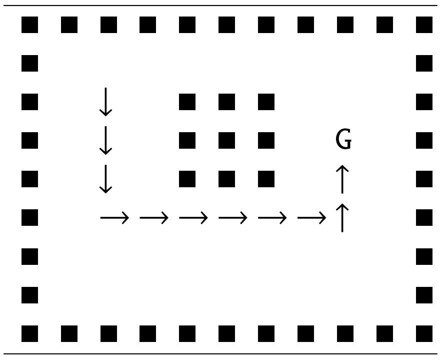


Figure 2.3: Initial Policy. *Deterministic Case:* The arrows represent the action taken in the respective states. In all states without arrows, the deterministic policy is to move west. *Stochastic Case:* Arrows represent the action that would be taken with probability 0.77. Other actions are equally probable. In states without arrows, all actions are equally probable.

The deterministic policy for MDP-PI and the stochastic policy for the rest of the algorithm are initiated as per figure 2.3. MDP-PI was performed for 10 iterations. MDP-EM was performed for 10 and 75 iterations. MDP-GD was performed for 100 iterations with the learning rates: 0.001, 0.01, 0.1 and 1. The resulting (typical) policies are in figure 2.4. The agents' names are coded as `MDP-PI-<niter>`, `MDP-GD-<niter>-<lrate>`, `MDP-EM-<niter>`, where `<niter>` is the number of iterations and `<lrate>` is the learning rate used for gradient descent.

Each policy was evaluated by running the corresponding agent for 100 episodes and finding the average reward per time-step. The experiment was repeated 10 times and the averaged results are reported in figure 2.5

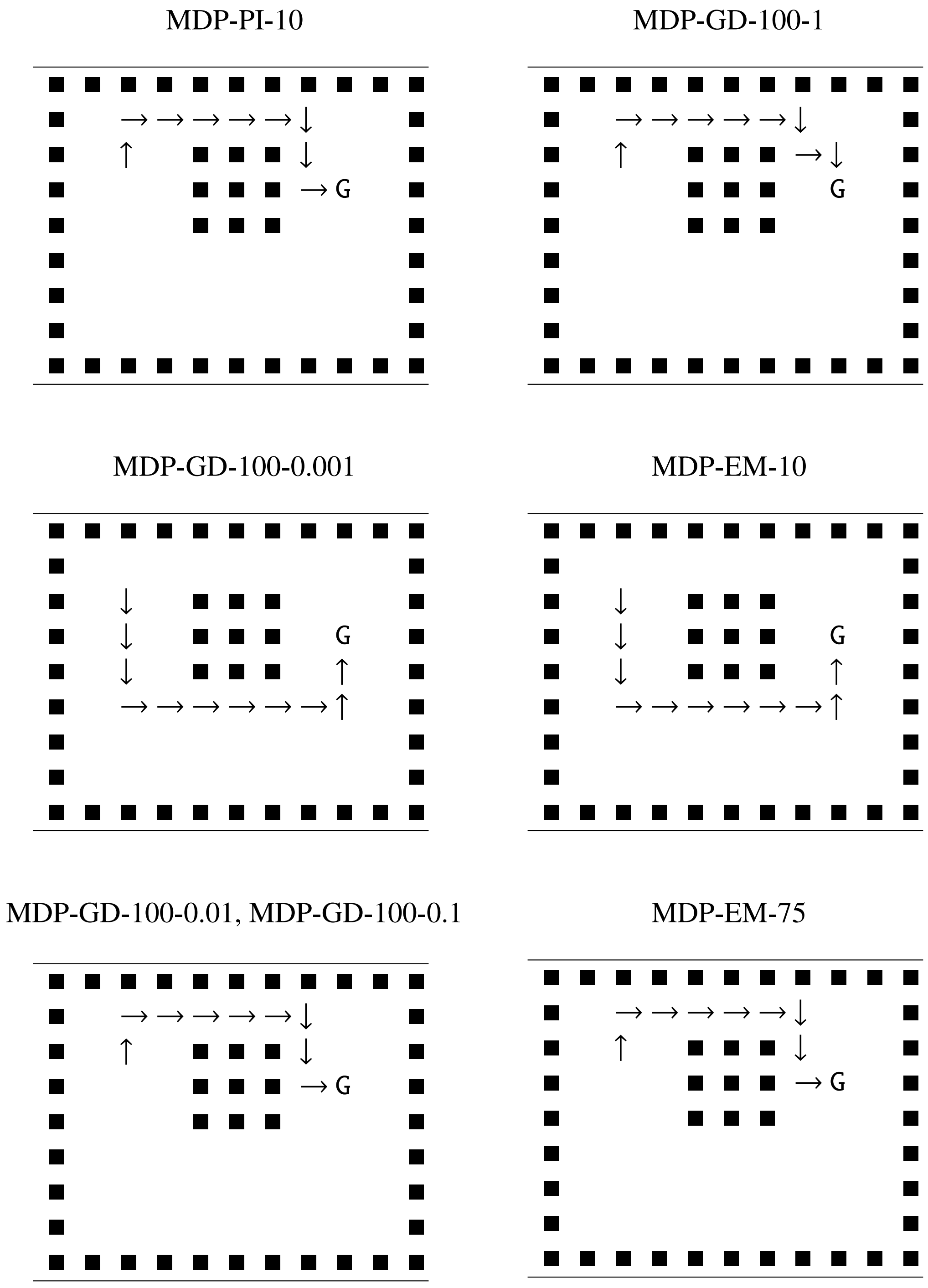


Figure 2.4: Typical Final Policies. Only shows the most likely action from the start state to the goal state.

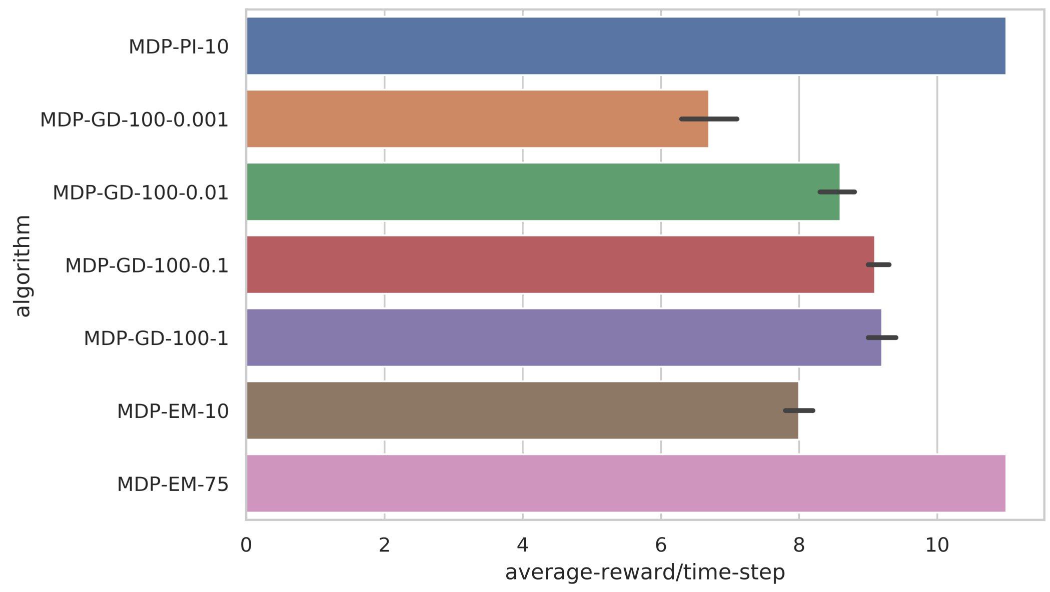


Figure 2.5: Average reward per time-step for each of the agents under study. The black lines indicate the 90% CI for the mean based on 10 repetitions of the experiment.

2.6.1.2 Results and Analysis

Policy iteration is the clear winner in this experiment. In just 10 iterations, it completely changed the initial policy and found the (one of the) best policy that gives the highest average reward per time-step.

Gradient ascent has the inherent limitation of searching in the vicinity of the initial policy. For the conservative learning rate of 0.001, it couldn't escape the local optimum (at which the policy was initialized) even after 100 iterations. Learning rates higher than 0.01 could jump out of the local optimum and find the shortest route; however, they couldn't (within the allowed 100 iterations) converge on the optimal policy of deterministically following the shortest route; thus, leading to a low average reward per time-step.

The EM algorithm does not suffer from the limitations of MDP-GD and is capable of making big changes to the policy in each iteration with a guaranteed increase in total expected reward and doesn't require hyper-parameters like the learning rate. Though it couldn't jump out of the local optimum in 10 iterations, it converged at the optimal policy of (almost) always following the shortest path to the goal. Unlike Policy Iteration, EM has no convergence guarantees and might end up converging at a local optimum; however, in this case it jumped out of the local optimum and converged at the global optimum that is almost as good as the one found by policy iteration.

This experiment was motivated by observations in [4]. It highlights some advantages of MDP-EM over MDP-GD. MDP-EM is shown here to produce almost equally good policies as that of MDP-PI; but, it should be noted that it is more computationally expensive than MDP-PI.

2.6.2 Experiment 2: Learning to Compromise

In the previous experiment, MDP-PI gave the best result with minimal effort; however, searching only for deterministic policies could be a serious limitation if the optimal policy is not deterministic. This experiment is a slightly modified version of the famous experiment used in numerous publications and is designed to highlight the limitation of searching only for deterministic policies. The comparison is between MDP-PI and MDP-EM.

2.6.2.1 Experimental Setup

Again, the experiment is conducted in a discrete deterministic grid-world as shown in the figure 2.6 with the same 4 actions (moving north, south, east and west), each moving the agent by one grid-step in the corresponding direction in each time-step. Again, walls hinder movement as previously described.

The critical change in this environment is that the state is no-longer represented as the absolute location of the agent in the grid-world. The agent can only perceive its state through its limited vision, which is represented as a vector of 4 grids-boxes that are to the immediate north, south, east and west of the agent. Thus, the agent is rendered incapable of distinguishing between the shaded locations in figure 2.6

The problem is again episodic with episodes of length 100 time-steps and the episodes begin with the agent placed randomly in one of the empty states. In the original version of the experiment, the reward for moving into the goal-state is 10 and the penalty for moving into the death-traps is -10. As MDP-EM is incapable of handling negative rewards, this had to be changed. In this experiment, the reward for

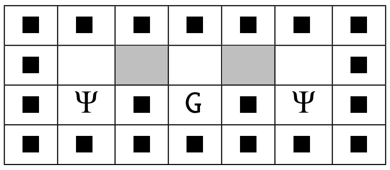
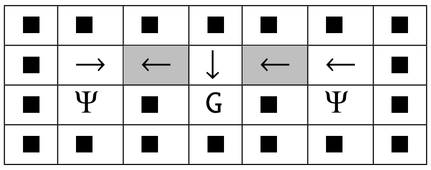
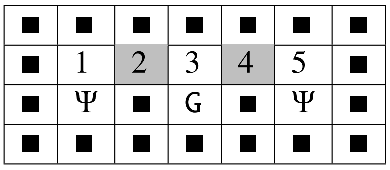


Figure 2.6: Dungeon Grid-World. ■ represents walls. G represents the goal-state and Ψ represents death traps. The agent could start in any one of the empty states. The shaded grid-boxes represent indistinguishable agent states based on its field of vision.

MDP-PI



MDP-EM



Location	1	2	3	4	5
Action: ←	0.00	0.17	0.00	0.17	0.99
Action: →	0.99	0.82	0.00	0.82	0.01
Action: ↑	0.01	0.00	0.00	0.00	0.00
Action: ↓	0.00	0.01	1.00	0.01	0.00

Figure 2.7: Typical Final Policies. The deterministic policy found by MDP-PI is represented with arrows in the 5 empty states. For the stochastic policy found by MDP-EM the probabilities of choosing each of the four actions in each of the numbered locations is shown as a table. For MDP-EM, the policy shown here is a particularly unlucky convergence among the 100 runs of experiments.

moving into the goal state is 20 and there is no penalty for moving into death-traps; however, there is a reward of 10 for every other action. This changes the dynamics of the problem (and gives high rewards for deterministic policies that never reach the goal) but the global optimal policy is still the same. Whenever the agent reaches the goal-state or one of the death-traps, it is teleported randomly to one of the five empty states. Again, a discount of 0.9 is assumed while solving the MDP using both the methods.

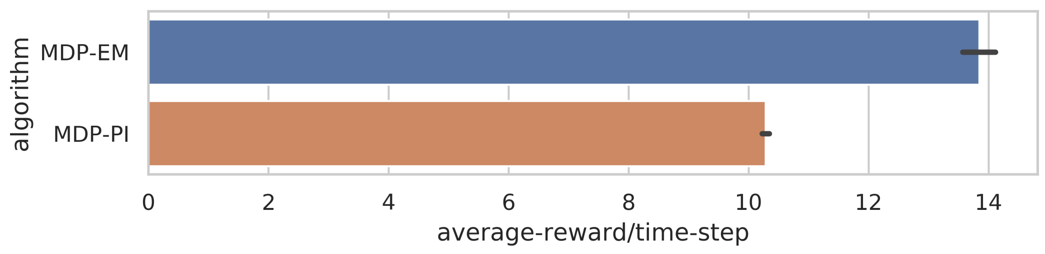


Figure 2.8: Average reward per time-step for the both the agents. The black lines indicate the 90% CI for the mean based on 100 repetitions of the experiment.

The policies were initialized randomly and both MDP-PI and MDP-EM were run for 250 iterations. The resulting final (typical) policies are in figure 2.7. The policies were evaluated for 100 episodes and the average reward per time-step was calculated. The experiment was repeated 100 times and the averaged result is reported in figure 2.8.

2.6.2.2 Results and Analysis

As expected, the deterministic policy underperforms in this case and the advantage of using MDP-EM (or any other method that allows stochastic policies) is obvious. It should also be noted that the final policy (for one among the 100 experiments) of MDP-EM (reported in figure 2.7) is suboptimal as the optimal choice is to select move-east and move-west with 0.5 probability each. This highlights the fact that EM is not guaranteed to converge to the global optimum. Nevertheless, it is certainly better than MDP-PI in this scenario.

2.7 Characteristics of EM

Based on the experimental results, it can be seen that MDP-EM performs better than the other two techniques in certain scenarios. It has to be noted that

- MDP-EM can make big changes to the policy unlike gradient ascent.
- Every EM-update is guaranteed to increase the total-expected reward unlike gradient ascent, as long as the q distribution is evaluated exactly without approximations.
- MDP-EM doesn’t require hyper-parameters like learning rate.

- MDP-EM can work with stochastic policies unlike policy iteration. This helps in cases where the true optimal policy is stochastic. Further, the stochastic policy being improved can be used as the behaviour policy (in RL) as it is inherently exploratory and helps with focused exploration (as opposed to random exploration). This is detailed in the next chapter.

However, it should also be noted that

- MDP-EM might converge at a sub-optimal (locally optimal) policy.
- MDP-EM cannot work with negative rewards.

The techniques discussed in this chapter are extended for use with the general reinforcement learning problem in the next chapter.

Chapter 3

Reinforcement Learning

The last chapter introduced three techniques for solving MDPs and highlighted scenarios in which MDP-EM has an advantage over the other two. This chapter extends all three algorithms for solving the general reinforcement learning problem. The similarities between MDP-EM and the other 2 techniques are explored, which then is used as a justification for extending MDP-EM in ways similar to that of policy iteration and gradient ascent.

The theme of the chapter closely revolves around the notoriously elusive balance between exploration and exploitation in reinforcement learning, and the potentially elegant way of dealing with it using Bayesian inference techniques is highlighted.

Only the vanilla version of Q-learning and actor-critic techniques are introduced in this chapter (as an extension of policy iteration and gradient ascent) to provide context for comparing with and improving MDP-EM while extending it to RL.

3.1 Transition to Reinforcement Learning

A Reinforcement Learning agent (in most general settings) starts to interact with its environment with zero knowledge. It learns a model of the environment (fitting transition and reward distributions) based on its interactions (observed transitions and rewards) and uses an MDP solving technique to find the best policy so as to maximize its rewards. The hard part is to decide how much to explore the environment to get a sufficiently good model of it and when to start exploiting the knowledge to reap as much rewards as possible.

An over-exploring agent might loose precious time-steps (interaction opportunities with the environment) and might end up getting a low total reward. On the other hand, an over-exploiting agent might not have had sufficient knowledge to find the best possible policy and might end up exploiting a sub-optimal policy, again leading to a low total reward.

The holy grail is to have an RL agent that simultaneouly carries out well-balanced exploration and exploitation in a wide range of scenarios.

3.2 Q-learning

It is not always necessary to explicitly learn the transition and reward distributions from the observed transitions and rewards. The method of policy iteration is extended to RL by devising a technique to learn the value function for state-action pairs directly instead of learning transition and reward distributions. The techniques that focus on learning the value function are generally classified as value-based methods. Historically [1], value functions were represented using a function q (we use f and g in this discussion), which led to the name Q-learning.

The idea of Q-learning is to learn the value function with experience, while implicitly treating the problem as an MDP and iteratively improving the policy in each time-step, as and when new experience is gathered i.e. exploration and exploitation is done simultaneously. The Bellman's optimality condition 2.13 (repeated below) matches this goal.

$$f(s_t, a_t) = \bar{r}(s_t, a_t) + \max_{a_{t+1}} \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) f(s_{t+1}, a_{t+1}) \tag{3.1}$$

When this Bellman's condition is not satisfied, (like most recursive conditions) it serves as an update towards the condition. So, random initial values can be updated in each time-step incorporating new observations, which leads to the update

$$f_{t+1}(s_t, a_t) = \bar{r}(s_t, a_t) + \max_{a_{t+1}} \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) f_t(s_{t+1}, a_{t+1}) \tag{3.2}$$

Though the expectations required for the update cannot be evaluated, they can be simply replaced with the empirical expectations with just one sample of reward and transition observed in each time-step. Note that the expectations become increasingly reliable estimates of the true expectations with each new added observation. However, the reward and transitions need not be explicitly stored and a moving average update can instead be used as follows.

$$f_{t+1}(s_t, a_t) = f_t(s_t, a_t) + \omega\big(\tilde{r}(s_t, a_t) + \max_{a_{t+1}} \gamma f_t(\tilde{s}_{t+1}, a_{t+1}) - f_t(s_t, a_t)\big) \tag{3.3}$$

where $\tilde{r}$ is the observed reward and $\tilde{s}$ is the observed transition. ω is taken as $1/t$. Other values of ω can be used for non-stationary problems, which is not considered in this discussion.

The last thing to consider is the behaviour policy i.e. the policy that the agent would follow to interact with the environment while making these updates. Following the optimal deterministic policy (based on the current un-converged value function using equation 2.12) can lead to high rewards. However, it must be noted that the Bellman equation based updates effectively rely on equations 2.10 and 2.12 and the equation 2.10 is an update over all state-action pairs. So, to get the same convergence guarantees of policy iteration, the behaviour policy must not deterministically avoid visiting any of the states, else the value function would converge to that of a sub-optimal policy. This is generally taken care of by using an ϵ-greedy policy that usually selects the optimal action based on current estimates of values and would select a random action with probability ϵ. This is how exploration and exploitation is traditionally balanced in Q-learning. Note that ϵ is hyper-parameter and the agent has no means of selecting or adjusting it leading to reduced rewards in the long term.

3.3 Actor-Critic Gradient Ascent

The Gradient Ascent technique described in the section 2.4 is computationally expensive and hence won't scale well to problems with more state-action possibilities. Instead of using the actual gradient, an unbiased estimate of the gradient can be used for ascending the policy space, which yields a more efficient stochastic gradient approach. The observations in each time-step can directly be used for estimating the gradient, thus naturally extending the MDP algorithm to reinforcement learning. Again, the observations are used to directly estimate the gradient, which in turn is used to alter the policy, without explicitly fitting the transition or reward distribution. Such methods that directly find a policy without explicitly fitting a model of the environment are generally classified as policy-based methods.

Starting with the expression of total expected reward in terms of value functions g as given in equation 2.17, we have,

$$U(\pi) = \sum_{a_1}\sum_{s_1} p(s_1)p(a_1|s_1)g(s_1,a_1) \tag{3.4}$$

Taking derivatives with respect to policy on both sides and by applying the product rule,

$$\nabla U(\pi) = \sum_{a_1}\sum_{s_1} p(s_1)\Big(\nabla p(a_1|s_1)g(s_1,a_1) + p(a_1|s_1)\nabla g(s_1,a_1)\Big) \tag{3.5}$$

Using the score function trick and changing to expectations notations,

$$\nabla U(\pi) = \sum_{a_1}\sum_{s_1} p(s_1)\frac{p(a_1|s_1)}{p(a_1|s_1)}\nabla p(a_1|s_1)g(s_1,a_1) + \Big\langle \nabla g(s_1,a_1)\Big\rangle_{p(s_1,a_1)} \tag{3.6}$$

$$= \Big\langle \nabla \log p(a_1|s_1)g(s_1,a_1)\Big\rangle_{p(s_1,a_1)} + \Big\langle \nabla g(s_1,a_1)\Big\rangle_{p(s_1,a_1)} \tag{3.7}$$

Unrolling the recursive definition of the value function g (equation 2.16) and using the score function trick again (and then repeatedly for each unrolled step),

$$\begin{aligned}\nabla U(\pi) =& \Big\langle \nabla \log p(a_1|s_1)g(s_1,a_1)\Big\rangle_{p(s_1,a_1)} + \\ & \gamma\Big\langle \big\langle \nabla \log p(a_2|s_2)g(s_2,a_2)\big\rangle_{p(s_2,a_2)} + \big\langle \nabla g(s_2,a_2)\big\rangle_{p(s_2,a_2)}\Big\rangle_{p(s_1,a_1)}\end{aligned} \tag{3.8}$$

$$\begin{aligned}=& \Big\langle \nabla \log p(a_1|s_1)g(s_1,a_1)\Big\rangle_{p(s_1,a_1)} + \\ & \gamma\Big\langle \nabla \log p(a_2|s_2)g(s_2,a_2)\Big\rangle_{p(s_2,a_2)} + \gamma\Big\langle \nabla g(s_2,a_2)\Big\rangle_{p(s_2,a_2)}\end{aligned} \tag{3.9}$$

$$= \sum_{t=1}^{H}\Big\langle \gamma^{t-1}\nabla \log p(a_t|s_t)g(s_t,a_t)\Big\rangle_{p(s_t,a_t)} \tag{3.10}$$

As the discount is a constant and as the transition, reward and policy distributions are stationary, we have

$$\nabla U(\pi) \propto \Big\langle \nabla \log p(a|s) g(s,a) \Big\rangle_{p(s,a)} \tag{3.11}$$

Now this expectation can be approximated with just one sample of observed state and action taken (as per a behaviour policy) in each time step. The proportionality constant can be absorbed into learning rate parameter of the gradient descent algorithm, while making the updates to policy in each time-step. The value function g of the current policy can be updated in parallel using update equations very similar to that of Q-learning. The value update is called as a critic and the policy update is called as an actor, hence the name actor-critic.

$$g_{t+1}(s_t, a_t) = g_t(s_t, a_t) + \omega\big(\tilde{r}(s_t, a_t) + \gamma g_t(\tilde{s}_{t+1}, a_{t+1}) - g_t(s_t, a_t)\big) \tag{3.12}$$

The above equation can approximately be considered as a extension to Q-learning, albeit without the convergence guarantees. They are discussed under the context of value function approximation and TD-learning in [1], but are not introduced here as they are not immediately pertinent to this discussion. Similarly the variance of the gradient estimates can be reduced by control variate methods, which again is not detailed here. But, introducing the control variate $v(s) = \sum_a p(a|s)g(s,a)$ gives

$$\nabla U(\pi) \propto \Big\langle \nabla \log p(a|s)\big[g(s,a) - v(s)\big] \Big\rangle_{p(s,a)} \tag{3.13}$$

This reduces the variance of the estimate [9].

Usually, the policy being optimized is itself used as a behaviour policy, and as the policy under consideration is stochastic it could reach all state action pairs (eventually) to give good estimate for the value function g used in the updates. Further the policy being optimized is closed to optimal, so when used as the behaviour policy, it could lead to a more focused exploration. However to prevent premature convergence, a regularization term is added to $U(\pi)$ that ensures some continued exploration and prevents the policy being updated from getting too exploitatory. Usually the regularization term is the entropy of the policy $(-\langle logp(a|s)\rangle_{p(a|s)})$. Similar to the ϵ in ϵ-greedy policies, the regularization co-efficient (a variable adjusting the relative weight of the regularization term as in the general supervised learning setting) is still a hyper-parameter, which cannot be decided by the agent for itself.

Once again, this exposes the need for improvement in exploration-exploitation balance.

3.4 EM with Maximum Likelihood Estimates

Both Q-learning and actor-critic methods bypassed learning the model explicitly and instead used the observations to learn the value function and in turn make either implicit (Q-learning) or explicit (actor-critic) updates to the policy. However, the MDP-EM algorithm needs the transition and reward distributions for performing the E-step. Techniques that explicitly fit the reward and transition distributions are generally classified as model-based methods.

The Maximum Likelihood (ML) approach is a common technique for fitting distributions using observations, in which, the parameters of the distributions are selected such that the probability of getting the observations is maximized. For the case of finite rewards and finite states, both the reward and the transition distribution can be represented using a categorical distribution. Note that these are sets of conditional distributions for each given state-action pairs and are independent of each other. Categorical distributions, for example the transition distribution, are parameterized by a set of parameters $\boldsymbol{\theta}$, such that $\theta_{s,a}^{s'}$ gives the probability of transitioning to s' when action a is performed in state s and it follows that $0 < \theta_{s,a}^{s'} < 1$ and $\sum_{s'} \theta_{s,a}^{s'} = 1$.

The ML method of finding these θ values is to count the transitions for each state-action pairs and normalizing them separately. A short proof that this gives the maximum likelihood solution can be found in [2]. Methods of fitting other distributions using ML method can also be found in [2].

The ML estimates can be combined with the EM algorithm using the method proposed in [7]. Starting with a random policy, all the transitions and rewards in an episode can be collected and ML estimates of the parameters of the transition and reward distribution can be evaluated. With this ML estimate of the model, the RL problem can momentarily be treated as an MDP and MDP-EM can be used to find the optimal policy. With this optimal policy as the behaviour policy, observations from the next episode can be accumulated and the ML estimate of the model can be updated. With this updated model, MDP-EM can again be used to find the optimal policy, which would be used as the behaviour policy for the next episode. The same process is repeated for subsequent episodes and the policy is improved after each episode. This simple way of extending MDP-EM to RL is referred to as ML-EM in this discussion.

Similar to the actor-critic methods, the behaviour policy in this case is the same as the policy being updated. The stochasticity of the policy is used for focused exploration. However, there is no regularization term. Further, the ML estimates of the model are assumed to be absolutely true and the uncertainty associated with the estimates is neglected while applying the MDP-EM algorithm. So, the policies found by this method are usually over-exploitatory. Nevertheless, as explained at the end of this chapter, Bayesian variants of this technique hold the potential of introducing a more natural and flexible method of motivating the agent to explore.

3.5 Similarities between Approaches

ML-EM might seem very different from the Q-learning and actor-critic methods, but is actually closely related as highlighted in works like [4] and [5].

Equations 2.24 and 2.25 correspond to the bulk of the EM algorithm. In the E-step, we have

$$q(s_{1:H}, a_{1:H}, t) \propto \gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}) \tag{3.14}$$

from which we calculate the necessary marginals (explained in appendix A) by

$$q(s_\tau = s, a_\tau = a, t) \propto \sum_{\backslash s_\tau, \backslash a_\tau} q(s_{1:H}, a_{1:H}, t) \tag{3.15}$$

$$\propto \sum_{\backslash s_\tau, \backslash a_\tau} \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}) \tag{3.16}$$

Based on the above equation it can be observed that the marginal $q(s_\tau = s, a_\tau = a, t)$ can be interpreted (in the un-normalized state) as the expected (discounted) reward received at time-step t, when the agent is in state s and takes action a in one of the previous time-steps τ.

In the M-step (repeated below), we further sum this quantity over all possible time-steps t and the correspondingly possible previous time-steps τ. This quantity is the total expected (discounted) reward for the agent to be in state s and take action a, irrespective of the time-step. This is quite similar to our expectations from the value function g, introduced in gradient ascent and learnt in the critic part of the actor-critic technique.

$$\pi_{a,s} \propto \sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t) \tag{3.17}$$

The M-step further uses this representation of state-action value to update the policy, and is capable of changing the policy by a huge extent, unlike the policy updates made in actor-critic. In fact, the policy update closely resembles the update (equation 2.12) of MDP-PI; however, instead of taking the best possible action, the M-step assigns a distribution based on the relative values associated with each action for a given state.

It can thus be seen that the EM based algorithm has the advantage of both MDP-PI (big changes to policy) and actor-critic gradient ascent (stochastic policies). However, the ML-EM algorithm differs from both Q-learning and actor-critic in that the updates are made only at the end of episodes and the E- and M-steps are iterated till convergence.

So, the following experiment is designed to compare the standard version of

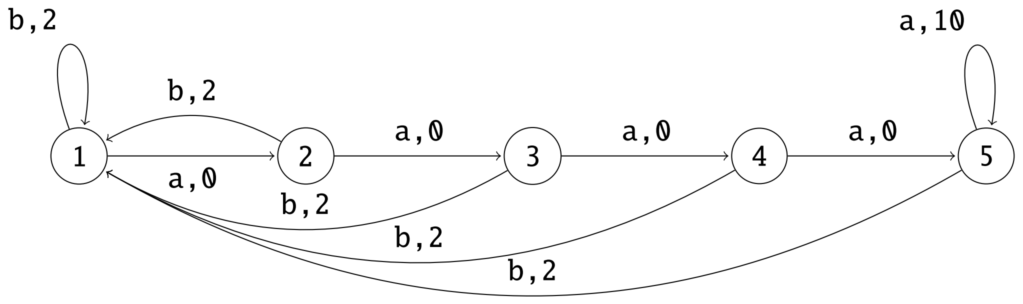


Figure 3.1: Slippery Slope Environment. The states are numbered from 1 to 5. Action a corresponds to move-right and action b corresponds to move-left. The rewards associated with the action are also shown. While the reward for action taken in guaranteed, the effect of actions on transition is flipped 20% of the time.

ML-EM and a slight variant named Semi-EM, in which only one E- and M-step is performed but after each time step, using fresh ML estimates of transition and reward distribution parameters updated after each time-step. So, Semi-EM updates the policy after each time-step, but the policy is not allowed to converge as only one E- and M-step is performed.

3.5.1 Experiment 3: Semi-Converging EM

This experiment compares ML-EM and Semi-EM in a simple stochastic environment.

3.5.1.1 Experimental Setup

The experiment is conducted in a small environment with 5 states and 2 actions. This is a slightly modified version of the one used in [7]. The 5 states are shown in figure 3.1 and the two actions are move-left and move-right.

Moving left takes the agent to the left-most state and gives a reward of 2. Moving right moves the agent to the right by one state and gives no reward. When the agent is in the right most state, then moving right retains the agent in the right most state and gives a reward of 10.

Actions result in the intended (as previously described) state transitions only 80% of the time. The remaining 20% of the time, the state transition associated with the opposite action takes place. Note that this stochasticity is only for state transitions and the agent gets the rewards for the actions taken deterministically. The

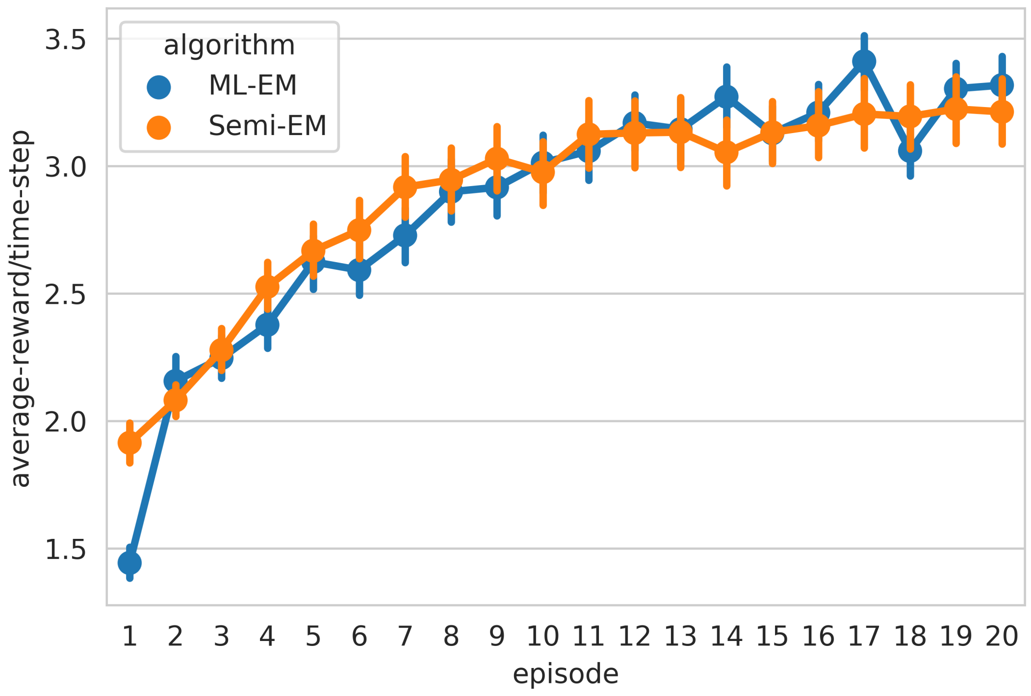


Figure 3.2: Average reward per time-step for both agents. The vertical lines indicate the 90% CI for the mean based on 200 repetitions of the experiment.

reward and transition distributions are independent.

For the sake of simplicity, the agents are made aware of the true reward distribution and only the parameters of transition distribution are learned from the experience using ML estimates.

The problem is episodic of length 100 time-steps and the episodes always begin with the agent in the left most state. This starting distribution is also supplied to the agents and need not be learned. No discounts were considered for both the agents. The policies were initialized randomly and the performance of the both the techniques were tracked for 20 episodes. ML-EM is performed at the end of each episode till convergence, which corresponds to the condition in which the Frobenius norm of difference in policies between EM-updates fall below 0.01. Semi-EM preforms one E- and M-step after each time-step, as previously discussed.

The results in terms of average reward per time-step were recorded at the end of each episode. The experiment was repeated 200 times and the averaged result is reported in figure 3.2. The typical policies at the end of 20 episodes is in figure 3.3.

ML-EM (Typical Policy)

State	1	2	3	4	5
$\rightarrow$	0.33	0.80	0.99	1.00	1.00
$\leftarrow$	0.67	0.20	0.01	0.00	0.00

Semi-EM (Typical Bad Policy)

State	1	2	3	4	5
$\rightarrow$	0.00	0.00	0.00	0.00	1.00
$\leftarrow$	1.00	1.00	1.00	1.00	0.00

Semi-EM (Typical Good Policy)

State	1	2	3	4	5
$\rightarrow$	1.00	1.00	1.00	1.00	1.00
$\leftarrow$	0.00	0.00	0.00	0.00	0.00

Figure 3.3: Typical final policies (found by simply eyeballing). The probabilities associated with the actions are given in the table.

3.5.1.2 Results and Analysis

From figure 3.2 it seems that there is not much difference in performance between the two variants; however, the policies shown in figure 3.3 reveal a crucial difference. The typical good policy at the end of 20 episodes using Semi-EM is the best possible global optimal policy. The typical final policies for ML-EM almost seldom end up being the globally optimal policy. Though Semi-EM finds the best possible policy, it sometimes end up with a really bad shortsighted policy, which brings the average reward closer to that of ML-EM, thus hiding the erratic supremacy of Semi-EM.

The reason for this behaviour is not clear. The current guess is that, ML-EM updates till the policy converges after each episode, which could end up being a locally optimal policy as EM doesn't have convergence guarantees to reach the global optimum. A converged policy explores less and might just lead to new observations that would reinforce the misinterpreted supremacy of a locally optimal policy. Thus ML-EM might find it very hard to snap out of this delusion and might end up with a reasonably good but locally optimal policy.

On the other hand, Semi-EM does just one EM-update, leading to un-converged policies, which might be expected to explore more. However, it might also converge faster (and hence based on fewer observations) than ML-EM as every episode has 100 EM updates (one per time-step), whereas ML-EM typically needs less than 50 iterations to meet the previously defined convergence criteria; and, highly converged policies are difficult to change as observations gathered by following such policy tend to support that policy. The current guess is that these two aspects of Semi-EM compete and when exploration wins, we get a really good policy, and when faster convergence wins, we get an acutely short sighted policy.

Though the results of this experiment is vague and inconclusive, it is clear that exploration is important and must be systematically introduced into EM based techniques.

3.6 Motivation for a Bayesian Approach

The need for extra-exploration is clear as the methods discussed are all naturally slightly exploitatory. In addition this extra-exploration must be adjustable so that a exploration-exploitation balance can be achieved leading to better rewards.

The regularization used in actor-critic gradient ascent is practically similar to ϵ-greedy policy used in Q-learning. In both cases, the policy is prevented from getting too deterministic by regularizing it directly instead of saving the model from overfitting[1]. This is inconsistent with the ethos of traditional supervised learning. Further, such direct policy regularization wastes time-steps as it is uninformed about the nature of uncertainty in the model and hence is incapable of using it for guided exploration.

A Bayesian approach can at times be thought of as a replacement for regularization [10] in supervised learning. By having Bayesian updates instead of ML updates for the reward and transition distributions, the uncertainty in the parameter estimates can be captured, which is expected to be transferred to the perceived (relative) value of state-action pairs. The policies based on such uncertainty-adjusted values are expected to explore to an extent commensurate with the degree of uncer-

[1]The term overfitting is used loosely here to refer to the scenario in which a ML-fitted model harshly assumes that certain transitions are impossible just because they were not observed.

tainty. This might lead to better exploration-exploitation balance. Further, Bayesian approaches provide a systematic way to incorporate prior information about the environment model before even starting to make updates.

EM based techniques are generally amenable to Bayesian extensions and the possibilities are discussed in the next chapter.

Chapter 4

Bayesian Reinforcement Learning

The previous chapter extended the MDP-EM algorithm into a model-based RL technique with ML estimates of the model. Hoping to achieve a better exploration-exploitation balance, this chapter introduces a variant of the algorithm that uses Bayesian estimates of the model parameters.

The inevitable computational intractabilities associated with the Bayesian approach are discussed and the effectiveness of a factorized approximation for the variational distribution in the E-step of EM algorithm is reviewed. Further, a Monte-Carlo approximation for the marginals of the variational distribution is proposed and its effectiveness is analyzed.

4.1 EM with Bayesian Estimates

While extending MDP-EM to ML-EM, the parameters of the transition distribution were estimated using the Maximum-Likelihood (ML) approach, which as previously discussed lacks information about the uncertainty of the estimates. The crux of the Bayesian approach is to maintain a distribution over the model parameter values so that its uncertainty is well defined throughout the process of estimation. For the sake of simplicity of exposition, the reward distribution is assumed to be known and the transition distribution's parameter $\boldsymbol{\theta}$ is learned from experience, but the technique presented here can easily be extended to include an unknown reward distribution by integrating over the additional reward parameters in equation 4.4 and by introducing an additional factor for reward parameters in equation 4.9 As the parameter values should satisfy the requirements of the parameters ($0 < \theta_{s,a}^{s'} < 1$ and $\sum_{s'} \theta_{s,a}^{s'} = 1$) of the categorical distribution, it is assumed that each set of parameter values (for each independent conditional categorical distribution given s and a) follow a Dirichlet distribution and that all these Dirichlet distributions are independent.

Thus, the probability of $\boldsymbol{\theta}$ before observing any data (prior) can be represented as

$$p(\boldsymbol{\theta}) = \prod_{s,a} \mathrm{Dir}(\theta_{s,a}^{\cdot} | \alpha_{s,a}^{\cdot}) \tag{4.1}$$

where, α is very similar to the counts discussed in the Maximum-Likelihood approach. It can even be shown that the mean of this distribution is the same as the ML estimate when $\alpha = c$. Thus the value of α can be appropriately set to reflect the initial belief. The strength of belief can also be varied as higher counts (with same proportions) represent more certain beliefs.

By rules of conditional probability, we have

$$p(\boldsymbol{\theta}|\zeta) \propto p(\zeta|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{4.2}$$

where, ζ is the set of observations made. As the functional form of categorical and Dirichlet distributions have similarities (conjugateness [2]), the probability distribution (posterior) over the parameters after observing the transitions (and calculating counts c) in the environment can be updated as

$$p(\boldsymbol{\theta}|\zeta) = \prod_{s,a} \mathrm{Dir}(\theta_{s,a}^{\cdot} | \alpha_{s,a}^{\cdot} + c_{s,a}^{\cdot}) \tag{4.3}$$

Now, this distribution over $\boldsymbol{\theta}$ is used instead of the ML estimate of $\boldsymbol{\theta}$ in each run of the EM algorithm, which can either be after each time-step or each episode depending on the flavour of the RL algorithm. Subsequent Bayesian updates can be made by substituting $\alpha^{\text{new prior}} = \alpha^{\text{old posterior}} + c^{\text{old posterior}}$ and then finding the new posterior using the new counts.

Similar to ML-EM, using the current estimate of $\boldsymbol{\theta}$, the RL problem can momentarily be considered as an MDP and the total expected reward can be maximized using the EM algorithm. As we have a distribution of $\boldsymbol{\theta}$ values instead of just one value, the total expected reward now includes an additional expectation over the posterior distribution on $\boldsymbol{\theta}$. The value being optimized is now,

$$U(\pi) = \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\zeta) \sum_{s_{1:H},a_{1:H}} p(s_{1:H}, a_{1:H}|\boldsymbol{\theta}) \sum_{t=1}^{H} \gamma^{t-1}\bar{r}_t \tag{4.4}$$

Note that the dependence of $p(s_{1:H}, a_{1:H})$ on $\boldsymbol{\theta}$ is shown explicitly as $\boldsymbol{\theta}$ is now a random variable. Again as in section 2.5, a variational distribution $q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta})$ can be introduced to get a lower bound on the total expected reward. Note that the q distribution now includes $\boldsymbol{\theta}$ as one of its random variables so that all integrations and summations can be represented by an expectation. Again, marginalizing out unnecessary variables from p and introducing q in numerator and denominator,

$$U(\pi) = \int_{\boldsymbol{\theta}} \sum_{t=1}^{H} \sum_{s_{1:H},a_{1:H}} \gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\zeta) \frac{q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta})}{q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta})} \tag{4.5}$$

$$= \left\langle \frac{\gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\zeta)}{q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta})} \right\rangle_q \tag{4.6}$$

$$\log U(\pi) \geq \left\langle \log \frac{\gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\zeta)}{q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta})} \right\rangle_q \tag{4.7}$$

Once again, the bound can be optimized iteratively using the M-Step and the E-Step as in the traditional EM algorithm. In fact, the M-step is exactly as in the previous case (equation 2.24 -repeated below). The M-Step derivation follows exactly as per section A.1, as the additional term introduced $p(\boldsymbol{\theta}|\zeta)$ is independent of π. So, we once again have the M-Step update

$$\pi_{a,s} \propto \sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t) \tag{4.8}$$

Unlike ML-EM, the $q(s_\tau = s, a_\tau = a, t)$ in the above equation is arrived at by marginalizing $\boldsymbol{\theta}$ out and hence would lead to a policy that is mindful of the uncertainty in theta, which is expected to implicitly take care of exploration, as explained in [7]. But, performing the E-step and finding the marginal is not as simple as in the previous case as now the variational distribution includes $\boldsymbol{\theta}$ and the expectation introduces a very high dimensional integration over $\boldsymbol{\theta}$, which makes computations intractable. Techniques from the approximate variational inference literature can be used to circumvent this shortcoming.

4.2 EM with Factorized Approximation

In E-Step, we search for a distribution q that maximizes the bound (tightens it) on the total expected reward. To make the computations tractable (as suggested in [7]), it can be assumed that the variational distribution factorizes into two variational distributions as follows

$$q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta}) = q(s_{1:H}, a_{1:H}, t)q(\boldsymbol{\theta}) \tag{4.9}$$

The problems associated with this approximation can be seen immediately as the assumption implies that the distribution of $\boldsymbol{\theta}$ and the distribution of state-actions are independent. However, as will be seen in equations 4.10 and 4.11, the interdependence of the two distributions (via E-step updates) implicitly forces the marginals of $q(s_{1:H}, a_{1:H}, t)$ to account for uncertainties in $\boldsymbol{\theta}$ and the product of the distributions will match the joint distribution as close as possible. Though the expressibility of this factorized approximation is limited, it might still lead to reasonably good approximations as it is actually a more structured variant of the mean field approximation, which is known to provide reasonably good results [11].

The two factors can be treated as two unknowns and can be updated alternatively by holding the other constant, such that the bound the maximized. These updates can be carried out for a fixed number of iterations or till convergence, just like the overall EM algorithm. The update equations are derived in section A.2 and are listed below.

$q(s_{1:H}, a_{1:H}, t)$ is updated as

$$q(s_{1:H}, a_{1:H}, t) \propto \gamma^{t-1}\bar{r}_t p(s_1)p(a_t|s_t)\prod_{\tau=1}^{t-1} p(a_\tau|s_\tau)e^{\langle \log \theta_{s_\tau, a_\tau}^{s_{\tau+1}} \rangle_{q_{\boldsymbol{\theta}}}} \tag{4.10}$$

$q(\boldsymbol{\theta})$ is updated as

$$q(\boldsymbol{\theta}) = \prod_{s,a} \mathrm{Dir}(\theta^{\cdot}_{s,a} | \alpha^{\cdot}_{s,a} + c^{\cdot}_{s,a} + r^{\cdot}_{s,a}) \tag{4.11}$$

where

$$r^{s'}_{s,a} = \sum_{t=2}^{H} \sum_{\tau=1}^{t-1} q(s_{\tau+1} = s', s_\tau = s, a_\tau = a, t) \tag{4.12}$$

Upon convergence of the afore mentioned E-step updates, the M-step is done as usual by updating the policy as per equation 4.8. This algorithm would be referred to as VF-EM in the rest of this discussion.

4.2.1 Experiment 4: Analyzing the suitability of factorization

This experiment compares ML-EM and VF-EM to assess the performance of the Bayesian approach with factorized approximation.

4.2.1.1 Experimental Setup

The experimental setup is the same as that of experiment 3, illustrated in figure 3.1 and detailed in section 3.5.1.1. Again the agents are made aware of the reward distribution and only the parameters of the transition distribution is learned from experience.

The Dirichlet prior over the transition distribution parameters are taken such that $\alpha^{\cdot}_{\cdot,\cdot} = 1$ to have an uninformed, weak prior that assumes that all transitions are equally probable. Further, inside the E-step of VF-EM, 10 updates are performed on the variational factors. The ML-EM algorithm was executed exactly as described in experiment 3. Both the algorithms are performed at the end of each episode (after updating model parameters) till convergence, which is again is considered as the condition in which the Frobenius norm of difference in policies between EM-updates fall below 0.01.

The performance of both the algorithms were tracked for 20 episodes. The typical final policies are reported in figure 4.2. The results in terms of average reward per time-step were recorded at the end of each episode. The experiment was repeated 200 times and the averaged result is reported in figure 4.1.

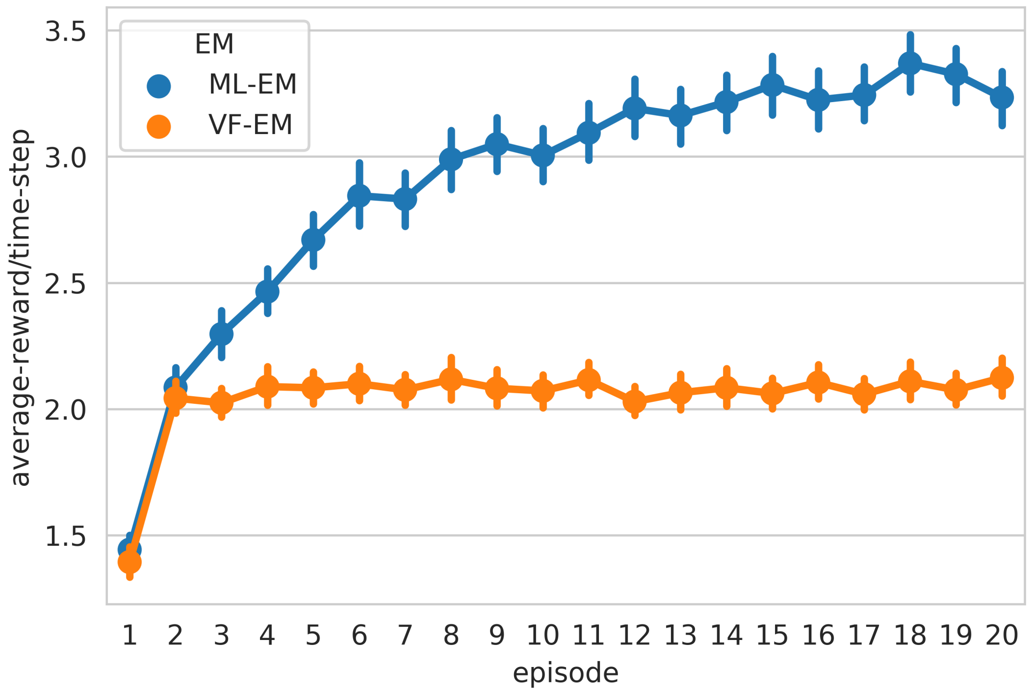


Figure 4.1: Average reward per time-step for both agents. The vertical lines indicate the 90% CI for the mean based on 200 repetitions of the experiment.

ML-EM (Typical Policy)

State	1	2	3	4	5
→	0.33	0.80	0.99	1.00	1.00
←	0.67	0.20	0.01	0.00	0.00

VF-EM (Typical policy)

State	1	2	3	4	5
→	0.00	0.00	0.00	0.00	1.00
←	1.00	1.00	1.00	1.00	0.00

Figure 4.2: Typical final policies (found by simply eyeballing). The probabilities associated with the actions are given in the table.

4.2.1.2 Results and Analysis

It can be seen that ML-EM performs much better than VF-EM. However, this does not discredit the proposed superiority of the Bayesian approach [7]. Instead, this is a confirmation of the unsuitability of the proposed factorized approximation.

Looking at the final policies, it can be seen that the policies found by VF-EM is acutely shortsighted. This shortsightedness can be attributed to the peculiarities of the digamma function [7] used to evaluate the expectations in the E-step (equation 4.10).

Note that the only difference between equation 4.10 and equation 2.25 (after expanding the joint distribution as per equation 2.3) is that equation 4.10 has the un-normalized distributions

$$\tilde{\theta}^{s'}_{s,a} = e^{\langle \log \theta^{s_{\tau+1}}_{s_\tau, a_\tau} \rangle_{q_{\boldsymbol{\theta}}}} = \frac{e^{\psi(\alpha^{s'}_{s,a} + c^{s'}_{s,a} + r^{s'}_{s,a})}}{e^{\psi(\sum_{s'} \alpha^{s'}_{s,a} + c^{s'}_{s,a} + r^{s'}_{s,a})}} \tag{4.13}$$

instead of the normalized distributions $\theta^{s'}_{s,a} = p(s'|s,a)$. ψ is the digamma function. As shown in figure 4.3, the digamma function evaluates slightly lower but almost equal to that of the log function. Further, for smaller values the evaluations of the digamma function is much lower. So, we have $\sum_{s'} \tilde{\theta}^{s'}_{s,a} < 1$. Therefore, while calculating the variational density in equation 4.10, the rewards in the initial time-steps of an episode are implicitly given more importance. This forced the algorithm to converge on a short-sighted policy.

Note that the root cause of this problem is the factorized approximation. Other approximations might offer better policies.

4.3 EM with Monte-Carlo Approximation

The factorized approximation described in section 4.2 can be categorized as deterministic as the approximation doesn't introduce any stochasticity in the EM algorithm. The only source of randomness is be the agent's experience. An alternative is to use non-deterministic approximations, where the approximation is stochastic and adds an additional source of randomness to the overall algorithm. Such approximations are common in machine learning, and are generally grouped together as

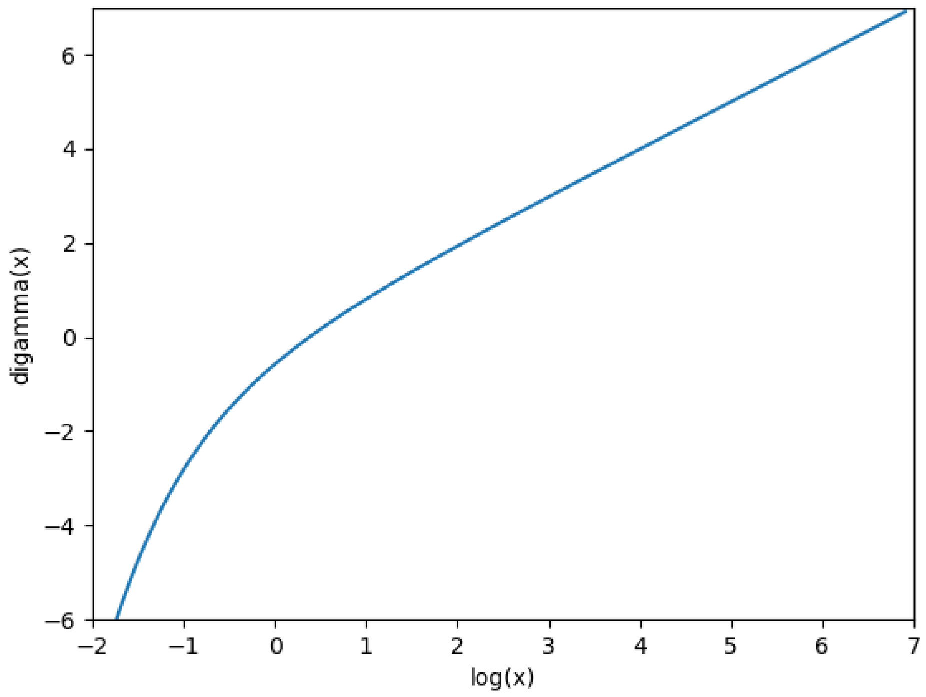


Figure 4.3: Comparing the evaluations of digamma and log functions for values in the range (0.01,100)

Monte-Carlo methods.

There are numerous ways of employing non-deterministic approximations while optimizing our lower bound on the log of total expected reward (equation 4.7 repeated below).

$$\log U(\pi) \geq \left\langle \log \frac{\gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\zeta)}{q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta})} \right\rangle_q \tag{4.14}$$

The expectation in the above equation (the bound itself) can be approximated using a Monte-Carlo estimate. The Monte-Carlo estimate of the expectation can found by evaluating the function with numerous samples from the q distribution and by taking the arithmetic mean of all such evaluations. To make such an estimate, q distribution is generally constrained to have a parametric form such that it can be easily sampled (and reparameterized[1]) and the necessary log-probabilities could be easily calculated. The estimated bound can be directly optimized using

[1]The distribution should be amenable to the reparameterization trick [12] so that the gradient could flow back and alter the parameters of the distribution.

gradient ascent, where the optimal policy and the corresponding parameters of the q distribution (values) are found simultaneously. This technique is usually referred to as Stochastic Gradient Variational Bayes (SGVB) in the literature [12] [13]. The special case, where the q distribution serves as an approximation for the posterior distribution over latent variables (unlike our case), the bound can be further simplified and the resulting algorithm is called as the Variational Auto-Encoder (VAE), which is quite famous in the deep learning community.

However, SGVB can't be directly applied in the current scenario as

- The q distribution is a joint distribution over discrete and continuous variables, and cannot be easily represented using parametric distributions that are suitable for the reparameterization trick [12].
- Using gradient ascent to find the optimal policy limits the algorithm in ways similar to that of MDP-GD as previously highlighted in section 2.6.1.2. The algorithm looses its resemblance with policy iteration and becomes incapable of making big changes to the policy as it would be limited by the learning rate.

Hoping to retain the advantages of the EM algorithm and noting that the M-step (equation 4.8) is not intractable, the exact nature of intractability can be tackled by trying to derive E-step in a way similar to that in section A.1.

In the E-step, for a fixed policy, the bound can be maximized (tightened) by having

$$q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta}) \propto \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t} | \boldsymbol{\theta}) p(\boldsymbol{\theta} | \zeta) \tag{4.15}$$

The marginals required in the M-step ($q(s_\tau = s, a_\tau = a, t)$) can be found by marginalizing out the unnecessary variables as follows.

$$q(s_\tau = s, a_\tau = a, t) \propto \int_{\boldsymbol{\theta}} \sum_{\backslash s_\tau, \backslash a_\tau} q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta}) \tag{4.16}$$

$$\propto \int_{\boldsymbol{\theta}} \sum_{\backslash s_\tau, \backslash a_\tau} \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t} | \boldsymbol{\theta}) p(\boldsymbol{\theta} | \zeta) \tag{4.17}$$

$$\propto \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta} | \zeta) \sum_{\backslash s_\tau, \backslash a_\tau} \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t} | \boldsymbol{\theta}) \tag{4.18}$$

The summation over the states and actions can be found efficiently by passing messages [2]. The relevant α and β expressions (for a given value of $\boldsymbol{\theta}$) can be found in section A.1 in equations A.12 and A.13.

Now, the marginals can be written as

$$q(s_\mathcal{T} = s, a_\mathcal{T} = a, t) \propto \pi_{a,s}\gamma^{t-1} \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\zeta)\alpha_\mathcal{T}(s)\beta_{t-\mathcal{T}}(s,a) \tag{4.19}$$

$$\propto \pi_{a,s}\gamma^{t-1}\Big\langle \alpha_\mathcal{T}(s)\beta_{t-\mathcal{T}}(s,a)\Big\rangle_{p(\boldsymbol{\theta}|\zeta)} \tag{4.20}$$

The source of intractability is this expectation taken with respect to the posterior distribution over the transition probability distribution's parameters. This posterior has a dimension much lower than the joint q distribution and the expectation can be approximated with a Monte-Carlo estimate with relatively fewer samples. Further, the posterior is a Dirichlet distribution (as previously noted) and hence can easily be sampled from, leading to an efficient approximation. The validity of the approximation can be intuitively appreciated as it is just a weighted average of value estimates for different probable values of $\boldsymbol{\theta}$. Further, this weighted average naturally accounts for the uncertainty in the estimate of $\boldsymbol{\theta}$ through the sampling process. So, this EM algorithm with Monte-Carlo approximation in E-step could lead to better policies.

Similar to ML-EM, these EM-updates can be performed till convergence after each episode with the Bayesian updates to model parameter distributions done after each episode. Such an algorithm is referred to as M-EM in this discussion.

Further, similar to the Semi-EM algorithm that does ML updates after each time-step, the Bayesian updates can also be done after each time-step with just one iteration of EM-updates per time-step. Such an algorithm is referred to as Semi-M-EM in this discussion.

4.3.1 Experiment 5: Bayesian vs Non-Bayesian RL

In this experiment, the newly proposed M-EM and Semi-M-EM are compared with their maximum likelihood counterparts: ML-EM and Semi-EM.

4.3.1.1 Experimental Setup

The experimental setup is very similar to that of experiment 3, explained in section 3.5.1.1. However, the reward distribution is slightly altered as shown in figure 4.4. Again there are 5 states (as numbered in the figure) and two actions: move-left and move-right. The rewards associated with state-action pairs are deterministic, whereas the effect of actions on transitions is flipped 20% of the time, making the environment stochastic.

Again the agents are made aware of the reward distribution and only the parameters of the transition distribution are learned from experience in both the ML and Bayesian cases. The problem is similarly episodic with episodes of length 100, with the agent always starting in the first state. This information about the start distribution is also supplied to the agent as before, and need not be learned.

In both ML-EM and M-EM, parameter updates and EM updates to policy were made at the end of episodes and the EM updates were made till convergence, which is again considered as the condition at which the Frobenius norm of the difference in policy between EM-updates fall below 0.01. In both Semi-EM and Semi-M-EM, the parameter updates and just one E- and M-update were made after each iteration. In M-EM and Semi-M-EM, 100 samples from the posterior distribution over $\boldsymbol{\theta}$ were used to evaluate the Monte-Carlo estimate of the expectation. Similar to experiment 4, for the Bayesian variants, the Dirichlet (weak, uninformed) prior over the transition distribution parameters are taken such that $\alpha_{\cdot,\cdot}^{\cdot} = 1$. Again no discounts were considered.

The performance of the algorithms were tracked for 20 episodes and the average reward per time-step is recorded at the end of every episode. The experiment was repeated 200 times and the averaged result is reported in figure 4.5. The typical policies found at the end of 20 episodes are reported in figure 4.6.

4.3.1.2 Results and Analysis

It can be seen that Semi-M-EM outperforms the other three algorithms by a large margin.

ML-EM's performance is very similar to the previous experiments with the

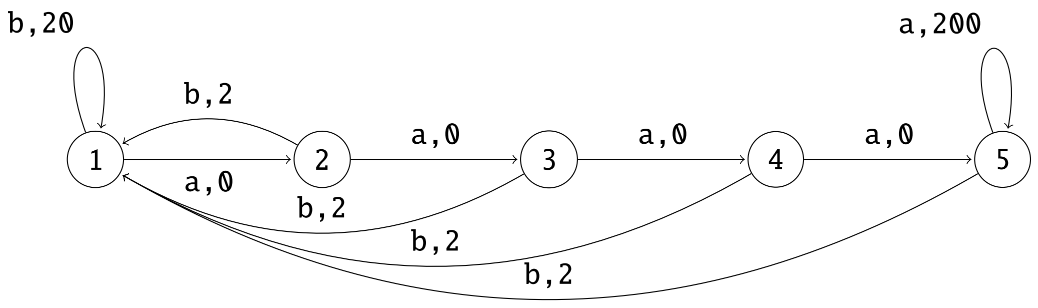


Figure 4.4: Slippery Slope Environment. The states are numbered from 1 to 5. Action a corresponds to move-right and action b corresponds to move-left. The rewards associated with the action are also shown. While the reward for action taken is guaranteed, the effect of actions on transition is flipped 20% of the time.

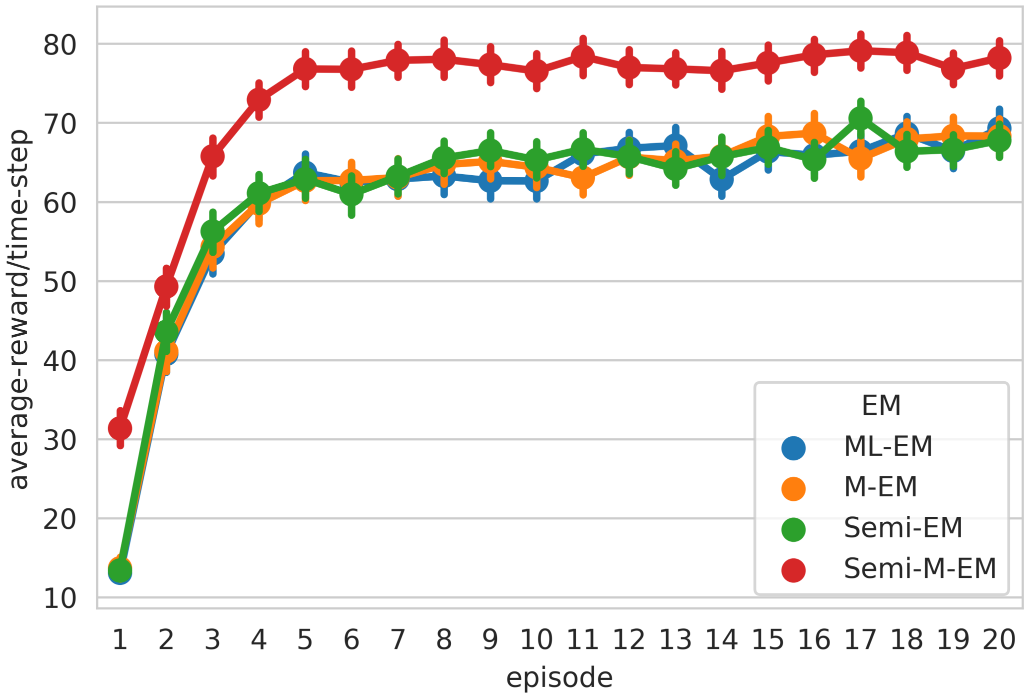


Figure 4.5: Average reward per time-step for all four agents. The vertical lines indicate the 90% CI for the mean based on 200 repetitions of the experiment.

ML-EM (Typical Policy)

State	1	2	3	4	5
$\rightarrow$	0.33	0.80	0.99	1.00	1.00
$\leftarrow$	0.67	0.20	0.01	0.00	0.00

Semi-EM (Typical Policy)

State	1	2	3	4	5
$\rightarrow$	0.33	0.80	0.99	1.00	1.00
$\leftarrow$	0.67	0.20	0.01	0.00	0.00

M-EM (Typical Policy)

State	1	2	3	4	5
$\rightarrow$	0.33	0.80	0.99	1.00	1.00
$\leftarrow$	0.67	0.20	0.01	0.00	0.00

Semi-M-EM (Typical –Almost Always– policy)

State	1	2	3	4	5
$\rightarrow$	1.00	1.00	1.00	1.00	1.00
$\leftarrow$	0.00	0.00	0.00	0.00	0.00

Figure 4.6: Typical final policies (found by simply eyeballing). The probabilities associated with the actions are given in the table.

same typical final policies. As expected, it converges at a local optimum, partially because of previously described insufficient exploration.

This time, Semi-EM performs very similar to ML-EM and converges at policies similar to that of ML-EM. The reason could be that the modified reward distribution provides a high reward for the agents to stay in the initial state, and the extra exploration contributed by the non-converging updates is apparently insufficient to find the optimum policy as before.

Not so surprisingly, M-EM, despite the Bayesian updates again performs very similarly to ML-EM with the same typical policies after 20 episodes. This can be attributed to the fact that the Bayesian estimates[2] and ML estimates are very close

[2]For direct comparison with ML estimates, mode or mean of the posterior can be used. However, the fact of interest is that the posterior gets increasingly peaked at the ML estimate when the number of observations is so high that it dwarves the effect of the prior.

as the number of observations increase. Even the 100 observations in an episode is apparently high enough in this case to overcome the influence of the prior. The EM updates converge at a suboptimal policy, which in turn influences subsequent observations, making it very difficult for EM to escape the local optimum.

On the other hand, Semi-M-EM performs Bayesian updates after each time-step. The prior is strong enough during initial steps as the number of observation is not yet high enough to dwarf the effect of prior. So, the exploration promoted by the Bayesian updates and the non-converging EM-updates happened to be just right for the algorithm to find the optimal policy. In fact, the algorithm converged at the optimal policy in more than 99.5% of the 200 experiments conducted. Further, Semi-M-EM starts to update the policy (although with importance to exploration) right after the first step, thereby providing a higher cumulative reward in the episodes, thus making its use more desirable.

Chapter 5

Conclusion

This work highlighted some of the possible applications of variational methods in RL. EM algorithm was introduced as a method to solve Markov decision processes. Empirically, using experiments 1 and 2, it was shown that EM performs better than policy iteration and gradient ascent in certain scenarios. Unlike gradient ascent, EM could make big changes to the policy, and unlike policy iteration, EM could work with stochastic policies.

EM was used in the general RL problem by using ML estimates of transition and reward distribution parameters. Realizing that the overconfidence of ML estimates lead to under-exploring policies, a Bayesian alternative, which considers the uncertainty in the model-parameter estimates was introduced. The computational intractabilities associated with the Bayesian approach were analyzed, and the need for efficient approximations to the variational distribution (and its marginals) was established. The problems associated with a factorized approximation were reviewed and a Monte-Carlo approximation was proposed.

Appreciating the similarities between the EM algorithm and policy iteration, a semi-converging variant of the EM algorithm was proposed. In this variant, the model parameters were updated after each time-step and only one E- and M-step is performed using these updated parameters in each time-step. Using Bayesian updates for the model parameters with this variant of EM resulted in the Semi-M-EM algorithm, which significantly outperformed all other variants of EM in experiment 5. It is also worthwhile to note that the chain environment used in experiment 5 is specially designed such that it is hard for other RL algorithms like Q-learning and actor-critic techniques to find the global optimal policy. Nevertheless, Semi-M-EM managed to find the global optimal in almost all repetitions of the experiment. However,

- The experiments detailed in this report were specifically selected to highlight the potential of these alternate RL techniques. The proposed algorithms should be tested on a wider range of problems to understand specific limitations and to work on appropriate improvements.

- Q-learning and actor-critic methods are computationally efficient. Whereas, EM based techniques, particularly Semi-M-EM is computationally expensive owing to its reliance on Monte-Carlo approximations. It is certainly necessary to search for alternative approximation techniques that could render Bayesian model-based EM as efficient as the other RL techniques.

- The "semi" variants of EM algorithm were motivated by empirical observations, and the apparent advantage of using them is not yet supported by theoretically sound results. A much deeper analysis on the ways by which these "semi" variants affect the dynamics of the EM algorithm should be conducted.

- All the problems considered in this work have discrete states and actions. Ways of extending the EM based algorithms to continuous domain problems must also be explored.

- This work focussed only on environments with stationary transition and reward distributions. Possibilities of extending the algorithms to non-stationary cases must be surveyed.

- Techniques like actor-critic methods are capable of working with complex environment models, which involve non-linearities parametrized using neural-networks. Capability of accommodating such non-linearities in the proposed EM based approaches is yet to be explored.

Future work could be focussed on the afore mentioned needs.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[2] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[4] Peter Dayan and Geoffrey E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Comput.*, 9(2):271–278, February 1997.

[5] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 945–952, New York, NY, USA, 2006. ACM.

[6] Thomas Furmston and David Barber. Solving deterministic policy (po)mdps using expectationmaximisation and antifreeze. In *European Conference on Machine Learning (LEMIR workshop*, 2009.

[7] Thomas Furmston and David Barber. Variational methods for reinforcement learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 241–248, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[9] Hado Van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement learning*, pages 207–251. Springer, 2012.

[10] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[11] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[12] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations*. ICRL, 2014.

[13] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[14] DeepMind. The pycolab game engine. `https://github.com/deepmind/pycolab`, 2018.

Appendix A

Detailed Derivations

A.1 E-step and M-step Derivation for Section 2.5

The lower bound (B) in equation 2.23 (repeated below) has two unknowns: the variational-distribution q and the policy π.

$$B = \langle \log \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}) \rangle_q - \langle \log q(s_{1:H}, a_{1:H}, t) \rangle_q \tag{A.1}$$

M-Step

The total expected reward is maximized by varying policy π for a fixed q. This can be done in the traditional way of solving for stationary points algebraically with the constrain that π parameterizes a set of conditional distributions. It should also be noted that the only term in B that depends on π are the factors of $p(s_{1:\tau}, a_{1:\tau})$ that are of form $p(a_t|s_t)$. Every other term that always evaluate to a constant value can be neglected during optimization. Thus the bound simplifies to

$$B = \sum_{t=1}^{H} \sum_{s_{1:H}} \sum_{a_{1:H}} q(s_{1:H}, a_{1:H}, t) \sum_{\tau=1}^{t} \log p(a_\tau|s_\tau) + \text{constant} \tag{A.2}$$

Moving around summations and marginalizing out unnecessary variables from the variational distribution, we have

$$B = \sum_{t=1}^{H} \sum_{\tau=1}^{t} \sum_{s_\tau} \sum_{a_\tau} q(s_\tau, a_\tau, t) \log p(a_\tau|s_\tau) + \text{constant} \tag{A.3}$$

This gives use the Lagrangian (one multiplier for each possible state s in the environment)

$$L = \sum_{t=1}^{H} \sum_{\tau=1}^{t} \sum_{s_\tau} \sum_{a_\tau} q(s_\tau, a_\tau, t) \log p(a_\tau | s_\tau) - \sum_{s} \lambda_s \Big[\sum_{a} p(a|s) - 1 \Big] \tag{A.4}$$

Finding the saddle point by solving $\dfrac{\partial L}{\partial \pi_{a,s}} = 0$ and $\dfrac{\partial L}{\partial \lambda_s} = 0$, we get

$$\pi_{a,s} = \frac{\sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t)}{\lambda_s} \tag{A.5}$$

$$\sum_{a} \pi_{a,s} = 1 \tag{A.6}$$

Summing over a on both sides of equation A.5 and substituting A.6, it can be seen that

$$\lambda_s = \sum_{a} \sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t) \tag{A.7}$$

Therefore, π can simply be updated as

$$\pi_{a,s} \propto \sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t) \tag{A.8}$$

and normalized such that it is a distribution.

E-Step

In the E-Step, the marginals $q(s_\tau = s, a_\tau = a, t)$ are found such that the bound B is maximized. From equation A.1, it can be seen that, if

$$q(s_{1:H}, a_{1:H}, t) \propto \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}) \tag{A.9}$$

the bound becomes the tightest possible, i.e. $B = U(\pi)$. As $U(\pi)$ is independent of the variational distribution q, the bound cannot be increased any further and the equation A.9 is the best possible update for q.

The marginals $q(s_\tau = s, a_\tau = a, t)$ can be found by marginalizing out the unnecessary variables in the following way.

$$q(s_\tau = s, a_\tau = a, t) \propto \sum_{\backslash s_\tau, \backslash a_\tau} q(s_{1:H}, a_{1:H}, t) \tag{A.10}$$

$$\propto \sum_{\backslash s_\tau, \backslash a_\tau} \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}) \tag{A.11}$$

Expanding as per equation 2.3, and with α and β expressions defined in the following way,

$$\alpha_\tau(s) = \begin{cases} p(s_1) & \text{if } \tau = 1 \\ \sum_{s',a'} \alpha_{\tau-1}(s') p(a'|s') p(s|s', a') & \text{otherwise} \end{cases} \tag{A.12}$$

$$\beta_{t-\tau}(s, a) = \begin{cases} \bar{r}(s, a) & \text{if } t - \tau = 0 \\ \sum_{s',a'} \beta_{t-\tau-1}(s', a') p(a'|s') p(s'|s, a) & \text{otherwise} \end{cases} \tag{A.13}$$

the marginals can be written as

$$q(s_\tau = s, a_\tau = a, t) \propto \pi_{a,s} \gamma^{t-1} \alpha_\tau(s) \beta_{t-\tau}(s, a) \tag{A.14}$$

Again, these expressions are slightly different from those reported in [6]; but, are nevertheless valid.

A.2 E-step Derivation for Section 4.2

Now, the bound to be maximized (or just tightened) in this E-step is

$$B = \langle \log \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\zeta) \rangle_q - \langle \log q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta}) \rangle_q \tag{A.15}$$

and we have assumed that

$$\begin{aligned} q(s_{1:H}, a_{1:H}, t, \boldsymbol{\theta}) &= q(s_{1:H}, a_{1:H}, t) q(\boldsymbol{\theta}) && \text{(A.16)} \\ &= q(x) q(\boldsymbol{\theta}) && \text{(A.17)} \\ &= q_x q_{\boldsymbol{\theta}} && \text{(A.18)} \end{aligned}$$

where x is a simple vector representing the state-actions and time-steps for notational convenience. Similarly, q_x and $q_{\boldsymbol{\theta}}$ are also simply redefined notations.

Optimization can be carried out using the co-ordinate ascent technique, where q_x and $q_{\boldsymbol{\theta}}$ are updated alternatively holding the other constant.

Updating $q(\boldsymbol{\theta})$

If $q(\theta)$ is the only unknown in the bound, then we have

$$B = \Big\langle \langle \log \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) \rangle_{q_x} + \log p(\boldsymbol{\theta}|\zeta) \Big\rangle_{q_{\boldsymbol{\theta}}} - \Big\langle \log q(\boldsymbol{\theta}) \Big\rangle_{q_{\boldsymbol{\theta}}} + \text{constant} \tag{A.19}$$

$$= -KL(q(\boldsymbol{\theta}) || \frac{1}{Z} e^{\langle \log \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) \rangle_{q_x} + \log p(\boldsymbol{\theta}|\zeta)}) + \text{constant} \tag{A.20}$$

where, Z is the normalizing constant for the distribution. As KL-Divergence cannot be negative [2], the following is the best possible update which makes the KL-Divergence 0.

$$q(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\zeta) e^{\langle \log \gamma^{t-1} \bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta}) \rangle_{q_x}} \tag{A.21}$$

Removing multiplicative constants that do not depend on theta, we have

$$q(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\zeta) e^{\langle \sum_{\tau=1}^{t-1} \log \theta_{s_\tau, a_\tau}^{s_{\tau+1}} \rangle_{q_x}} \tag{A.22}$$

Marginalizing out unnecessary variables in q_x,

$$q(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\zeta)e^{\sum_{t=2}^{H}\sum_{\tau=1}^{t-1}\sum_{s_\tau}\sum_{a_\tau}\sum_{s_{\tau+1}} q(s_{\tau+1},s_\tau,a_\tau,t)\log\theta_{s_\tau,a_\tau}^{s_{\tau+1}}} \tag{A.23}$$

Comparing the argument of the exponential function with that of the exponential family form of Dirichlet distribution ($p(\theta) = h(\theta)e^{\alpha^T T(\theta)-A(\alpha)}$), it can be seen that $q(s_{\tau+1}, s_\tau, a_\tau, t) = \alpha_{s,a}^{s'}$, as the innermost summation is just the dot-product. As $p(\boldsymbol{\theta}|\zeta)$ is again a Dirichlet distribution, we have the update

$$q(\boldsymbol{\theta}) = \prod_{s,a} \mathrm{Dir}(\theta_{s,a}^{\cdot}|\alpha_{s,a}^{\cdot} + c_{s,a}^{\cdot} + r_{s,a}^{\cdot}) \tag{A.24}$$

where

$$r_{s,a}^{s'} = \sum_{t=2}^{H}\sum_{\tau=1}^{t-1} q(s_{\tau+1} = s', s_\tau = s, a_\tau = a, t) \tag{A.25}$$

Updating $q(x)$

If $q(x)$ is the only unknown in the bound, then we have

$$B = \Big\langle \langle \log \gamma^{t-1}\bar{r}_t p(s_{1:t}, a_{1:t}|\boldsymbol{\theta})\rangle_{q_{\boldsymbol{\theta}}} \Big\rangle_{q_x} - \Big\langle \log q(x) \Big\rangle_{q_x} + \text{constant} \tag{A.26}$$

$$B = -KL(q(x)||\frac{1}{Z}e^{\langle \log \gamma^{t-1}\bar{r}_t p(s_{1:t},a_{1:t}|\boldsymbol{\theta})\rangle_{q_{\boldsymbol{\theta}}}}) + \text{constant} \tag{A.27}$$

where Z is the normalizing constant for the distribution. As KL-Divergence cannot be negative [2], the following is the best possible update which makes the KL-Divergence 0.

$$q(x) \propto e^{\langle \log \gamma^{t-1}\bar{r}_t p(s_{1:t},a_{1:t}|\boldsymbol{\theta})\rangle_{q_{\boldsymbol{\theta}}}} \tag{A.28}$$

$$q(s_{1:H}, a_{1:H}, t) \propto \gamma^{t-1}\bar{r}_t p(s_1)p(a_t|s_t)\prod_{\tau=1}^{t-1} p(a_\tau|s_\tau)e^{\langle \log \theta_{s_\tau,a_\tau}^{s_{\tau+1}}\rangle_{q_{\boldsymbol{\theta}}}} \tag{A.29}$$

The expression $e^{\langle \log \theta_{s_\tau,a_\tau}^{s_{\tau+1}}\rangle_{q_{\boldsymbol{\theta}}}}$ can be evaluated using standard digamma (ψ) functions by noting that

$$\langle \log \theta^{s'}_{s,a} \rangle_{q_{\boldsymbol{\theta}}} = \psi(\alpha^{s'}_{s,a} + c^{s'}_{s,a} + r^{s'}_{s,a}) - \psi(\sum_{s'} \alpha^{s'}_{s,a} + c^{s'}_{s,a} + r^{s'}_{s,a}) \tag{A.30}$$

Defining the α and β expressions in the following way

$$\alpha_\tau(s) = \begin{cases} p(s_1) & \text{if } \tau = 1 \\ \sum_{s',a'} \alpha_{\tau-1}(s') p(a'|s') e^{\langle \log \theta^{s}_{s',a'} \rangle_{q_{\boldsymbol{\theta}}}} & \text{otherwise} \end{cases} \tag{A.31}$$

$$\beta_{t-\tau-1}(s) = \begin{cases} \sum_{a} \bar{r}(s,a) p(a|s) & \text{if } t - \tau - 1 = 0 \\ \sum_{s',a'} \beta_{t-\tau-2}(s') p(a'|s) e^{\langle \log \theta^{s'}_{s,a'} \rangle_{q_{\boldsymbol{\theta}}}} & \text{otherwise} \end{cases} \tag{A.32}$$

The marginals required for updating $q(\boldsymbol{\theta})$ can be expressed as follows

$$q(s_{\tau+1} = s', s_\tau = s, a_\tau = a, t) \propto \gamma^{t-1} \alpha_\tau(s) p(a|s) e^{\langle \log \theta^{s'}_{s,a} \rangle_{q_{\boldsymbol{\theta}}}} \beta_{t-\tau-1}(s') \tag{A.33}$$

Once the iterative updates for $q(\boldsymbol{\theta})$ and $q(x)$ converge, the marginals required for performing the M-step can be calculated.

Marginals required for the M-step

Defining the α and β expressions in the following way

$$\alpha_\tau(s) = \begin{cases} p(s_1) & \text{if } \tau = 1 \\ \sum_{s',a'} \alpha_{\tau-1}(s') p(a'|s') e^{\langle \log \theta^{s}_{s',a'} \rangle_{q_{\boldsymbol{\theta}}}} & \text{otherwise} \end{cases} \tag{A.34}$$

$$\beta_{t-\tau}(s,a) = \begin{cases} \bar{r}(s,a) & \text{if } t - \tau = 0 \\ \sum_{s',a'} \beta_{t-\tau-1}(s',a') p(a'|s') e^{\langle \log \theta^{s'}_{s,a} \rangle_{q_{\boldsymbol{\theta}}}} & \text{otherwise} \end{cases} \tag{A.35}$$

The marginals required for M-step can be expressed as follows.

$$q(s_\tau = s, a_\tau = a, t) \propto \pi_{a,s} \gamma^{t-1} \alpha_\tau(s) \beta_{t-\tau}(s,a) \tag{A.36}$$

Appendix B

Code

A copy of the programs used to run the experiments is hosted at `https://github.com/sajaysurya/ucl_msc_csml_project`

Apart from the standard scientific computing python packages, the code depends on pre-existing works like `pycolab` [14]. Please refer to `README.md` in the repository for a complete list of dependencies and other relevant details.