# Planning with Expectation Models

**Yi Wan** [*], **Muhammad Zaheer** [*], **Adam White**, **Martha White** and **Richard S. Sutton**
Reinforcement Learning and Artificial Intelligence Laboratory, University of Alberta
{wan6, mzaheer, amw8, whitem, rsutton}@ualberta.ca

## Abstract

Distribution and sample models are two popular model choices in model-based reinforcement learning (MBRL). However, learning these models can be intractable, particularly when the state and action spaces are large. Expectation models, on the other hand, are relatively easier to learn due to their compactness and have also been widely used for deterministic environments. For stochastic environments, it is not obvious how expectation models can be used for planning as they only partially characterize a distribution. In this paper, we propose a sound way of using approximate expectation models for MBRL. In particular, we 1) show that planning with an expectation model is equivalent to planning with a distribution model if the state value function is linear in state features, 2) analyze two common parametrization choices for approximating the expectation: linear and non-linear expectation models, 3) propose a sound model-based policy evaluation algorithm and present its convergence results, and 4) empirically demonstrate the effectiveness of the proposed planning algorithm. [1]

## 1 Introduction

Learning models of the world and effectively planning with them remains a long-standing challenge in artificial intelligence. Model-based reinforcement learning formalizes this problem in the context of reinforcement learning where the *model* refers to the environment's transition dynamics and reward function. The output of the model is one of the key choices in the design of a planning agent, as it determines the way the model is used for planning. Should the model produce 1) a distribution over the next state feature vector, 2) a sample of the next state feature vector, or 3) the expected next state feature vector? For stochastic environments, distribution and sample models can be used effectively, particularly if the distribution can be assumed to be of a special form [Deisenroth and Rasmussen, 2011; Chua *et al.*, 2018]. For arbitrarily stochastic environments, learning a sample or distribution model could be intractable or even impossible. For deterministic environments, expectation models appear to be the default choice as they are easier to learn and have been used [Oh *et al.*, 2015; Leibfried *et al.*, 2016]. However, for general stochastic environments, it is not obvious how expectation models can be used for planning as they only partially characterize a distribution. In this paper, we develop an approach to use expectation models for arbitrarily stochastic environments by restricting the value function to be linear in the state-feature vector.

Once the choice of expectation models with linear value function has been made, the next question is to develop an algorithm which uses the model for planning. In previous work, planning methods have been proposed which use expectation models for policy evaluation [Sutton *et al.*, 2012]. However, as we demonstrate empirically, the proposed methods require strong conditions on the model which might not hold in practice, causing the value function to diverge to infinity. Thus, a key challenge is to devise a *sound* planning algorithm which uses approximate expectation models for policy evaluation and has convergence guarantees. In this work, we propose a new objective function called *Model Based-Mean Square Projected Bellman Error* (MB-MSPBE) for policy evaluation and show how it relates to *Mean Square Projected Bellman Error* (MSPBE) [Sutton *et al.*, 2009]. We derive a planning algorithm which minimizes the proposed objective and show its convergence under conditions which are milder than the ones assumed in the previous work [Sutton *et al.*, 2012].

It is important to note that in this work, we focus on the value prediction task for model-based reinforcement learning. Predicting the value of a policy is an integral component of *Generalized Policy Iteration* (GPI) on which much of modern reinforcement learning control algorithms are built [Sutton and Barto, 2018]. Policy evaluation is also key for building predictive world knowledge where the questions about the world are formulated using value functions [Sutton *et al.*, 2011; Modayil *et al.*, 2014; White and others, 2015]. More recently, policy evaluation has also been shown to be useful for representation learning where value functions are used as auxiliary tasks [Jaderberg *et al.*, 2016]. While model-based reinforcement learning for policy evaluation is interesting in its own right, the ideas developed in this paper can also be extended to the control setting.

[*] denotes joint first authorship

[1] Supplementary materials: https://arxiv.org/abs/1904.01191

## 2 Problem Setting

We formalize an agent's interaction with its environment by a finite Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{R}$ is a set of rewards, $\gamma \in [0, 1)$ is a discount factor, and $p : \mathcal{S}\times\mathcal{A}\times\mathcal{S}\times\mathcal{R} \mapsto [0, 1]$ is the transition dynamics such that $p(s', r|s, a) \doteq \Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ for all $s, s' \in \mathcal{S}, a \in \mathcal{A}$, and $r \in \mathcal{R}$. A stationary policy $\pi$ determines the behavior of the agent. The value function $v_\pi : \mathcal{S} \mapsto \mathbb{R}$ describes the expected discounted sum of rewards obtained by following policy $\pi$ from each state.

In practice, the agent does not have access to the states directly, but only through an $m$-dimensional real-valued feature vector $\mathbf{x}_t = \mathbf{x}(S_t)$ where $\mathbf{x} : \mathcal{S} \mapsto \mathbb{R}^m$ is the feature mapping, which can be an arbitarily complex function for extracting the state-features. Thus, policies are formally mappings from state-feature vectors and actions to the probability of taking the action in response to the feature vector. Tile-coding [Sutton, 1996] and Fourier basis [Konidaris *et al.*, 2011] are examples of state-feature mapping functions which are expert designed. An alternative is to learn the mapping $\mathbf{x}$ using auxiliary tasks and approximate the value function using the learned state-features [Chung *et al.*, 2018; Jaderberg *et al.*, 2016]. In that case the value function is usually approximated using a parametrized function with an $n$-dimensional weight vector $\mathbf{w} \in \mathbb{R}^n$, where typically $n \ll |\mathcal{S}|$. We write $\hat{v}(\mathbf{x}(s), \mathbf{w}) \approx v_\pi(s)$ for the approximate value of state $s$. The approximate value function can either be a linear function of the state-features $\hat{v}(\mathbf{x}, \mathbf{w}) = \mathbf{x}^\top\mathbf{w}$ where $n = m$, or a non-linear function $\hat{v}(\mathbf{x}, \mathbf{w}) = f(\mathbf{x}, \mathbf{w})$ where $f : \mathbb{R}^m \times \mathbb{R}^n \mapsto \mathbb{R}$ is an arbitrary function. Similarly, it's common to use the state feature vector as the input of the policy and both input and output of the approximate model, which we will discuss in the next section.

The Dyna architecture [Sutton, 1991] is an MBRL algorithm which unifies learning, planning, and acting via updates to the value function. The agent interacts with the world, using observed state, action, next state, and reward tuples to estimate the model $p$, and update an estimate of the action-value function for policy $\pi$. The planning step in Dyna repeatedly samples possible next state, and rewards from the model, given input state-action pairs. These hypothetical experiences can be used to update the action-value function, just as if they had been generated by interacting with the environment. The *search control* process decides what states and actions are used to query the model during planning. The efficiency of planning can be significantly improved with non-uniform search control such as prioritized sweeping [Moore and Atkeson, 1993; Sutton *et al.*, 2012; Pan *et al.*, 2018]. In the function approximation setting, there are three factors that can affect the solution of a planning algorithm: 1) the distribution of data used to train the model, 2) the search control process's distribution for selecting the starting feature vectors and actions for simulating the next feature vectors, and 3) the policy being evaluated.

Consider an agent wanting to evaluate a policy $\pi$, i.e., approximate $v_\pi$, using a Dyna-style planning algorithm. Assume that the data used to learn the model come from the agent's interaction with the environment using policy $b$. It is common to have an ergodicity assumption on the markov chain induced by $b$:

**Assumption 2.1** *The markov chain induced by policy b is ergodic.*

Under this assumption we can define the expectation $\mathbb{E}_b[\cdot]$ in terms of the unique stationary distribution, for example, $\mathbb{E}_b[\mathbf{x}_t\mathbf{x}_t^\top] = \sum_{s\in\mathcal{S}} \eta(s)\mathbf{x}(s)\mathbf{x}(s)^\top$.

Let $\eta$ denote $b$'s stationary state distribution and let $H_{\boldsymbol{\phi}} \doteq \{s \in \mathcal{S} : \mathbf{x}(s) = \boldsymbol{\phi}\}$ as the set of all states sharing feature vector $\boldsymbol{\phi}$. Consequently, the stationary feature vector distribution corresponding to $\eta$ would be $\mu(\boldsymbol{\phi}) = \sum_{s\in H_{\boldsymbol{\phi}}} \eta(s)$. Let's suppose the search control process generates a sequence of i.i.d random vectors $\{\boldsymbol{\phi}_k\}$ where each $\boldsymbol{\phi}_k$ follows distribution $\zeta$, and chooses actions $\{A_k\}$ according to policy $\pi$ i.e. $A_k \sim \pi(\cdot|\boldsymbol{\phi}_k)$, which is the policy to be evaluated. The $\boldsymbol{\phi}_k$ is usually assumed to be bounded.

**Assumption 2.2** $\{\boldsymbol{\phi}_k\}$ *is bounded.*

Since we assumed the finite MDP setting, the number of states, actions and feature vectors are all finite and the model output is, therefore, always bounded. For the uniqueness of the solution, it is also assumed that the feature vectors generated by the search-control process are linearly independent:

**Assumption 2.3** $\mathbb{E}[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top]$ *is non-singular.*

## 3 Models

A model of the dynamics can be learned if the dynamics are not known. The model may map a state-feature vector and an action to either a distribution over the next state-feature vectors and rewards (distribution model), a sample from the distribution over the next-state feature vectors and rewards (sample model), or to the expected next state-feature vector and reward (expectation model).

A distribution model takes a state-feature vector and action as input and produces (typically) the expectation of the next reward and a distribution over the feature vectors corresponding to the next state. A distribution model consists of two functions, $\hat{r}$ and $\hat{p}$, where $\hat{r}(\mathbf{x}, a) \approx \mathbb{E}_b[R_{t+1}|\mathbf{x}_t = \mathbf{x}, A_t = a]$ and $\hat{p}(\mathbf{x}'|\mathbf{x}, a) \approx \mathbf{Pr}[\mathbf{x}_{t+1} = \mathbf{x}'|\mathbf{x}_t = \mathbf{x}, A_t = a]$. Distribution models have typically been used with special forms such a Gaussians [Chua *et al.*, 2018] or Gaussian processes [Deisenroth and Rasmussen, 2011]. In general, however, learning a distribution can be impractical as distributions are potentially large objects. For example, if the state is represented by a feature vector of dimension $m$, then the first moment of its distribution is a $m$-vector, but the second moment is a $m \times m$ matrix, and the third moment is $m \times m \times m$, and so on.

Sample models are a more practical alternative, as they only need to generate a sample of the next-state feature vector and reward, given a state-feature vector and action. Sample models can use arbitrary distributions to generate the samples—even though they do not explicitly represent those distributions—but can still produce objects of a limited size (e.g. feature vectors of dimension $m$). They are particularly well suited for sample-based planning methods such as Monte Carlo Tree Search [Coulom, 2006]. Unlike distribution models, however, sample models are stochastic which

creates an additional branching factor in planning, as multiple samples are needed to be drawn to gain a representative sense of what might happen.

Expectation models are an even simpler approach, where the model produces the *expectation* of the next reward and the next-state feature vector. An expectation model consists of $\hat{r}$ as above for a distribution model and a second function $\hat{\mathbf{x}}$ such that $\hat{\mathbf{x}}(\mathbf{x}, a) \approx \mathbb{E}_b[\mathbf{x}_{t+1}|\mathbf{x}_t = \mathbf{x}, A_t = a]$. The advantages of expectation models are that the state output is compact (like a sample model) and deterministic (like a distribution model). The potential disadvantage of an expectation model is that it is only a partial characterization of the distribution. For example, if the result of an action (or option) is that two binary state features both occur with probability 0.5, but are never present (=1) together, then an expectation model can capture the first part (each present with probability 0.5), but not the second (never both present). This may not be a substantive limitation, however, as we can always add a third binary state feature, for example, for the AND of the original two features, and then capture the full distribution with the expectation of all three state features.

## 4 Expectation Models and Linearity

Expectation models can be less complex than distribution and sample models and, therefore, can be easier to learn. This is especially critical for model-based reinforcement learning where the agent is to learn a model of the world and use it for planning. In this work, we focus on answering the question: how can expectation models be used for planning in Dyna, despite the fact that they are only a partial characterization of the transition dynamics?

There is a surprisingly simple answer to this question: if the value function is linear in the state features, then there is no loss of generality when using an expectation model instead of a distribution model for planning. Let $\hat{p}$ be the next-state part of an arbitrary distribution model and $\hat{\mathbf{x}}$ be the next-state part of the corresponding expectation model, that is, with $\hat{\mathbf{x}}(\mathbf{x}, a) = \sum_{\mathbf{x}'} \mathbf{x}' \hat{p}(\mathbf{x}'|\mathbf{x}, a)$. Let both models share the same $\hat{r}$. Then planning (policy evaluation) with the distribution model and linear function approximation performs an update of $\mathbf{w}$ for each state-feature vector $\boldsymbol{\phi}$ generated by the search control process such that $\hat{v}(\boldsymbol{\phi}, \mathbf{w})$ is moved toward:

$$\sum_a \pi(a|\boldsymbol{\phi}) \left[ \hat{r}(\boldsymbol{\phi}, a) + \gamma \sum_{\boldsymbol{\phi}'} \hat{p}(\boldsymbol{\phi}'|\boldsymbol{\phi}, a) \hat{v}(\boldsymbol{\phi}', \mathbf{w}) \right] \tag{1}$$

$$= \sum_a \pi(a|\boldsymbol{\phi}) \left[ \hat{r}(\boldsymbol{\phi}, a) + \gamma \sum_{\boldsymbol{\phi}'} \hat{p}(\boldsymbol{\phi}'|\boldsymbol{\phi}, a) \boldsymbol{\phi}'^\top \mathbf{w} \right]$$

$$= \sum_a \pi(a|\boldsymbol{\phi}) \left[ \hat{r}(\boldsymbol{\phi}, a) + \gamma \hat{\mathbf{x}}(\boldsymbol{\phi}, a)^\top \mathbf{w} \right] \tag{2}$$

The last expression, using just an expectation model, is equal to the first, thus showing that no generality has been lost compared to an arbitrary distribution model, if the value function is linear. Further, the same equations also advocate the other direction: if we are using an expectation model, then the approximate value function should be linear. This is because (2) is unlikely to equal (1) for general distributions if $\hat{v}$ is not linear in state-features.

It is important to point out that linearity does not particularly restrict the expressiveness of the value function since the mapping $\mathbf{x}$ could still be non-linear and, potentially, learned end-to-end using auxiliary tasks [Jaderberg *et al.*, 2016; Chung *et al.*, 2018].

## 5 Linear & Non-Linear Expectation Models

We now consider the parameterization of the expectation model: should the model be a linear projection from state-features to the next state-features or should it be an arbitrary non-linear function? In this section, we discuss the two common choices and their implications in detail.

We assume a mapping $\mathbf{x}$ for state-features and a value function $\hat{v}$ which is linear in state-features. The general case for an expectation model is that $\hat{\mathbf{x}}$ and $\hat{r}$ are arbitrary non-linear functions. A special case is of the linear expectation model, in which both of these functions are linear, i.e., $\hat{\mathbf{x}}(\boldsymbol{\phi}, a) = \mathbf{F}_a \boldsymbol{\phi}$ and $\hat{r}(\boldsymbol{\phi}, a) = \mathbf{b}_a^\top \boldsymbol{\phi}$ where $\mathbf{F}_a$ is the transition matrix and $\mathbf{b}_a$ is the expected reward vector.

We now define the best linear expectation model and the best non-linear expectation model trained using data generated by policy $b$. In particular, let $\{\mathbf{F}_a^*, \mathbf{b}_a^*\}$ be the best linear expectation model where

$$\mathbf{F}_a^* \doteq \arg\min_{\mathbf{G}} \mathbb{E}_b[\mathbb{I}(A_t = a) \|\mathbf{G}\mathbf{x}_t - \mathbf{x}_{t+1}\|_2^2]$$

$$\mathbf{b}_a^* \doteq \arg\min_{\mathbf{u}} \mathbb{E}_b[\mathbb{I}(A_t = a)(\mathbf{u}^\top \mathbf{x}_t - R_{t+1})^2]$$

For the uniqueness of the best linear model, we assume

**Assumption 5.1** $\mathbb{E}_b[\mathbb{I}(A_t = a)\mathbf{x}_t \mathbf{x}_t^\top]$ *is non-singular* $\forall a \in \mathcal{A}$

Under this assumption we have closed-from solution for linear model.

$$\mathbf{F}_a^* = \mathbb{E}_b[\mathbb{I}(A_t = a)\mathbf{x}_{t+1}\mathbf{x}_t^\top]\mathbb{E}_b[\mathbb{I}(A_t = a)\mathbf{x}_t\mathbf{x}_t^\top]^{-1}$$

$$\mathbf{b}_a^* = \mathbb{E}_b[\mathbb{I}(A_t = a)\mathbf{x}_t\mathbf{x}_t^\top]^{-1}\mathbb{E}_b[\mathbb{I}(A_t = a)\mathbf{x}_t R_{t+1}]$$

The best non-linear expectation model $\{\hat{\mathbf{x}}^*, \hat{r}^*\}$.

$$\hat{\mathbf{x}}^*(\boldsymbol{\phi}, a) \doteq \mathbb{E}_b[\boldsymbol{\phi}'|\boldsymbol{\phi}, a] = \frac{\sum_{s \in H_{\boldsymbol{\phi}}} \eta(s) \mathbb{E}[\mathbf{x}(S')|S = s, A = a]}{\mu(\boldsymbol{\phi})}$$

$$\hat{r}^*(\boldsymbol{\phi}, a) \doteq \mathbb{E}_b[R|\boldsymbol{\phi}, a] = \frac{\sum_{s \in H_{\boldsymbol{\phi}}} \eta(s) \mathbb{E}[R|S = s, A = a]}{\mu(\boldsymbol{\phi})}$$

Both linear and non-linear models can be learned using samples via stochastic gradient descent.

### 5.1 Why Linear Models are Not Enough?

In previous work, linear expectation models have been used to simulate a transition and execute TD(0) update [Sutton *et al.*, 2012]. Convergence to the TD-fixed point using TD(0) updates with a *non-action linear expectation model* is shown in theorem 3.1 and 3.3 of [Sutton *et al.*, 2012]. An additional benefit of this method is that the point of convergence does not rely on the distribution $\zeta$ of the search-control process.

Critically, a non-action model cannot be used for evaluating an arbitrary policy, as it is tied to a single policy – the one that generates the data for learning the model. To evaluate multiple policies, an action model is required. In this case, the point of convergence of the algorithm is dependent on $\zeta$. From corollary 5.1 of [Sutton *et al.*, 2012], the convergent point of TD(0) update with action model $\{\mathbf{F}_a, \mathbf{b}_a\}$ is:

$$(\mathbf{I} - \gamma \mathbf{F}^\top)^{-1}\mathbf{b} \tag{3}$$

where $\mathbf{F} = \mathbb{E}[\mathbf{F}_{A_k}\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top]\mathbb{E}[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top]^{-1}, \mathbf{b} = \mathbb{E}[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top]^{-1}\mathbb{E}[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top\mathbf{b}_{A_k}]$. It is obvious that the convergence point changes as the feature vector generating distribution $\zeta$ changes. We now ask even if $\zeta$ equals $\mu$, do the TD(0)-based planning updates converge to the TD fixed-point. In the next proposition we show that this is not true in general for the best linear model, however, it is true for the best non-linear model.

Let the TD-fixed point with real environment be $\mathbf{w}_{\text{real}}$, with the best-linear model $\{\mathbf{F}_a^*, \mathbf{b}_a^*\}$ be $\mathbf{w}_{\text{linear}}$ , and with the best non-linear model $\{\hat{\mathbf{x}}^*, \hat{r}^*\}$ be $\mathbf{w}_{\text{non-linear}}$ (assuming they exist). We can write their expressions as follow:

$$\mathbf{w}_{\text{real}} = \mathbb{E}[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top]^{-1}\mathbb{E}[\rho_t R_{t+1}\mathbf{x}_t] \tag{4}$$
$$\mathbf{w}_{\text{non-linear}} = \mathbb{E}[\boldsymbol{\phi}_k(\boldsymbol{\phi}_k - \gamma\hat{\mathbf{x}}^*(\boldsymbol{\phi}_k, A_k))^\top]^{-1}\mathbb{E}_b[r^*(\boldsymbol{\phi}_k, A_k)\boldsymbol{\phi}_k]$$
$$\mathbf{w}_{\text{linear}} = \text{equation (3) with } \mathbf{F}_a = \mathbf{F}_a^*, \mathbf{b}_a = \mathbf{b}_a^*$$

where $\rho_t = \frac{\pi(A_t|\mathbf{x}(S_t))}{b(A_t|\mathbf{x}(S_t))}$ is the importance sampling ratio.

**Proposition 5.1** *suppose*

1. *assumptions 2.1, 5.1 hold*
2. $\zeta = \mu$.

*then* $\mathbf{w}_{real} = \mathbf{w}_{non\text{-}linear} \neq \mathbf{w}_{linear}$

### 5.2 An Illustrative Example on the Limitation of Linear Models

In order to clearly elucidate the limitation of linear models for planning, we use a simple two-state MDP, as outlined in Figure 1. The policy $b$ used to generate the data for learning the model, and the the policy $\pi$ to be evaluated are also described in Figure 1. We learn a linear model with the data collected by interacting with the real system using policy $b$ and verify that it is the best linear model that could be obtained. We can then obtain $\mathbf{w}_{\text{linear}}$ using equation(3). The solution of the real system is then calculated by *off-policy LSTD*[Yu, 2010] using the same data that is used to learn the linear model. In agreement to proposition 5.1, the two resulting fixed points are considerably different: $\mathbf{w}_{\text{linear}} = [0.953]^\top$ and $\mathbf{w}_{\text{real}} = [8.89]^\top$.

Previous works [Parr *et al.*, 2008; Sutton *et al.*, 2012] showed that a non-action linear expectation model could just be enough if the value function is linear in features. Proposition 5.1 coupled with the above example suggests that this is not true for the more general case of linear expectation models, and expressive non-linear models could potentially be a better choice for planning with expectation models. From now on, we focus on non-linear models as the parametrization of choice for planning with expectation models.

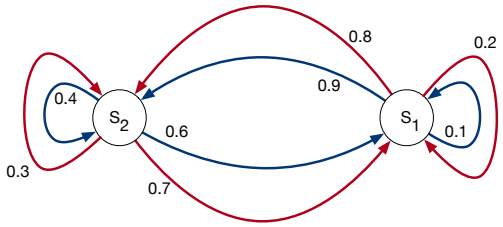


Figure 1: A simple two state MDP in which two actions, *red* and *blue*, are available in each state. Each action causes the agent to probabilistically move to the next-state or stay in the same state. The specific transition probabilities are specified next to the arrows representing the transitions. The reward is zero everywhere except when the *red* action is taken in state $s_1$. The policy $b$ used to generate the data for learning the model is given as: $b(red|s_1) = 0.1, b(blue|s_1) = 0.6, b(red|s_2) = 0.5$ and $b(red|s_2) = 0.5$. The feature vector has only one component: $\mathbf{x}(s_1) = [0.5]^\top$ and $\mathbf{x}(s_2) = [-0.1]^\top$

## 6 Gradient-based Dyna-style Planning Methods

In the previous section, we established that more expressive non-linear models are needed to recover the solution obtained by the real system. An equally crucial choice is that of the planning algorithm: do TD(0) planning updates converge to the fixed-point? For this to be true for linear models, the numerical radius of $\mathbf{F}$ needs to be less than 1 [Sutton *et al.*, 2012]. We conjecture that this condition might not hold in practice causing the planning to diverge. We illustrate this point using the *Baird's Counter Example* [Baird, 1995] in the next section.

Proposition 5.1 implies that the expected TD(0) planning update with the best non-linear model $\mathbb{E}[\Delta_k\boldsymbol{\phi}_k]$ is the same as the expected model-free TD(0) update $\mathbb{E}_b[\rho_t\delta_t\mathbf{x}_t]$, where $\Delta_k = r(\boldsymbol{\phi}_k, A_k) + \gamma\mathbf{w}^\top\hat{\mathbf{x}}(\boldsymbol{\phi}_k, A_k) - \mathbf{w}^\top\boldsymbol{\phi}_k$ and $\delta_t = R_{t+1} + \gamma\mathbf{w}^\top\mathbf{x}_{t+1} - \mathbf{w}^\top\mathbf{x}_t$. However, for off-policy learning, TD(0) is not guaranteed to be stable. This suggests that even with the best non-linear model, the TD(0)-based planning algorithm is also prone to divergence.

Inspired by the *Gradient-TD* off-policy policy evaluation algorithms [Sutton *et al.*, 2009] which are guaranteed to be stable under function approximation, we propose a family of convergent planning algorithms. The proposed methods are guaranteed to converge for both linear and non-linear expectation models. This is true even if the models are imperfect, which is usually the case in model-based reinforcement learning where the models are learned online.

We consider an objective function similar to Mean Square Projected Bellman Error (MSPBE), which we call *Model-Based Mean Square Projected Bellman Error* (MB-MSPBE). Let MB-MSPBE$(\mathbf{w}) = \mathbb{E}[\Delta_k\boldsymbol{\phi}_k]^\top\mathbb{E}[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top]^{-1}\mathbb{E}[\Delta_k\boldsymbol{\phi}_k]$. This objective can be minimized using a variety of gradient-based methods – as we will elaborate later. We call the family of methods optimizing this objective *Gradient-based Dyna-style Planning* (Gradient Dyna) methods.

One observation is that if MB-MSPBE is not strictly convex then minimizing it will give us infinite solutions for $\mathbf{w}$. Since features are assumed to be independent, this

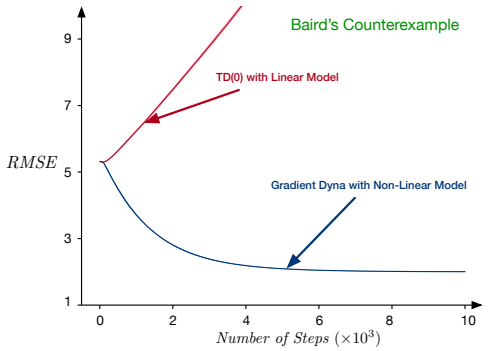


Figure 2: Gradient Dyna vs TD(0)-based planning in *Baird's counterexample*: Gradient Dyna remains stable, whereas TD(0)-based planning algorithm diverges. The reported curve is the average of 30 runs and the standard deviation is too small to be visible clearly

would mean that we have infinite different solutions for the approximate value function and some of them might even have unbounded components. Note that this is also true for MSPBE objective. Similar to the GTD learning methods for MSPBE, we assume that the solution for minimizing MB-MSPBE is unique, denoted by $\mathbf{w}^*$. This is true iff the Hessian $\nabla^2\text{MB-MSPBE}(\mathbf{w}) = \mathbf{A}^\top\mathbf{C}^{-1}\mathbf{A}$, where $\mathbf{A} = \mathbb{E}[\boldsymbol{\phi}_k(\boldsymbol{\phi}_k - \gamma\hat{\mathbf{x}}(\boldsymbol{\phi}_k, A_k))^\top]$ and $\mathbf{C} = \mathbb{E}[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top]^{-1}$, is invertible. This is equivalent to $\mathbf{A}$ being non-singular.

**Assumption 6.1** $\mathbf{A}$ *is non-singular.*

It can be shown that the solution for minimizing this objective is $\mathbf{A}^{-1}\mathbf{c}$, where $\mathbf{c} = \mathbb{E}[r(\boldsymbol{\phi}_k, A_k)\boldsymbol{\phi}_k]$, if the above assumption holds. Note that the solution is the same as the TD(0)-based planning solution if $\hat{\mathbf{x}}(\boldsymbol{\phi}, a) = \mathbf{F}_a\boldsymbol{\phi},\ r(\boldsymbol{\phi}, a) = \mathbf{b}_a$, but since Gradient Dyna optimizes this objective by gradient descent, the numerical radius condition is not required anymore.

We note that there is an equivalence between MB-MSPBE and MSPBE, where $\text{MSPBE}(\mathbf{w}) = \mathbb{E}_b[\rho_t\delta_t\mathbf{x}_t]^\top\mathbb{E}_b[\mathbf{x}_t\mathbf{x}_t^\top]^{-1}\mathbb{E}_b[\rho_t\delta_t\mathbf{x}_t]$. That is, if a best non-linear model is learned from the data generated from some policy $b$ and $\zeta$ in the search control process equals $b$'s stationary feature vector distribution $\mu$, then MB-MSPBE is the MSPBE.

**Proposition 6.1** *If*

1. *Assumption 2.1 and 2.3 hold.*
2. $\zeta = \mu$
3. $\hat{\mathbf{x}}(\boldsymbol{\phi}, a) = \hat{\mathbf{x}}^*(\boldsymbol{\phi}, a)$ *and* $\hat{r}(\boldsymbol{\phi}, a) = \hat{r}^*(\boldsymbol{\phi}, a)$

*Then MB-MSPBE*$(\mathbf{w})$ = *MSPBE*$(\mathbf{w})$.

The proposition 6.1 does not hold for the best linear model for the same reason elaborated in proposition 5.1.

Let's now consider algorithms that can be used to minimize this objective. Consider the gradient of MB-MSPBE: $\nabla\text{MB-MSPBE}(\mathbf{w}) = \mathbb{E}[(\gamma\hat{\mathbf{x}}(\mathbf{x}_k, A_k) - \mathbf{x}_k)\mathbf{x}_k^\top]\mathbb{E}[\mathbf{x}_k\mathbf{x}_k^\top]^{-1}\mathbb{E}[\Delta_k\boldsymbol{\phi}_k]$. In the above expression, we have a product of three expectations and, therefore, we cannot simply use one sample to obtain an unbiased estimate of the gradient. In order to obtain an unbiased estimate of the gradient, we could either draw three independent samples or learn the product of the the last two factors using a linear least-square method. GTD methods take the second route leading to an algorithm with $O(m)$ complexity in which two sets of parameters are mutually related in their updates. However, if one uses a linear model, the computational complexity for storing and using the model is already $O(m^2)$. For a non-linear model, depending on the parameterization choices, the complexity can be either smaller or greater than $O(m^2)$. Thus, a planning algorithms with $O(m^2)$ complexity can be an acceptable choice. This leads to two choices: we can sample the three expectations and and then combine them to produce an unbiased estimate of the gradient. Note that this would still lead to an $O(m^2)$ algorithm as the matrix inversion can be done in $O(m^2)$ using Sherman-Morrison formula. Alternatively, we can use the linear least-square method to estimate the first two expectations and sample the third one. In this case, there are still two sets of parameters but their updates are not mutually dependent. This can potentially lead to faster convergence. Although both of these approaches have $O(m^2)$ complexity, we adopt the second approach, which is summarized in algorithm 1. We now present the convergence theorem for the proposed algorithm, which is followed by its empirical evaluation. The reader can refer to the supplementary materials for the proof of the theorem.

**Algorithm 1** Gradient Dyna Algorithm

**Input**: $\mathbf{w}_0$, policy $\pi$, feature vector distribution $\zeta$, expectation model $\{\hat{\mathbf{x}}, \hat{r}\}$, stepsizes $\alpha_k, \beta_k$ for $k = 1, 2, \cdots$
**Output**: $\mathbf{w}_k$

**for** $k = 1, 2, \cdots$ **do**
  Sample $\boldsymbol{\phi}_k \sim \zeta(\cdot)$
  Sample $A_k \sim \pi(\cdot|\boldsymbol{\phi}_k)$

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k\mathbf{V}_k\Delta_k\boldsymbol{\phi}_k$$
$$\mathbf{V}_{k+1} \leftarrow \mathbf{V}_k + \beta_k((\gamma\hat{\mathbf{x}}(\boldsymbol{\phi}_k, A_k) - \boldsymbol{\phi}_k)\boldsymbol{\phi}_k^\top - \mathbf{V}_k\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top)$$

**end for**

**Theorem 6.1 (Convergence of Gradient Dyna Algorithm)** *Consider algorithm 1. If*

1. *Assumptions 2.2, 2.3 and 6.1 hold*
2. $\alpha_k > 0, \beta_k > 0, \sum_{k=0}^{\infty}\alpha_k = \infty, \sum_{k=0}^{\infty}\beta_k = \infty, \sum_{k=0}^{\infty}\alpha_k^2 < \infty, \sum_{k=0}^{\infty}\beta_k^2 < \infty$

*Then for any initial weight vector* $\mathbf{w}_0$, $\lim_{k\to\infty}\mathbf{w}_k = \mathbf{w}^*$ *w.p. 1.*

## 7 Experiments

The goal of the experiment section is to validate the theoretical results and investigate how Gradient Dyna algorithm performs in practice. Concretely, we seek to answer the following questions: 1) is the proposed planning algorithm stable for the non-linear model choice, especially when the model is learned online and 2) what solution does the proposed planning algorithm converge to.

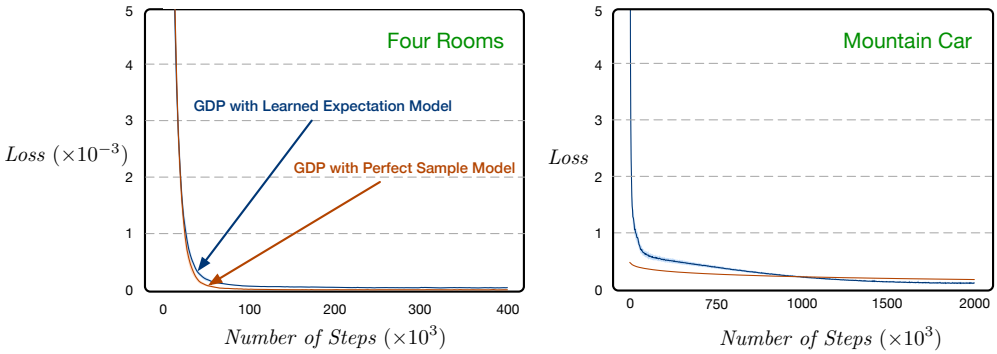


Figure 3: Gradient Dyna with a learned expectation model remains stable and converges to the off-policy LSTD solution. The reported curve is the average of 30 runs and the standard deviation is too small to be visible clearly

### 7.1 Divergence in TD(0)-based Planning

Our first experiment is designed to illustrate the divergence issue with TD(0)-based planning update. We use *Baird's counterexample* [Baird, 1995; Sutton and Barto, 2018] with the same dynamics and reward function - a classic example to highlight the off-policy divergence problem with the model-free TD(0) algorithm. The policy $b$ used to learn the model is arranged to be the same as the behavior policy in the counterexample, whereas the policy $\pi$ to be evaluated is arranged to be the same as the counterexample's target policy. For TD(0) with linear model, we initialize the matrix $\mathbf{F}_a$ and vector $\mathbf{b}_a$ for all $a$ to be zero. For Gradient Dyna, we use a neural network with one hidden layer of 200 units as the non-linear model. We initialize the non-linear model using Xavier initialization [Glorot and Bengio, 2010]. The parameter $\mathbf{w}$ for the estimated value function is initialized as proposed in the counterexample. The model is learned in an online fashion, that is, we use only the most recent sample to perform a gradient-descent update on the mean-square error. The search-control process is also restricted to generate the last-seen feature vector, which is then used with an $a \sim \pi$ to simulate the next feature vector. The resulting simulated transition is used to apply the planning update. The evaluation metric is the Root Mean Square Error (RMSE): $\sqrt{\sum_s (\hat{v}(s, \mathbf{w}) - v_\pi(s))^2 / |\mathcal{S}|}$. The results are reported for hyperparameters chosen based on RMSE over the latter half of a run. In Figure 2, we see that TD(0) updates with the linear expectation model cause the value function to diverge. In contrast, Gradient Dyna remains sound and converges to the RMSE of 2.0. Interestingly, stable model-free methods also converge to the same RMSE value (not shown here) [Sutton and Barto, 2018].

### 7.2 Convergence in Practice

In this set of experiments, we want to investigate how Gradient Dyna algorithm performs in practice. We evaluate the proposed method for the non-linear model choice in two simple yet illustrative domains: stochastic variants of Four Rooms [Sutton *et al.*, 1999; Ghiassian *et al.*, 2018] and Mountain Car [Sutton, 1996]. Similar to the previous experiment, the model is learned online. Search control, however, samples uniformly from the recently visited 1000 feature vectors to approximate the i.i.d. assumption in Theorem 6.1.

We modified the Four Rooms domain by changing the states on the four corners to terminal states. The reward is zero everywhere except when the agent transitions into a terminal state, where the reward is one. The episode starts in one of the non-terminal states uniform randomly. The policy $b$ to generate the data for learning the model takes all actions with equal probability, whereas the policy $\pi$ to be evaluated constitutes the shortest path to the top-left terminal state and is deterministic. We used tile coding [Sutton, 1996] to obtain features(4 $2 \times 2$ tilings). In mountain car, the policy $b$ used to generate the data is the standard energy-pumping policy with 50% randomness [Le *et al.*, 2017], where the policy $\pi$ to be evaluated is also the standard energy-pumping policy but with no randomness. We again used tile coding to obtain features (8 $8 \times 8$ tilings). We inject stochasticity in the environments by only executing the chosen action 70% of the times, whereas a random action is executed 30% of the time. In both experiments, we do one planning step for each sample collected by the policy $b$. As noted in proposition 6.1, if we have $\zeta = \mu$, the minimizer of MB-MSPBE for the best non-linear model is the off-policy LSTD solution $\mathbf{A}_{\text{LSTD}}^{-1}\mathbf{c}_{\text{LSTD}}$ $\mathbf{A}_{\text{LSTD}} = \mathbb{E}_b[\rho_t \mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]$, $\mathbf{c}_{\text{LSTD}} = \mathbb{E}_b[\rho_t R_{t+1} \mathbf{x}_t]$ [Yu, 2010]. Therefore, for both domains, we run the off-policy LSTD algorithm for 2 million time-steps and use the resulting solution as the evaluation metric: $Loss = \|\mathbf{A}_{\text{LSTD}}\mathbf{w} - \mathbf{c}_{\text{LSTD}}\|_2^2$.

The results are reported for hyper-parameters chosen according to the LSTD-solution based loss over the latter half of a run. In Figure 3, we see that Gradient Dyna remains stable and converges to off-policy LSTD solution in both domains.

## 8 Conclusion

In this paper, we proposed a sound way of using the expectation models for planning and showed that it is equivalent to planning with distribution models if the state value function is linear in state-features. We made a theoretical argument for non-linear expectation models to be the parametrization of choice even if the value-function is linear. Lastly, we proposed Gradient Dyna, a model-based policy evaluation algorithm with convergence guarantees, and empirically demonstrated its effectiveness.

## 9 Acknowledgement

We would like to thank Huizhen Yu, Sina Ghiassian, Banafsheh Rafiee and Khurram Javed for useful discussions and feedbacks.

## References


[Baird, 1995] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[Chua *et al.*, 2018] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4759–4770, 2018.

[Chung *et al.*, 2018] Wesley Chung, Somjit Nath, Ajin Joseph, and Martha White. Two-timescale networks for nonlinear value function approximation. 2018.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[Deisenroth and Rasmussen, 2011] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[Ghiassian *et al.*, 2018] Sina Ghiassian, Andrew Patterson, Martha White, Richard S. Sutton, and Adam White. Online off-policy prediction. *CoRR*, abs/1811.02597, 2018.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[Jaderberg *et al.*, 2016] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[Konidaris *et al.*, 2011] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[Le *et al.*, 2017] Lei Le, Raksha Kumaraswamy, and Martha White. Learning sparse representations in reinforcement learning with sparse coding. *arXiv preprint arXiv:1707.08316*, 2017.

[Leibfried *et al.*, 2016] Felix Leibfried, Nate Kushman, and Katja Hofmann. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*, 2016.

[Modayil *et al.*, 2014] Joseph Modayil, Adam White, and Richard S Sutton. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 22(2):146–160, 2014.

[Moore and Atkeson, 1993] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

[Oh *et al.*, 2015] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[Pan *et al.*, 2018] Yangchen Pan, Muhammad Zaheer, Adam White, Andrew Patterson, and Martha White. Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains. *arXiv preprint arXiv:1806.04624*, 2018.

[Parr *et al.*, 2008] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 752–759. ACM, 2008.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[Sutton *et al.*, 1999] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[Sutton *et al.*, 2009] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000. ACM, 2009.

[Sutton *et al.*, 2011] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[Sutton *et al.*, 2012] Richard S Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael P Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*, 2012.

[Sutton, 1991] R.S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In *Advances in Neural Information Processing Systems*, 1991.

[Sutton, 1996] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[White and others, 2015] Adam White et al. Developing a predictive approach to knowledge. 2015.

[Yu, 2010] Huizhen Yu. Convergence of least squares temporal difference methods under general conditions. In *ICML*, pages 1207–1214, 2010.