Aggregation-Based Learning in the Inverted Pendulum Problem

Gerald van den Berg

Advised by Prof. Warren Powell

Program in Applied and Computational Mathematics

Princeton University

Contents

Abstract

We consider the problem of adapting approximate dynamic programming techniques to the inverted pendulum task. This is a particularly challenging task as we work with a relatively uninformative reinforcement signal and have no a priori information about our system. Success in this task requires an effective solution to the credit assignment problem, incorporation of noisy and biased information into our belief about the system, and efficient learning. We use an aggregation-based Bayesian prior to estimate our value function and explore the performance of the knowledge gradient algorithm relative to other policies. We deal with the credit assignment problem through the use of a decaying trace of the reinforcement signal. Although this updating mechanism violates some assumptions of traditional learning models, we find that the knowledge gradient policy is effective in improving performance.

1 Introduction

The task of balancing an inverted pendulum (also known as the pole-balancing problem) was originally used to demonstrate a series of conventional control techniques. Roberge was one of the first to show that we can develop effective controls despite the presence of nonlinearities and the inherently unstable nature of the system (Roberge, 1960). The problem has since been used to demonstrate the effectiveness of reinforcement learning techniques ranging from genetic algorithms to neural networks, and there has been an increased focus on finding effective algorithms for this task with little or no prior systems information. The problem becomes significantly more difficult under these conditions because the algorithm now needs to develop a policy while learning about the underlying dynamics of the system.

The goal of the inverted pendulum task is to balance a pendulum on a cart while ensuring that the cart remains within certain bounds on a track. We balance the system by applying either a leftward or rightward fixed-magnitude force. The system is reset every time the pendulum either falls over or the cart goes off of the track.

The state of our system S_t is given by four variables: θ_t (rad), x_t (m), $\dot{\theta}_t$ (rad/s), and $\dot{x}_t$ (m/s). θ is the angle of the pole relative to an upright position, and $\dot{\theta}$ is the angular velocity. The position of the cart is given by x, and $\dot{x}$ is the velocity of the cart. An illustration of the system is given below in Figure 1.

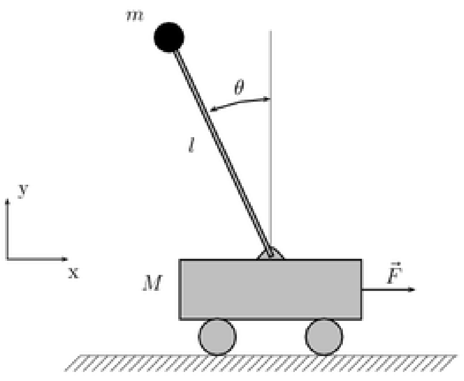


Figure 1: The Inverted Pendulum System

The dynamics of the system are determined by a number of parameters, which we give in Table 1. These parameters are consistent with the standard values used when working on this problem. The angular and horizontal acceleration at time t are given by:

$$\begin{aligned}
\ddot{\theta}_t &= \frac{mg\sin\theta_t - \cos\theta_t\left[F_t + m_p l\dot{\theta}_t^2\sin\theta_t\right]}{(4/3)ml - m_p l\cos^2\theta_t}, \\
\ddot{x}_t &= \frac{F_t + m_p l\left[\dot{\theta}_t^2\sin\theta_t - \ddot{\theta}_t\cos\theta_t\right]}{m}.
\end{aligned}$$

We use a discrete time Euler approximation of the dynamics with a time-step of size $\tau = 0.02s$ to simulate the system, which leads to the following updating equations.

F_t	± 10 N	force provided at time t
m	1.1 kg	mass of the cart-pendulum system Position of the cart (m)
m_c	1.0 kg	mass of the cart
m_p	0.1 kg	mass of the pendulum
l	0.5 m	distance from center of mass of pole to the pivot
g	9.8 m/s	acceleration due to gravity

$$\begin{aligned}
\theta_{t+1} &= \theta_t + \tau\dot{\theta}_t,\\
x_{t+1} &= x_t + \tau\dot{x}_t,\\
\dot{\theta}_{t+1} &= \dot{\theta}_t + \tau\ddot{\theta}_t,\\
\dot{x}_{t+1} &= \dot{x}_t + \tau\ddot{x}_t.
\end{aligned}$$

We consider our pendulum to have fallen over if $|\theta| > 12^\circ$ and set the bounds of our track to be at $\pm$ 2.4 m. We reset our system to a balanced state $(\theta, x, \dot{\theta}, \dot{x}) = (0, 0, 0, 0)$ upon failure. Our reinforcement signal $r_t = -1$ after failure, and $r_t = 0$ otherwise.

2 The Credit Assignment Problem

A central challenge of the inverted pendulum problem is establishing the contribution of individual decisions and determining which decisions were important in achieving success or failure. This is known as the credit assignment problem. A classic example of this problem is the game of checkers. In checkers, there are only three scenarios in which a non-zero signal is received: we receive a positive signal if we jump over our opponent's piece, a negative signal if our opponent jumps over one of our pieces, and a positive signal if we win the game. Clearly this signal is insufficient for learning how to play checkers. For example, a common move in checkers is to sacrifice one piece in order to take two or more of the opponent's pieces. Such a rule could never be discovered if we only use the basic reinforcement signal because we would consistently assign a negative value to the sacrificial move. Samuel (1967) was nevertheless able to develop a machine learning algorithm for playing the game by looking backward over a tree of all possible moves in order to evaluate the scores of different positions on the board.

A similar problem exists in our inverted pendulum task. Although our reinforcement signal will be sufficient to learn that extreme values of θ and x are undesirable, it does not allow us to determine which decisions led us to those extreme states in the first place.

A number of different approaches have been used to try to deal with this problem. One approach, taken in Connell and Utgoff (1987), is to make our signal more informative by using the distance from the stable $(0, 0, 0, 0)$ state as a metric for evaluating the value of different states. An alternative approach, used in Rosen et al. (1988), treats the inverted pendulum problem as a failure avoidance task and tries to identify actions which lead to cycles in the state space.

Some of the more successful solutions to the credit assignment problem have come from neural network approaches. One of the most significant innovations was proposed by Michie and Chambers (1968), who use the the expected lifetime from a certain state to drive decision-making. In particular, they discretize the state space into boxes, each of which keep track of what decision (left or right) is made from that state. Once a failure signal is received, the value at each box is updated depending on the decision that was made. The probability of making that decision in the future is then implictly updated as well. Barto et al. (1983) extend this approach and train a neural network using the discretized state space using an adaptive heuristic critic. Their approach was quite successful, and they were able to avoid failure for over thirty minutes after less than 100 failures.

Our goal was to eliminate as much structure and system information from the algorithm as possible. Nevertheless, simply using the basic reinforcement signal would not allow us to achieve success in this task. For this reason, we choose to use a variant of the reinforcement signal proposed by Michie and Chambers (1968). We only update our estimate of the value of a given state after failure. We define some constant penalty value c, and then set

$$r(S_{t^v}) = -\alpha^(t^f - t^v) \cdot c,$$

where α is a parameter $(0 < \alpha < 1)$ that indicates the rate of decay, S_{t^v} is the state we visit at time t^v, and t^f is our time of failure. This reinforcement signal mimics a decaying signal trace through the history of visited states that led us to failure. States that are visited close to failure will receive signals close to $-c$, whereas states that were visited a long time ago will receive signals ≈ 0. Note that we are not imparting any additional information to our system, except for the parameter α (which has no real physical correspondence). It does, however, mean that updating only occurs at failures. This is a weakness of the system and will contradict some of the models we use for deriving the knowledge gradient policy. Nevertheless, it remains the case that we will obtain a measurement of every state we visit: the only obstacle is that we are now dealing with a lagged information process.

3 Approximate Dynamic Programming

In this paper we use a dynamic programming approach to solve the inverted pendulum problem. Dynamic programming is a powerful technique for making decisions in a general setting. In particular, it seeks to explicitly balance the contributions of our decisions in the short term with the value of being in a particular state $V(S_t)$ at time t, where we define

$$V(S_t) = \sup_{\pi} \mathbb{E} \sum_{n=0}^{\infty} C(S_n, X^{\pi,n}, W_{n+1}).$$

Bellman's equation gives us a formula for recursively computing the value of each state:

$$V_n(S) = \max_{x \in \mathcal{X}} \left[C_n(S_n, x_n) + \gamma \mathbb{E} \left[V_{n+1}(S_{n+1}) \right] \right],$$

where $S_{n+1} = S^M(S_n, x_n, W_{n+1})$ is the state we transition to given that we take decison x_n starting from state S_n. Note that we need to take an expectation because our transition could depend on exogenous information arriving at $n+1$ (and hence unknown at time n). Traditionally the expectation is calculated by working backward through a tree with all possible future states. Unfortunately this computation can be intractable for many large scale problems. Approximate dynamic programming (ADP) seeks to remedy this by replacing our backward calculation of the expectation with a forward looking estimate of the value of future states:

$$\hat{\nu}_n(S) = \max_{x \in \mathcal{X}_n} C(S_n, x_n) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P} \left\{ s' | S_n, x_n \right\} V_n(s'),$$

where $V_n(s)$ is our time n estimate of the value of being in state s. By using an estimate we no longer have to consider all possible future states in our calculation, but only the set of states that we could reach at time $n+1$.

In a basic updating scheme, we use our estimate $\hat{\nu}_n(S)$ to update our estimate of the value of state s:

$$V_{n+1}(S_n) = (1-\alpha_{n-1})V_n(S_n) + \alpha_n \hat{\nu}_n + 1.$$

Such an approach is problematic, however, because we only update the value of the state that we actually visit. If our states were totally independent this would be a perfectly reasonable updating scheme. Generally speaking, however, our states are related to each other. For example, in our inverted pendulum problem states with *similar* values of θ will have similar values. Our approximation of the value function should take into account that similar states are related. The ability to update our estimate of the value of states other than the one we are measuring becomes particularly important as the size of our state space grows, because the frequency with which we observe our individual states becomes smaller.

4 Aggregation

4.1 Introduction

We use an aggregation-based updating model to express the relations between different states. Aggregation can be defined as a method for combining different estimates to form an approximation of the value of a state. It is a powerful tool in the context of dynamic programming because it allows us to work around the curse of dimensionality by approximating our value function in an aggregated space with a smaller number of dimensions. Aggregation-based updating schemes can also be efficiently implemented and allow us to tackle large state spaces.

We limit ourselves to *hierarchical* aggregation structures in this paper. A set of hierarchical aggregation functions satisfies

$$
\begin{aligned}
|\mathcal{S}^{(g)}| &\leq |\mathcal{S}^{(g-1)}| && (1)\\
\mathcal{A}^g(S) &= \mathcal{A}^g(S')\ if\ \mathcal{A}^{g-1}(S) = \mathcal{A}^(g-1)(S') && (2)\\
\mathcal{A}^{g-1}(S) &\neq \mathcal{A}^{g-1}(S')\ if\ \mathcal{A}^g(S) \neq \mathcal{A}^g(S'). && (3)
\end{aligned}
$$

These rules specify a hierarchy amongst our different levels of aggregation. If two states share an aggregated alternative at level $g-1$, they *must* also share an aggregated alternative at level g. Letting $\mathcal{A}^g(S) = i$ indicate that S belongs to aggregated alternative i at level g, we will have a set of indices $1, \ldots, |\mathcal{S}^{(g)}|$ for each level of aggregation. Typically at the highest level our aggregation function sets $\mathcal{A}^G(S') = 0\ \forall\ S'$, and at our lowest level $\mathcal{A}^0(\mathcal{S}) \rightarrow \mathcal{S}$.

4.2 Bayesian Model with Hierarchical Aggregation

We now define a series of properties of the aggregated alternatives and look to relate these to our estimates at a disaggregate level. In our model we consider each aggregated alternative $A^g_{(x)}$ as its own alternative with some true value θ^g_x. We then hold some belief μ^g_x with some precision β^g_x about each aggregated alternative. We update these parameter values every time that we make a measurement of that alternative.

We use a Bayesian model, which implies that we have a prior belief which we update given new evidence. Placing a prior on our value function approximation is appealing because of its generality, as we can model all of the uncertainty in our problem as a part of our value function approximation. When working with aggregation, our prior consists of three parameters: $\{\mu^0_x\}_{x\in\mathcal{X}}$, $\{\beta^x_0\}_{x\in\mathcal{X}}$, and our aggregation structure. If we let $\beta^0_{x'} = 0 \sim \forall x' \in \mathcal{X}$, then we call our prior uninformative. We have some existing estimate of the value but have zero confidence in our estimate. Even in such cases, however, we must specify an aggregation structure. We also assume that we can collect independent, unbiased, normally distributed observations $\hat{y}_x \sim N(\mu_x, \lambda_x)$ of the unknown means μ_x.

We model each of our aggregated alternatives as independent and normally distributed. Although normality is a reasonable assumption in most cases, it should be clear that our aggregated alternatives are not at all independent. Indeed, the fact that we are using an aggregation structure indicates that they are structurally related. We proceed with this modeling assumption, however, because it makes the updating equations more tractable, and we will use bias terms to alleviate the dangers of this assumption.

At the disaggregate level, we also model each of our alternatives as independent and normally distributed:

$$\theta_x \sim \mathcal{N}(\mu_x, (\sigma_x^0)^2).$$

We model our aggregated alternatives as

$$\theta_x^g \sim \mathcal{N}(\mu_x, \nu_x^g), \tag{4}$$

where ν_x^g is the variance of $\theta_x - \theta_x^g$. We consider $(\theta_x^g - \theta_x)$ to be independent across alternatives and across different values of g. We currently consider ν_x^g a fixed parameter of the model, but will eventually approximate it using an empirical Bayesian approach.

We are now prepared to model our observations and their effect on our prior distribution. Generally speaking, whenever we measure alternative x the observation $\hat{y}_x^n$ will have a distribution $\mathcal{N}(\theta_x, \lambda_x)$, where λ_x is the measurement variance. For our model we go one step further and assume that at the aggregate level we have $\hat{y}_x^{g,n} \sim \mathcal{N}(\theta_x, (\sigma_x^{g,n,\epsilon})^2$, where $(\sigma_x^{g,n,\epsilon})^2$ is in some sense the measurement variance for the aggregated alternative. In practice we set $\hat{y}_x^n = \hat{y}_x^{g,n}$, where the intuition is that $\hat{y}_x^n$ is a sample realization from $\mathcal{N}(\theta_x, (\sigma_x^{g,n,\epsilon})^2)$. This means that $(\sigma_x^{g,n,\epsilon})^2$ should be a decreasing function of g, as our range of possible sample realizations grows larger and larger as we move to lower levels of aggregation.

In the proposed Bayesian model $\hat{y}_x^{g,n}$ induces a posterior on our belief μ_x^g. If our observation $\hat{y}_{x'}^{n+1}$ does not share an aggregated alternative with x at level g ($A^g(x') \neq$

$A^g(x)$), then we simply have:

$$\begin{aligned}\mu_x^{g,n+1} &= \mu_x^{g,n},\\ \beta_x^{g,n+1} &= \beta_x^{g,n}.\end{aligned}$$

If they do share an aggregated alternative at level g, our updating equations are given by:

$$\mu_x^{g,n+1} = [\ \beta_x^{g,n}\mu_x^{g,n} + \beta_x^{g,n,\epsilon}\hat{y}_x^{n+1}\ ]\ /\ \beta_x^{g,n+1}, \tag{5}$$

$$\beta_x^{g,n+1} = \beta_x^{g,n} + \beta_x^{g,n,\epsilon}, \tag{6}$$

where $\beta_x^{g,n} = (1/\sigma_x^{g,n})^2$ and $\beta_x^{g,n,\epsilon} = (1/\sigma_x^{g,n,\epsilon})^2$ are the aggregated precision and measurement precision, respectively. We can relate our belief about the aggregated alternatives to the disaggregate belief. Using induction, we have

$$\mu_x^n = \frac{1}{\beta_x^n}\left[\beta_x^0\mu_x^0 + \sum_{g\in G}((\sigma_x^{g,n})^2 + \nu_x^g)^{-1}\mu_x^{g,n}\right], \tag{7}$$

$$\beta_x^n = \beta_x^0 + \sum_{g\in G}((\sigma_x^{g,n})^2 + \nu_x^g)^{-1}. \tag{8}$$

Our estimate of the value of the alternative is given by a weighted average of the value at the different levels of aggregation, where the weights are proportional to our knowledge of the value at that level of aggregation. That is,

$$\mu_x^n = \frac{\beta_x^0}{\beta_x^n}\mu_x^0 + \sum_{g\in G} w_x^{g,n}\mu_x^{g,n},$$

where our weights $w_x^{g,n}$ are inversely proportional to the sum of our bias and variance:

$$w_x^{g,n} = \frac{((\sigma_x^{g,n})^2 + \nu_x^g)^{-1}}{\beta_x^0 + \sum_{g'\in G}\left((\sigma_x^{g',n})^2 + \nu_x^{g'}\right)^{-1}}.$$

If we are highly uncertain or have a high level of bias at a certain level of aggregation we will largely ignore that level. If there is a low bias and high precision we will instead emphasize our estimate at this level of aggregation. It is important to note the impact of the bias term, ν_x^g. If it were not present, we would continually place emphasis on the higher levels of aggregation, despite the fact that lower levels of aggregation are likely to contain more specific information about a particular alternative.

We use approximations to compute the error term $\nu_x^{g,n}$ and the aggregate measurement error $(\sigma_x^{g,n,\epsilon})^2$. Mes et al. (2009) propose that we approximate the bias term $\nu_x^g = Var\left[\theta_x - \theta_x^g\right]$ by $(\delta_x^{g,n})^2$, where $\delta_x^{g,n} = \mu_x^{g,n} - \mu_x^{0,n}$ is an estimate of the mean of $(\theta_x - \theta_x^g)$. We note that the bias $\delta_x^{g,n}$ can be undefined if we have never measured alternative x. We resolve this by approximating the bias by $\mu_x^{g,n} - \mu_x^{g^*,n}$ for all $g > g^*$ and setting it to 0 for all $g \leq g^*$, where g^* denotes the lowest level of aggregation at which we have seen an observation. Although this is a logical assumption given the state of knowledge (we cannot approximate a bias towards a value we know nothing about), it should also be evident that our value for $\nu_x^{g,n}$ is heavily approximated.

The aggregate measurement error $(\sigma_x^{g,n,\epsilon})^2$ can be calculated using analysis of variance. We will sketch a derivation of the estimate here, but more details can be found in Mes et al. (2009) and Snijders and Bosker (1999). We first note that at the zeroth level of aggregation, our aggregate measurement error is simply the disaggregate measurement error, $\sigma_x^{0,n,\epsilon} = \lambda_x$. The aggregate measurement error at higher levels can be calculated using the fact that the group variance over a number of subgroups equals the variance within each subgroup plus the variance between the subgroups. For us, the variance within each subgroup is a weighted average of the variances $\lambda_{x'}$ of the disaggregate alternatives x' sharing an aggregated alternative at the gth level of aggregation. The variance between subgroups is given by the sum of squared deviations of the disaggregate estimates and the aggregate estimates of each alternative. Using the measurements of each alternative m_x^0 as our weights, this leads us to the following formula:

$$(\sigma_x^{g,n,\epsilon})^2 = \frac{1}{m_x^{0,n}} \sum_{x' \in \mathcal{X}^g(x)} m_{x'}^{g,n} \left[\lambda_{x'} + (\mu_{x'}^{0,n} - \mu_{x'}^{g,n})^2\right]. \tag{9}$$

Mes notes that this approximation depends strongly on the measurements, and hence on the measurement policy. He proposes instead that we place equal weight on each disaggregate alternative. Then, equation 9 becomes:

$$(\sigma_x^{g,n,\epsilon})^2 = \frac{1}{|\mathcal{X}^g(x)|} \sum_{x' \in \mathcal{X}^g(x)} \left[\lambda_{x'} + (\mu_{x'}^{0,n} - \mu_{x'}^{g,n})^2 \right]. \tag{10}$$

One will note that there exists a similar issue as before, namely that $\mu_{x'}^{0,n}$ is not defined if x' has not been measured yet. The resolution in this case is the same, and we set the bias term $(\mu_{x'}^{0,n} - \mu_{x'}^{g,n})^2$ to be equal to 0 whenever $g \leq g^*$, where g^* is again the lowest level of aggregation at which the alternative has been measured.

5 The Knowledge Gradient Policy

Having motivated the need for learning in dynamic programming, we now apply our knowledge gradient philosophy to ADP. The idea of incorporating learning concepts into ADP was first introduced by Dearden et al. (1998) and Duff and Barto (1996). Ryzhov et al. (2010) applied a knowledge gradient algorithm to the two-agent newsvendor problem modeled using a Markov decision process. Ryzhov and Powell (2010) then derived a knowledge gradient policy for an ADP setting using a multivariate normal prior. We extend this idea to hierarchical aggregation here.

Recall that a pure exploitation policy in an approximate dynamic programming context picks the decision x that satisfies

$$x_n = \arg\max_x C(S_n, x) + \gamma V_n(S^{M,x}(S_n, x)).$$

We can write $Q^n(S^n, x) = C(S_n, x) + \gamma V_n(S^{M,x}(S_n, x))$, in which case our policy maximizes $Q^n(S^n, x)$, which we call our "Q-factor". A Q-factor is simply a particular type of post-decision state, $S^x = (S, x)$. In a general dynamic programming context, we can distinguish between a pre-decision state, a post-decision state, and the next pre-decision state. The transition from the pre-decision state to the post-decision state (denoted $S^{M,x}$) implements our decision and takes into account all resulting non-stochastic transitions.

The transition to the next pre-decision state then incorporates the random realizations of the various stochastic processes driving the system. Although there are variants of the inverted pendulum problem or other stochastic elements, the post-decision state S_n^x will be equivalent to our next pre-decision state for this problem.

Q-learning is an approach to dynamic programming in which we do not keep track of the value of a state, but rather keep track of our Q-factors. This is a powerful technique because it can be applied without a model of the problem environment. We use our post-decision state variable as an alternative to a Q-factor. Our technique of assigning our maximum Q-value to our post-decision state variables can hence be viewed as a modified form of Q-learning.

Our goal is to introduce an explorative element to our approximate dynamic programming policy. A pure exploitation policy assumes that our value function approximation is accurate, and makes decisions purely on this basis. In practice there are many problems for which we have little or no prior knowledge of the value function, and for which following a pure exploitation policy can lead to poor results. The idea behind the knowledge gradient policy is that we "choos[e] the measurement that would be optimal if it were our last chance to learn." (Ryzhov and Powell, 2009) Suppose we are at state S_n at time n, and our next decision will be our last opportunity to update our beliefs. In other words, $V_{n'} = V_{n+1}$ for all $n' \geq n+1$. Under these conditions, our optimal action at time n is given by

$$x_n^* = \arg\max_x C(S_n, x) + \gamma \mathbb{E}_n^x V_{n+1}(S^{M,x}(S_n, x)), \tag{11}$$

where our expectation $\mathbb{E}_n^x$ is given with respect to our filtration $\mathcal{F}_n$ up to time n and our decision x_n. We note the similarity to Bellman's equation, except that we replace our value function $V(S_n)$ by our *updated* value function approximation $V_{n+1}(S_n)$. The knowledge gradient algorithm seeks to maximize the improvement made by the resulting change from V_n to $V_n + 1$. Recall that our value function approximation is updated after we transiton from our post-decision state S_n^x to our next pre-decision state, S_{n+1}. In order to evaluate equation 11, we need to expand the Q-factor of action x as:

$$\begin{aligned} Q^*_n(S_n, x) &= C(S_n, x) + \gamma \mathbb{E}^x_n V_{n+1}\left(S^{M,x}(S_n, x)\right), \\ &= C(S_n, x) + \gamma \sum_{S_{n+1}} \mathbb{P}(S_{n+1}|S^x_n)\mathbb{E}^x_n \max_{x'} Q_{n+1}(S_{n+1}, x'). \end{aligned}$$

We see that our transition matrix enters the equation again. We will see momentarily how we can use simulation to deal with computationally intractable transition matrices or continuous state spaces, but for now we proceed under the assumption that we can compute this value (or an estimate thereof). In the following sections we will derive our knowledge gradient value using this expanded Q-factor.

5.1 Bayesian Model for ADP

In order to compute a knowledge gradient value we first extend our Bayesian aggregation model to an ADP setting. Consider an infinite-horizon dynamic programming problem with state space $\mathcal{S}$. Our set of possible decisions is given by $\mathcal{X}$, and we require a discount factor $\gamma \in (0, 1)$. Assuming a deterministic contribution function $C(S, x)$, we can write our objective function as

$$\sup_\pi \sum_{n=0}^{\infty} \gamma_n C(S_n, X^\pi_n(S_n)),$$

where we take the maximum over all policy rules π, and use X^π_n to denote the decision rule associated with policy π. We let $R(S^x)$ be the total infinite-horizon discounted reward that we would receive if we were to start in post-decision state S^x. Because our transition from our post-decision states to our next pre-decision states are random, $R(S^x)$ is a random variable. The true value of our state is the expectation of this random variable: $V(S^x) = \mathbb{E}[R(S^x)]$. As a part of our model we assume that $R(S^x)$ has a normal distribution with unknown mean $V(S^x)$ and known variance $\lambda(S^x)$. In practice we can justify normality using a central limit argument (see Dearden et al. (1998) for details), but it is usually not the case that $\lambda(S^x)$ is known. We nevertheless proceed with this assumption, and consider $\lambda(S^x)$ a tunable parameter in cases for which it is unknown. We also assume in our model that it is possible to obtain unbiased observation

$\hat{R}(S_n^x) \sim \mathcal{N}((V(S_n^x), \lambda(S_n^x))$. We carry this assumption over from Ryzhov and Powell (2010) and Dearden et al. (1998). It is a standard assumption when dealing with optimal learning problems but does not hold in approximate dynamic programming because we cannot obtain unbiased observations of the true value of a state. The best we can do is approximate $\hat{R}$ using our biased approximation of the value of our post-decision state $\hat{\nu}_{n+1}$:

$$\hat{\nu}_{n+1} = \max_x C(S_{n+1}, x) + \gamma V_n(S^{M,x}(S_{n+1}, x)).$$

The Bayesian model with hierachical aggregation places a univariate Gaussian prior over the value of our disaggregate alternatives. We use $V_0(S)$ and $\beta_0(S)$ to denote our prior estimate of the value of our state S and the precision of our prior estimate, respectively. We use $\mu_n^{x,g}$ to denote our time n estimate of the value function for state x at level g. $\beta_n^{x,g}$ represents the precision of our approximation $\mu_n^{x,g}$. Given an observation of the value of $S_n^{x'}$, we can use the updating equations presented in section 4 to write:

$$V_{n+1}(s) = \sum_{g \in \mathcal{G}} w_{n+1}^{x,g} \mu_{n+1}^{x,g}. \tag{12}$$

Our updated means at each level of aggregation $\mu_{n+1}^{x,g}$ are given by

$$\mu_{n+1}^{x,g} = \begin{cases} \frac{\beta_n^{x,g} \mu_n^{g,x} + \beta_n^{x,g,\epsilon} \hat{R}(S_n^{x'})}{\beta_{n+1}^{x,g}} & \text{if } x \in \mathcal{X}^g(S_n^{x'}) \\ \mu_n^{x,g} & \text{otherwise} \end{cases}$$

and our updated precisions are given by

$$\beta_{n+1}^{x,g} = \begin{cases} \beta_n^{x,g} + \beta_n^{x,g,\epsilon} & \text{if } x \in \mathcal{X}^g(S_n^{x'}) \\ \beta_n^{x,g} & \text{otherwise} \end{cases}$$

Finally, we can write our updated weights as

$$w_{n+1}^{x,g} = \frac{\left((\beta_{n+1}^{x,g})^{-1} + \nu^{x,g}\right)^{-1}}{\beta^{x,0} + \sum_{g' \in G} \left((\beta_{n+1}^{x,g})^{-1} + \nu^{x,g'}\right)^{-1}}.$$

5.2 Calculating the Knowledge Gradient

In order to compute our knowledge gradient we are required to estimate our future value function approximation given a particular decision x', $V_{n+1}(S^{M,x}(S^{n+1}, x'))$, which is required for determining our updated Q-factor $Q_{n+1}(S_{n+1}, x')$. We use use $\mu_n^{x,g}$ to denote our time n estimate of the value function for state x at level g. We can write the conditional distribution of V_{n+1} given our filtration $\mathcal{F}_n$ and our decision x at time n as:

$$V_{n+1}(S^{x'}) = \tilde{V}_{n+1}(S^{x'}) + \tilde{\sigma}(S^{x'})Z, \tag{13}$$

where

$$\begin{aligned}
\tilde{V}_{n+1}(S^{x'}) &= \sum_{g\in\mathcal{G}} w_{n+1}^{x',g}\mu_n^{x',g} + \sum_{g\in\mathcal{G}(S^x,S_n^{x'})} w_{n+1}^{x',g}\frac{\beta_n^{x,g,\epsilon}}{\beta_n^{x,g}+\beta_n^{x,g,\epsilon}}\left(V_n(S^x)-\mu_n^{x,g}\right), && (14)\\
\tilde{\sigma}(S^{x'}) &= \sum_{g\in\mathcal{G}(S_n^x,S^{x'})} w_{n+1}^{x',g}\left(\frac{\beta_n^{x,g,\epsilon}\sqrt{(1/\beta_n^{S^x})+\lambda(S^x)}}{\beta_n^{x,g}+\beta_n^{x,g,\epsilon}}\right). && (15)
\end{aligned}$$

Note that we have picked $\tilde{\sigma}(s, S_n^x)$ such that our normal random variable Z is no longer indexed by s and is independent of our post-decision state S_n^x. We now rewrite our Q-factor in terms of an expectation of the conditional distribution of $V_{n+1}(s)$:

$$\begin{aligned}
Q_n^*(S_n, x) &= \mathbb{E}\max_{x'} Q_{n+1}\left(S_{n+1}(x')\right) && (16)\\
&= \mathbb{E}\max_{x'}(a_n^{x'} + b_n^{x'}Z), && (17)
\end{aligned}$$

where

$$\begin{aligned}
a_n^{x'} &= C(S^{n+1}, x') + \gamma\tilde{V}_{n+1}(S^{x'}), && (18)\\
b_n^{x'} &= \gamma\tilde{\sigma}(S^{x'}). && (19)
\end{aligned}$$

An algorithm for computing this expectation of a maximum is presented in Frazier et al. (2009). We set

$$\mathbb{E}\max_{x'}(a_n^{x'} + b_n^{x'}Z) = (\max_{x'} a_{x'}^n) + \sum_{y\in\mathcal{A}}(b_{y+1}^n - b_y^n)f(-|c_y|), \tag{20}$$

where $\mathcal{A}$ is the set of all y for which we can find numbers $c_{y-1} \leq c_y$ for which $y = \arg\max_{x'} a_{x'}^n + b_{x'}^n z$ for $z \in (c_{y-1}, c_y)$, with ties broken by the largest-index rule. The elements of A are renumbered in order of increasing $b_{x'}^n$, and our function $f(z) = z\Phi(z) + \phi(z)$. Using this computation, we can define our knowledge gradient to be the *difference*

$$\begin{aligned}\nu_n^{KG}(S_n^x, S^{n+1}) &= \mathbb{E}_n^x \max_{x'}(a_n^{x'} + b_n^{x'}Z) - \max_{x'} a_n^{x'} \\ &= \sum_{y\in\mathcal{A}}(b_n^{y+1} - b_n^y)f(-|c_y|).\end{aligned}$$

In other words, our knowledge gradient is the expected improvement in our estimate of $\max Q_{n+1}(S_{n+1}, x')$ after taking action x from starting state S^n.

5.3 The Knowledge Gradient Policy for ADP

We now incorporate our knowledge gradient value $\nu^{KG,x}$ into our policy. Recall that our optimal action under the knowledge gradient policy is given by $x_n^* = \arg\max_x Q_n(S_n, x)$, where our Q-factor was given by:

$$Q_n(S_n, x) = C(S_n, x) + \gamma \sum_{S_{n+1}} \mathbb{P}(S^{n+1}|S_n^x)\mathbb{E}_n^x \max_{x'} Q_{n+1}(S_{n+1}, x').$$

We can rewrite the sum on the right hand side as

$$\begin{aligned}&\sum_{S_{n+1}} \mathbb{P}(S_{n+1}|S_n^x)\ \mathbb{E}\max_{x'} Q_{n+1}(S_{n+1}, x') \\ &= \sum_{S_{n+1}} \mathbb{P}(S_{n+1}|S_n^x)\ \max_{x'} Q_n(S_{n+1}, x') \\ &= \sum_{S_{n+1}} \mathbb{P}(S_{n+1}|S_n^x)\ \nu_n^{KG}(S_n^x, S_{n+1}).\end{aligned}$$

We now use the fact that our value function approximation of the post-decision state is in fact an estimate of our expected Q-factor to write

$$\sum_{S_{n+1}} P(S_{n+1}|S_n^x) \max_{x'} Q_n(S_{n+1}, x') = V_n(S_n^x). \tag{21}$$

Finally, we can rewrite our optimal action given by equation 11 as

$$x_n^* = \arg\max_x C(S_n, x) + \gamma \left(V_n(S_n^x) + \sum_{S^{n+1}} P(S_{n+1}|S_n^x) \nu_n^{KG}(S_n^x, S_{n+1}) \right) \tag{22}$$

We note that if we eliminate the last term of the equation involving our sum of weighted KG factors, we have the pure exploitation rule given in equation 5 at the beginning of this chapter. We modify the rule, however, using a KG "bonus" value which reflects the uncertainty in our value function approximation. We know have an explicit, quantifiable balance between the expected value of information and the expected reward and value of our future state which helps us solve the problem of balancing exploitation and exploration. When we are uncertain about our value function, decisions with high KG values may be preferred to our pure exploitation decision. As we become more certain, our KG factors become smaller and we gradually transition towards an exploitation strategy.

6 Results

We applied our policy to the inverted pendulum problem. Traditionally, the efficacy of a policy is determined by examining the sum of the discounted contributions:

$$C^\pi = \sum_{i=0}^{\infty} C(S_i, X^\pi(S_i)). \tag{23}$$

For our problem the contribution is determined by the reinforcement signal. Since this number is always either negative or 0, this is a cost-minimization problem. We used a mixed exploration and an epsilon-greedy policy in order to benchmark our results. We tuned the mixed exploration policy paramter ρ (our probability of exploration) to a value of .01. We also tuned an epsilon-greedy policy for which the probability of exploration took the form

$$\epsilon_n = \frac{a}{a+n},$$

where we set $a = 100$ for our experiments.

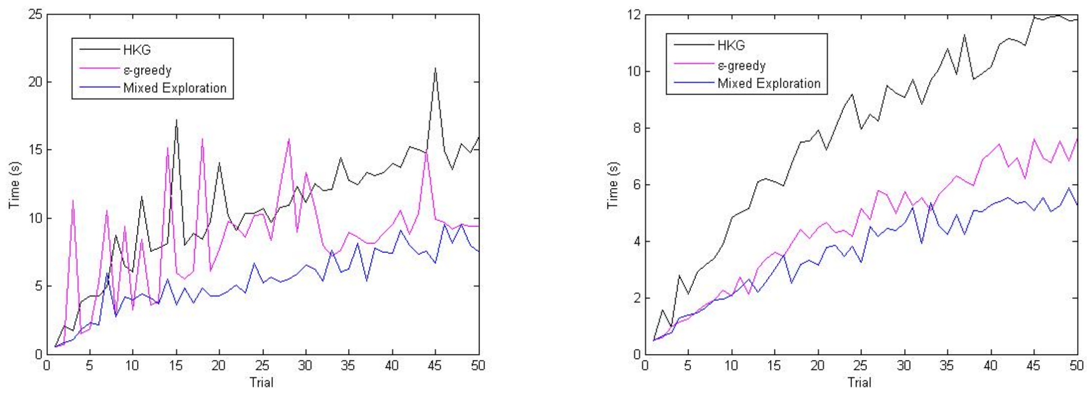
We chose to amplify our reinforcement signal by a factor of 100 in order to avoid working with excessively small numbers. This is merely a rescaling of the problem and had no effect on the relative results. We also used a discount factor of $\gamma = .9999$; our discount factor had to be close to 1 as we were comparing contributions over tens of thousands of iterations. The dynamics of the system were identical to those presented in Chapter 1, and we ran each policy for 100 trials, each of which consisted of $50,000$ iterations, or 1000 seconds. Finally, we used a basic symmetric aggregation structure, in which we discretized each dimension into 2^{G-g} elements at level g for all $g \in 0 \dots G$, where $G = 6$.

We can see in Table 1 that our knowledge gradient policy significantly outperforms both an epsilon-greedy and a mixed exploration policy when considering our dynamic programming objective function. Although this is a first indicator of success, we also examine alternative objective functions. The objective given in equation 23 is a reasonable approach for dynamic programming, but its may not be the most appropriate mechanism for determining success in the inverted pendulum task as a whole.

Table 1: Comparative Results using Traditional DP Objective

	Mean	Avg. SE
HKG	**-648.03**	20.95
Epsilon-Greedy	-940.56	50.76
Mixed Exploration	-993.43	29.74

A number of other scoring mechanisms present themselves for the inverted pendulum task. In particular, we were interested to see how each policy maximizes the amount learned in a given iteration. We therefore focused on examining how long a given policy was able to maintain a balanced system after a given set of trials (failures). We show the mean balancing time at an individual trial for our three policies below in Figure 2(a). We can see that, generally speaking our HKG policy outperforms both the epsilon-greedy and the mixed exploration policies. As we would expect, our epsilon-greedy performs better than our mixed exploration policy, which is too static and does not take into account the increasing accuracy of the value function approximation over time.



(a) Mean Balancing Time vs. Failures (b) Median Balancing Times vs. failures

Figure 2: Comparative Balancing Times vs. Failures

Note that despite sampling 100 trials for each policy there is still a large amount of volatility in our results. This is in large part due to the inherent volatility of our problem. In one sample path for our epsilon-greedy policy, for example, the system failed twice in the first three seconds but never failed again. Similar sample paths were present for each of these policies. For this reason, we also compared the median failure times after each iteration in Figure 2(b). In this case we see a similar pattern but also observe a more clear distinction between the policies.

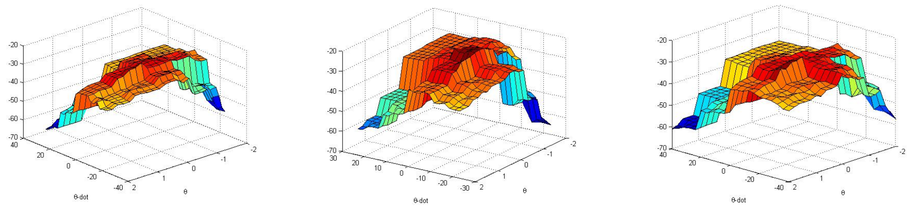
An alternative way of defining success in this task is to determine how many failures occurred during a given trial. Such a policy is similar to our dynamic programming objective, but we now choose to ignore discounting. We report the statistics regarding total failures below in Table 2. We can see that the same ordering of policies is maintained. It is also interesting to note that the distinction between epsilon-greedy and a mixed exploration policy is more pronounced here. It seems that reducing the probability of exploration over time helps reduce failures in later iterations.

In addition to validating the value of our knowledge gradient algorithm, it is important to note that we succesfully used aggregation to approximate our value function. This is

Table 2: Comparative Results using Total Failures

	Mean	Avg. SE
HKG	**97.36**	3.21
Epsilon-Greedy	127.11	8.80
Mixed Exploration	152.61	7.74

not at all given, as we are bootstrapping a truth using a fairly general updating scheme. We show comparative plots of the value function in the θ vs. $\dot{\theta}$ dimensions under our three policies below. Note that we constrain ourselves to the most interesting region of the state space, and that our knowledge gradient policy shows the finest approximation of this region, and mixed exploration shows the coarsest approximation.



(a) HKG (b) Epsilon-Greedy (c) Mixed Exploration

Figure 3: Value Function Approximations

7 Conclusion

We have examined the validity of an approximate dynamic programming policy towards solving the inverted pendulum task. We have seen that all of our policies were able to gradually improve performance over time by learning the value function using an aggregation-based approach. This is signficant because our approach starts with no a priori information about the system. The only modification to the reinforcement signal was the use of a time-decaying signal trace to update states visited in the past, which was necessary to heuristically solve the credit assignment problem. We note, however, that this is a generic solution to the credit assignment problem and is in no way particular to our problem.

In addition to the success of an ADP approach, we also saw that our knowledge gradient policy outperformed an epsilon-greedy and a mixed exploration policy using a number of different objective functions. This is somewhat remarkable given that our updating procedure explicitly violates a number of modeling assumptions present in our derivation of the policy, so we examine possible reasons for the success of our algorithm despite these modeling issues.

The primary issue with our model is that we are not able to update any of our states until we fail. This makes the definition of the “best possible measurement given that this will be our last chance to learn” tricky. If our knowledge gradient policy were “correctly” adjusted to match this inuition, for example, we would always choose to fail (and hence update a large number of states) given a choice to do so. In our implementation, however, we ignore the lag of information. What we are claiming is that the difference between receiving information and receiving information later is negligible. In many problems this claim would not hold, but it is not entirely unreasonable in the context of the inverted pendulum task. Our discount factor γ is close to 1 and our trial times are relatively short, particularly in the early iterations when we are most interested in the value of learning.

A secondary reason for the success of our algorithm is that it takes both the value function approximation and our confidence in the value function approximation into account. Whereas both our competing policies explore the state space randomly, the knowledge gradient policy explores intelligently. This means that in later iterations the KG policy will not be interested in exploring near-failure states regardless of the confidence differential, whereas the determination of which state to explore is entirely random for our alternative policies.

Although the results presented here are promising, more work remains to be done. In particular, the performance of our algorithm in the long run remains an open question. Although our focus here was on learning, and hence on short-term results, it remains to be seen whether an ADP approach can come close to matching the performance shown by algorithms such as those presented in Barto et al. (1983). An additional area of further research is the development of a more accurate knowledge gradient policy for lagged information processes. One possible approach is to calculate an approximate discount factor to scale the value of future information. In our inverted pendulum problem, for example, we could store a smoothed estimate of trial times in order to compute an approximate discount factor.

In conclusion, we find that approximate dynamic programming is able to achieve success in the inverted pendulum problem. Moreover, we find that a knowledge gradient outperforms both an epsilon-greedy and a mixed exploration policy using both the traditional dynamic programming objective and a series of alternative problem-dependent objectives. Although this research shows the significant promise of ADP and knowledge gradient algorithms, we also note that significant work remains to be done in the future to examine possible refinements to our algorithm and extensions to other problems.

References

Anderson, C. (1989). Learning to control an inverted pendulum using neural networks. *Control Systems Magazine*, 9(3):31–37.

Barto, A., Sutton, R., and Anderson, C. (1983). Neuronlike adaptive elements that can solve difficult learning problems. *IEEE Transcations on Systems, Man, and Cybernetics*, SMC–13(5):835–846.

Bellman, R. (1957). *Dynamic Programming.* Princeton University Press.

Chick, S., Branke, J., and Schmidt, C. (2007). New myopic sequential sampling procedures. *INFORMS Journal on Computing*, 22(1):71–80.

Connell, M. and Utgoff, P. (1987). Learning to control a dynamical physical system. *Proceedings of the American Association for Artificial Intelligence*, 2:456–640.

Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian q-learning. In *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, pages 761–768.

Duff, M. and Barto, A. (1996). Local bandit approximation for optimal learning problems. *Advances in Neural Information Processing Systems*, 9:1019–1025.

Engel, Y., Mannor, S., and Meir, R. (2003). Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 154–161.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with gaussian process priors. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 208–215.

Frazier, P., Powell, W., and Dayanik, S. (2008). A knowledge gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439.

Frazier, P., Powell, W. B., and Dayanik, S. (2009). The knowledge-gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613.

George, A., Powell, W. B., and Kulkarni, S. (2008). Value function approximation using multiple aggregation for multiattribute resource management. *Journal of Machine Learning Research*, 9:2079–2111.

Gupta, S. S. and Miescke, K. J. (1996). Bayesian look ahead one-stage sampling allocations for selection of the best population. *Journal of Statistical Planning and Inference*, 54(2):229–244.

Mes, M., Powell, W. B., and Frazier, P. (2009). Hierarchical knowledge gradient for sequential sampling. *Journal of Machine Learning Research*, pages 1–33.

Michie, D. and Chambers, R. (1968). Boxes: An experiment in adaptive control. In Michie, D., editor, *Machine Intelligence*. Oliver and Boyd, Edinburgh, UK.

Negoescu, D. (2011). Optimal learning for drug discovery in ewing's sarchoma. To appear.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curse of Dimensionality*. Wiley-Interscience.

Powell, W. B. and Frazier, P. (2008). Optimal learning. *INFORMS*, Tutorials:213–246.

Roberge, J. K. (1960). *The mechanical seal*. PhD thesis, M.I.T., Cambridge, Massachusetts.

Rosen, B., Goodwin, J., and Vidal, J. (1988). State recurrence learning. In *First Annual International Neural Neotwork Society Meeting*, Boston, MA.

Ryzhov, I. O. and Powell, W. B. (2009). The knowledge gradient algorithm for online subset selection. In *Proceedings of the 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 137–144, Nashville, TN.

Ryzhov, I. O. and Powell, W. B. (2010). Approximate dynamic programming with correlated bayesian beliefs. In *Forty-Eighth Annual Allerton Conference on Communication, Control, and Computing*.

Ryzhov, I. O. and Powell, W. B. (2011). Information collection on a graph. *Operations Research*, 59(1):188–201.

Ryzhov, I. O., Valdez-Vivas, M. R., and Powell, W. B. (2010). Optimal learning of transition probabilities in the two-agent newsvendor problem. In Johansson, B., Jain, S., Montoya-Torres, J., Hugan, J., and Yucesan, E., editors, *Proceedings of the 2010 Winter Simulation Conference*, pages 1088–1098.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. *IBM Journal of Research Development*, 3(3).

Selfridge, O., Sutton, R., and Barto, A. (1985). Training and tracking in robotics. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 670–672.

Silver, E. (1963). Markov decision processes with uncertain transition probabilities or rewards. Technical Report 1, MIT Operations Research Center.

Snijders, T. A. and Bosker, R. J. (1999). *Multilevel analysis: An introduction to basic and advanced multilevel modeling.* Sage Publications Ltd.