POLICY TREE NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Decision-time planning policies with implicit dynamics models have been shown to work in discrete action spaces with Q learning. However, decision-time planning with implicit dynamics models in continuous action space has proven to be a difficult problem. Recent work in Reinforcement Learning has allowed for implicit model based approaches to be extended to Policy Gradient methods. In this work we propose Policy Tree Network (PTN). Policy Tree Network lies at the intersection of Model-Based Reinforcement Learning and Model-Free Reinforcement Learning. Policy Tree Network is a novel approach which, for the first time, demonstrates how to leverage an implicit model to perform decision-time planning with Policy Gradient methods in continuous action spaces. This work is empirically justified on 8 standard MuJoCo environments so that it can easily be compared with similar work done in this area. Additionally, we offer a lower bound on the worst case change in the mean of the policy when tree planning is used and theoretically justify our design choices.

1 INTRODUCTION

Reinforcement Learning (RL), the study of learning what to do. Learning what to do in a given situation so as to maximize reward signals. Generally speaking, Reinforcement Learning problems are approached from either a Model-Free or Model-Based perspective. A Model-Free approach builds a policy through interacting with the environment and uses the acquired experience to improve the policy. A Model-Based approach builds a dynamics model of the environment and uses this improve a policy in a number of ways. Such as planning, exploration, and even training on imagined data (Sutton, 1990; Ha & Schmidhuber, 2018). Model-Free Reinforcement Learning generally offers superior final performance. However, in exchange for strong final performance a large number of samples are required. Model-Based Reinforcement Learning offers better sample complexity but comes at the cost of a weaker final policy. In light of this, recent research (Wellmer & Kwok, 2019; Silver et al., 2016; Farquhar et al., 2017; Oh et al., 2017) has been focused on mixing both Model-Free and Model-Based Reinforcement Learning.

Model-Based Reinforcement Learning can be broken down further into two categories (Wellmer & Kwok, 2019). The first is explicit dynamics models. Explicit dynamics models are when the observations are directly being reconstructed. The second is implicit dynamics models. Implicit dynamics models are when the dynamics model is learned indirectly. Some examples of this could be through predicting future rewards, values, policies or other auxiliary signals.

The work we present here, Policy Tree Network, lies at the intersection of Model-Free and Model-Based Reinforcement Learning. Our work builds off the work done by Wellmer & Kwok (2019), where they showed how to build an implicit dynamics model for Policy Gradient methods in continuous action space. Policy Tree Network takes this a few steps further by showing how to leverage the implicit dynamics model for decision-time planning.

Our empirical results are on eight MuJoCo (Todorov et al., 2012) environments. In our experiments we validate the decision-time planning scheme that we introduce. Lastly we show Policy Tree Network outperforms the model-free baseline (Schulman et al., 2017) and the mixed model-free model-based baseline (Wellmer & Kwok, 2019).

2 Related Works

Value Prediction Network (VPN) (Oh et al., 2017) offers an approach to building and leveraging an implicit dynamics model in a discrete action space. The implicit dynamics model is constructed with overshooting objectives which predict future rewards and value estimates. At behavior time, a Q tree is expanded and actions are selected according to ϵ-greedy of the backed-up Q estimates. Decision-time planning is trivial in this case since the action space is discrete. When the action space is discrete it's possible to try all possibilities.

TreeQN and ATreeC offer a policy gradient and Q-learning approach building implcit dynamics models in discrete action spaces. TreeQN (Farquhar et al., 2017) is similar to VPN except the authors claim that VPN has an issue since the policy being trained is different than the policy actually being used. In light of this, TreeQN directly optimizes the backed-up Q estimates as opposed to indirectly (as is done in VPN). Additionally, TreeQN uses overshooting reward estimates as an auxiliary objective.

ATreeC (Farquhar et al., 2017) takes a similar approach to TreeQN except instead of a Q-learning approach ATreeC uses a policy gradient approach. We note that even though ATreeC uses a policy gradient approach it still does not work with continuous action spaces. Crucially, ATreeC optimizes the backed-up "Q" estimates to function as logits for a multinomial distribution. Because of this, the backed-up "Q" values in this case can be thought of as pseudo Q values.

Policy Prediction Network (PPN) (Wellmer & Kwok, 2019) is an approach to building implicit dynamics models with policy gradient methods. This was the first work to show how to build an implicit dynamics model in continuous action spaces. This was done by introducing overshooting objectives and a clipping scheme designed for overshooting objectives. The training algorithm is located in Section A.2.

$$\mathcal{L}_t = \alpha_v \mathcal{L}_t^{d,v} + \sum_{i=0}^{d-1} (\mathcal{L}_t^{i,\pi} + \alpha_v \mathcal{L}_t^{i,v} + \alpha_r \mathcal{L}_t^{i,r}) \tag{1}$$

$$\mathcal{L}_t^{i,\pi} = \frac{1}{2} \max(-\text{ratio}_t^i A_{t+i}^{\text{GAE}}, -\text{ratio}_{t,\text{clip}}^i A_{t+i}^{\text{GAE}}) - \alpha_h H_t^i \tag{2}$$

$$\mathcal{L}_t^{i,v} = \frac{1}{2} \max((\hat{v}_{t,\theta}^i - R_{t+i})^2, (\hat{v}_{t,\text{clip}}^i - R_{t+i})^2) \tag{3}$$

$$\mathcal{L}_t^{i,r} = \frac{1}{2} \max((\hat{r}_{t,\theta}^i - r_{t+i})^2, (\hat{r}_{t,\text{clip}}^i - r_{t+i})^2) \tag{4}$$

Where $\mathcal{L}_t^{i,\pi}$, $\mathcal{L}_t^{i,v}$, and $\mathcal{L}_t^{i,r}$ are the policy, value, and reward losses grounded at time t predicting i steps into the future. Additionally, subscript θ refers to estimates from the latest model parameters, H is policy entropy, A_{t+i}^{GAE} is the generalized advantage estimate (Schulman et al., 2016), r is a reward target, $\hat{r}$ is a predicted reward, $\hat{v}$ is a predicted value, and R is a boot-strapped n-step return target. The α coefficients are hyper-parameters used to trade off importance of objectives. The clipped estimates are used to stay near estimates from old parameters(θ') and are further defined in Section A.3 along with the definition of the importance sampling ratio.

The short coming of PPN is that it did not lend itself well to leveraging the implicit dynamics model to perform decision-time planning and thus used the model-free policy to sample actions. Additionally, it does not make use of the Q function, a separate measure of how "good" an action is at a given state, as discussed in Section 3.2.

3 Policy Tree Network

Policy Tree Network uses a combination of model-free and model-based techniques. Policy Tree Network builds off the PPN (Wellmer & Kwok, 2019) approach to learning an implicit-model. Actions during behavior time are chosen by a model-free policy. Learning is done with a model-based approach that follows the behavior policy's rollout trajectory. However, the test policy follows a model-based approach. An implicit transition model is embedded into the architecture. Through overshooting objectives, a dynamics model is learned and the collection of forward predictions offer the benefit of additional signal in the gradient updates.

Our novel contribution in this works is a decision-time planning algorithm that allows us to leverage the implicit dynamics model. The decision-time planning algorithm is based on building a tree of possible future values, actions, states, and rewards and then performing a tree backup algorithm we introduce. In previous works it was not possible to directly leverage the implicit model for decision time planning with policy gradient methods in continuous action space. Our empirical results in Section 4 demonstrate the advantage of PTN over the model-free baseline (PPO) and the mixed model-free & model-based baseline (PPN).

3.1 Learning

Policy Tree Network is trained over a collection of overshooting objectives. We follow an identical clipping approach (Section A.3) to the one found in PPN. Targets used for training are computed exactly the same way as in PPN. More explicitly, no aspect of training depends on the decision-time planning algorithm described later in Section 3.2. While this might be desirable, we describe why it is difficult in A.1. We additionally introduce a term β Hafner et al. (2018) used to trade off importance of short and long-term predictions. Lastly, for simplicity we drop the extra value loss term found in PPN (Equation 1), leaving us with the following: $\mathcal{L}_t = \sum_{i=0}^{d-1} \beta_i(\mathcal{L}_t^{i,\pi} + \alpha_v \mathcal{L}_t^{i,v} + \alpha_r \mathcal{L}_t^{i,r})$. Where $\mathcal{L}_t^{i,\pi}$, $\mathcal{L}_t^{i,v}$, and $\mathcal{L}_t^{i,r}$ (equations 2, 3, 4) are the policy, value, and reward losses grounded at time t predicting i steps into the future.

The implicit dynamics model is jointly learned from the objectives in L_t whenever $i > 0$. When $i = 0$ estimates are not required to pass through the transition network. In Section A.2 we show the training algorithms for PPO (Schulman et al., 2017), PPN (Wellmer & Kwok, 2019), and PTN which are quite similar.

PPN (Wellmer & Kwok, 2019) was shown to have returns drastically differ between training depth values. The point of β is to stabilize returns over different choices of training depth (Section A.6).

3.2 Decision-Time Planning

Including branching in PTN is appealing because it would allow for decision-time planning. Furthermore, this *appears* to be a simple task since a transition model is already being learned. However, because the decision-time planning policy is difficult to directly measure with a PDF (Section A.1), we resort to performing decision-time planning exclusively at evaluation time.

Traditionally, at every state, the next action in a state is either determined by a policy, π, learned usually based on a policy gradient method, or an action value function Q that induces a policy that takes actions with the largest Q value. However, in PTN, we have access to independently learned policy and Q functions.[1] There are three ways of using π and Q to create the final policy, π_F.

1. setting $\pi_F = \pi$ as done in PPN, (Wellmer & Kwok, 2019), which ignores the Q function.
2. setting π_F = argmax Q, called Q-backup which ignores π[2].
3. setting π_F = $f(\pi, Q)$, called π-Q-backup, for some function f.

As (1) was described in PPN we will further dig into the details of (2) in Section 3.2.1 and (3) in Section 3.2.2. Methods discussed in this section do not impact the learning procedure discussed in Section 3.1 and are only used for decision-time planning at evaluation time.

We use a tree planning method which expands a tree up to a depth d expanding b branches at each depth and collecting reward, policy, and value predictions along the way. Tree expansion is done by recursively calling the core module (f^{core}) with b action samples and the abstract state. The action being passed to the core module could be sampled from a uniform distribution or the policy ($\pi(a|\hat{s}_{t,\theta}^{i,j})$), where $\hat{s}_{t,\theta}^{i,j}$ represents the predicted abstract-state i steps into the future with branch j according to parameters θ. We show that if the correlation between π and Q is positive, sampling π will be more suitable and Section 4.2 shows that they are in fact positively correlated in practice. Candidate actions represent branches in the tree. Then the predicted estimates are backed-up with a TD-λ scheme (Sutton, 1988; Sutton & Barto, 2018; Farquhar et al., 2017) to assign a Q value to each branch as described in Algorithm 1.

[1]by independent we mean π is not just a policy induced by selecting max of Q.

[2]when $b = \infty$

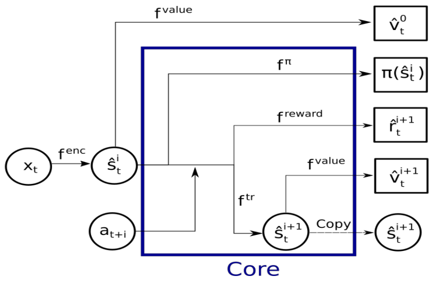

Figure 1: Predict policies, rewards, abstract states, and the value of the abstract states (Wellmer & Kwok, 2019).

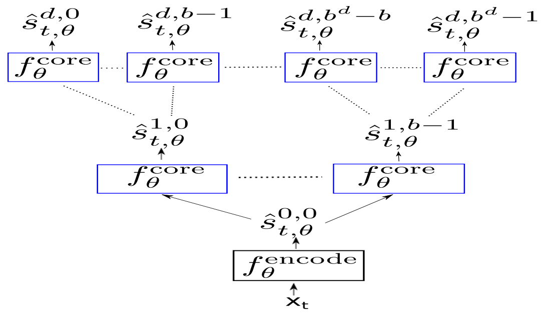

Figure 2: PTN Expansion, a, r, v outputs from f^{core} are omitted for simplicity

When performing decision-time planning, the action associated with the max backed-up Q-value is taken. This policy is referred to as $\pi_{\theta',F}$, where θ' denotes which parameters were used in the tree.

3.2.1 Q Backup

Given an abstract state $s = f_\theta^{enc}(x)$ and an action a, the backed-up Q-value calculated from d-step planning is defined as:

$$Q^i(s,a) = r + \gamma V^i(s') \quad (5) \qquad V^i(s) = \begin{cases} v & \text{if } i = d \\ (1-\lambda)v + \lambda \max_a Q^{i+1}(s,a) & \text{if } i < d \end{cases} \quad (6)$$

where $s' = f_\theta^{tr}(s,a), v = f_\theta^{v}(s)$, and $r = f_\theta^{r}(s,a)$.

In Q backup, the final policy, π_F, aims at selecting an action that maximizes the action value function, Q^0. However, Q^0 is a combination of neural networks and finding $\arg\max_a Q^0$ is a difficult problem. We use our tree expansion to achieve this. With tree expansion, at each depth i, we sample b actions based on π and choose the one with maximum Q^{i+1} to estimate $\arg\max_a Q^{i+1}$ in equation 6.

Intuitively, if π and Q are positively correlated, sampling from π allows us to find actions with higher Q compared with sampling from a uniform distribution, because, for a state s and an action a, larger $\pi(a|s)$ value would suggest larger $Q(s,a)$. This is the subject of the following lemma. Note that, for a given state s, the covariance between $\pi(X|s)$ and $Q(X,s)$ over a bounded action space is defined as $cov_U(\pi(X|s), Q(X,s))$ where X is a random variable with distribution U, the uniform distribution over the action space. π and Q are positively correlated if $cov_U(\pi(X|s), Q(X,s))$ is positive. Proof of the lemma can be found in the appendix.

Lemma 1. $E_{X\sim\pi}[Q(X,s)] > E_{X\sim U}[Q(X,s)]$ *if* $cov_U(\pi(X|s), Q(X,s)) > 0$.

Section 4.2 shows that the correlation between π and Q^0 is in fact positive in practice. Therefore, sampling actions according to π is justified.

Since sampling is done based on π, our final policy π_F now depends on π as well as Q^0. Next we investigate the relationship between π_F, π and Q^0. Consider the abstract state $s = \hat{s}_{t,\theta}^{0,0}$. Recall that $Q^0(a,s)$ is an action value function calculated based on the tree expansion and backup. Q^0 uses sampling, is probabilistic, and is a function of f_θ^{core}, λ, b and d. Action selection is done by sampling $X_i \sim \pi$ for $1 \le i \le b$ and selecting $\arg\max_i Q^0(X_i, s)$. Thus, for an action a, cumulative density function (cdf) of $\pi_F(a|s)$ is given by $\int_{-\infty}^{a} \pi_F(z|s)dz = Pr(\arg\max_i Q^0(X_i,s) \le a)$.

Intuitively, the branching factor b can be thought of as interpolating how much *confidence* we have in π and reward, value and transition networks (which make up Q^0): low b signifies confidence in π and larger b signifies more confidence in reward, value and transition networks. As b increases π_F becomes less dependant on π. When b goes to infinity, π_F only depends on reward, value and transition networks and not π. To investigate the maximum possible difference between mean of π and π_F over all Q^0 functions we provide the next theorem. The theorem is provided in a one dimensional action space for better illustration, but the argument can be extended to higher dimensions. Proof of the theorem can be found in the appendix.

Theorem 1. *Let μ_F be the mean of actions selected based on π_F and μ, σ the mean and standard deviation of actions selected based on π. In the worst-case over all possible Q^0 functions,* $|\mu - \mu_F| \ge$

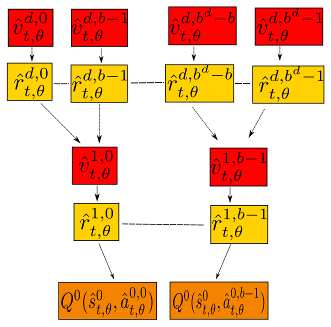
$\hat{v}^{d,0}_{t,\theta}$
$\hat{v}^{d,b-1}_{t,\theta}$
$\hat{v}^{d,b^d-b}_{t,\theta}$
$\hat{v}^{d,b^d-1}_{t,\theta}$
$\hat{r}^{d,0}_{t,\theta}$
$\hat{r}^{d,b-1}_{t,\theta}$
$\hat{r}^{d,b^d-b}_{t,\theta}$
$\hat{r}^{d,b^d-1}_{t,\theta}$
$\hat{v}^{1,0}_{t,\theta}$
$\hat{v}^{1,b-1}_{t,\theta}$
$\hat{r}^{1,0}_{t,\theta}$
$\hat{r}^{1,b-1}_{t,\theta}$
$Q^0(\hat{s}^0_{t,\theta}, \hat{a}^{0,0}_{t,\theta})$
$Q^0(\hat{s}^0_{t,\theta}, \hat{a}^{0,b-1}_{t,\theta})$

Figure 3: PTN Backup

Algorithm 1 π_Q_expand(s, i)

```
initialize $\hat{a}$, $r$, $s$, $v'$, $Q$
for j in b do
    $\hat{a}[j] \sim \pi(s)$
    $\hat{r}[j] = f^{r}_{\theta}(s, \hat{a}[j])$
    $\hat{s}'[j] = f^{tr}_{\theta}(s, \hat{a}[j])$
    $\hat{v}'[j] = f^{va}_{\theta}(\hat{s}'[j])$
    if $i + 1 == d$ then
        $Q[j] = \sqrt{\Big(\hat{r}[j] + \gamma\hat{v}'[j]\Big)\pi(a = \hat{a}[j]|s)}$
    else
        $Q\_\text{tmp} = \pi\_\text{Q\_expand}(\hat{s}'[j], i + 1)$
        $Q[j] = \sqrt{\Big(\hat{r}[j] + \gamma((1-\lambda)\hat{v}'[j] + \lambda \max_{\hat{a}} Q\_\text{tmp})\Big)\pi(a = \hat{a}[j]|s)}$
return Q
```

Figure 4: pseudo-code for π-Q expansion and backup algorithm. Where the Q returned is a b dimensional vector.

$b|\sigma \int_{-\infty}^{\infty} z\phi(z)(\frac{erfc(z)}{2}(\frac{z}{\sqrt{2}}))^{b-1}dz|$, where erfc is the complementary error function and ϕ is the p.d.f of standard normal distribution. Setting $b = 2$, we get the looser bound $|\mu - \mu_F| \geq \frac{\sigma}{\sqrt{\pi}}$ in the worst-case.

The bound is a lower bound on the worst-case and it shows that μ_F *can* move at least as far as $\frac{\sigma}{\sqrt{\pi}}$ from μ. In other words, there exist Q^0 functions for which the difference between mean of π_F and π is at least $\frac{\sigma}{\sqrt{\pi}}$. Observe that the bound decreases as σ decreases, which intuitively makes sense since with lower σ, it takes more number of samples to obtain an action further away from μ.

The bound above sheds a light on one of the downsides of this approach. The final policy π_F *can* become significantly different from π, depending on Q^0 and σ, even when $b = 2$ and as b grows it becomes only reliant on Q^0 (Section A.5 shows the numerical value of the bound for several b values). Although Section 4.2 shows empirically that π and Q^0 are *on average* correlated, there can be states where they are not. In such a scenario, Q backup may rely too heavily on Q^0.

We previously mentioned that b can be seen as a confidence parameter, adjusting how much we rely on π and the networks that make up Q^0. Nevertheless, it does not remedy the above mentioned problem. This is for two reasons, one is that b only takes discrete values. We mentioned above that $b = 2$ may shift the policy by too much, but we cannot reduce b any more without exclusively relying on π. Secondly, choosing a b for a given confidence level is a difficult problem, even if a discrete confidence parameter is sufficient. This is because b has a complicated relationship with π_F, π and Q involving integrals. Thus, it is not clear that, for instance, if we have equal confidence in Q and π, what the value of b should be.

3.2.2 π-Q BACKUP

Motivated by the issues discussed in section 3.2.1, we introduce a π-Q backup. Given a policy π and a Q-function Q, we want our final policy π_F to depend on Q and π based on how much confidence we have in each. If we have equal confidence in both Q and π, then we would like π_F to be a function depending on π and Q *equally*. One way to achieve this is using geometric mean of π and Q as an indicator of how good an action is [3]. That is, we would like to select an action such that $\sqrt{\pi \times Q}$ is maximized (note that if we have reasons to believe π or Q provide better estimates, we can take the weighted geometric mean).

The backup procedure is defined as mentioned in Section 3.2.1 , however, given an abstract state $s = f^{enc}_{\theta}(x)$ and an action a, the backed-up Q-value calculated from d-step planning is defined as:

$$Q^i(s,a) = \sqrt{(r + \gamma V^i(s'))\pi(a|s)} \quad (7) \qquad V^i(s) = \begin{cases} v & \text{if } i = d \\ (1-\lambda)v + \lambda \max_a Q^{i+1}(s,a) & \text{if } i < d \end{cases} \quad (8)$$

Note that neither Q^i or V^i in Equations (5, 6, 7, 8) are directly being optimized, they are only used for decision-time planning. Notice how estimates are scaled by likelihood of the action being sampled from π. This removes the previously mentioned issue of selected actions no longer depending on π. It's interesting to note that in this case when $b = \infty$ the policy is deterministic.

[3]When a negative or zero valued Q exists in the tree backup, modify the geometric mean by adding $|\min(Q)| + \delta$ to the Q estimates. δ is a positive small non-zero term which can be optimized Habib (2012)

This does not manifest itself as an issue because the decision-time planning policy isn't used as the behavior policy so we are less concerned about exploration.

Note that we still need to estimate $\max_a Q^{i+1}$ in equation 8. We take a similar approach to Q-backup mentioned in Section 3.2.1, since π and $\sqrt{\pi Q}$ are expected to be correlated. Furthermore, regarding the use of geometric mean, we note that it is more meaningful than the arithmetic mean in this scenario as the values may be of different scales and that it preserves proportional change as opposed to absolute change.

In Section 4.3 we will empirically explore both approaches to backup. The π-Q backup is shown in Algorithm 4. Though we note that if the scaling of Q estimates by π and the square root are dropped it would reduce to the max Q-backup algorithm described in Section 3.2.1.

4 EXPERIMENTS

Our experiments seek to answer the following questions: (1) Is correlation between π and Q positive? (2) What style of backup performs better: a standard Q-value backup or a policy weighted Q-value backup and how does branching effect the returns of the decision-time planning policy? (3) Does PTN outperform the baselines?

4.1 EXPERIMENTAL SETUP

Preprocessing in our experiments was done similarly to that of PPO (Schulman et al., 2017) and identical to the preprocessing in PPN (Wellmer & Kwok, 2019). All models, PPO2, PPN, and PTN were implemented in Pytorch (Paszke et al., 2017). The parameters (θ) are updated with the Adam optimizer (Kingma & Ba, 2014). All the experiments are run for 1 million time steps unless otherwise noted.

Our PPO2 implementation uses the same hyperparameters as the baselines implementation (Dhariwal et al., 2017): 3 fully connected layers with 128 hidden units and tanh activations for the policy. 3 fully connected layers 128 hidden units and tanh activations for the critic. The largest difference in our PPO2 implementation is that we do not perform orthogonal initialization.

Our PTN implementation uses similar hyperparameters (identical to PPN): 2 fully connected layers with 128 hidden units and tanh activations for the embedding. 2 fully connected residual layers with 128 hidden units, tanh activations, and unit length projections of the abstract-state (Farquhar et al., 2017) for the transition module. 1 fully connected layer with 128 hidden units for the policy mean. 1 fully connected layer with 128 hidden units for the value. 1 fully connected layer with 128 hidden units for the reward. In practice we use Huber losses instead of L2 losses, as was done in related implicit model based works (Oh et al., 2017).

We set $\beta_0 = d$ and $\beta_{>0} = 1$. Values used for branching and depth are explicitly stated in corresponding experiments. We test on the 8 standard OpenAI MuJoCo environments (Todorov et al., 2012; Brockman et al., 2016): Hopper, Walker2d, Swimmer, HalfCheetah, InvertedPendulum, InvertedDoublePendulum, Humanoid, and Ant.

4.2 CORRELATION

	sample correlation mean	sample correlation variance
Hopper-v2	0.937	0.018
Walker2d-v2	0.971	0.017
Swimmer-v2	0.945	0.016
HalfCheetah-v2	0.968	0.018
InvertedPendulum-v2	0.969	0.008
InvertedDoublePendulum-v2	0.971	0.005
Humanoid-v2	0.934	0.022
Ant-v2	0.972	0.019

Table 1: Summary of correlation between Q and π.

In this experiment we empirically measure the correlation between π and Q^0(with $d = 1$). Positive correlation empirically justifies the use of lemma 1 that sampling from π offers a better approach to maximizing the backup objective (for both Q backup and πQ backup) as opposed to uniform sampling. To measure correlation we first train a PPN policy for 200 thousand time-steps on the MuJoCo environments. Next we collect trajectories over 10 thousand timesteps. Then at each observation we uniformly sample 100 actions and compute the corresponding Q-values and corresponding PDF points from π. This gives us 10 thousand estimates of correlation. From here we fit a normal distribution to the correlation results and report them in Table 3.

Positive correlation tells us that sampling according to π will better maximize π_F than a uniform sample over actions. This empirically justifies our approach to performing tree expansion.

4.3 Branching & Backup

	PTN π-Q backup				PTN Q Backup			
	b=1	b=2	b=3	b=5	b=1	b=2	b=3	b=5
Hopper	2296.8 ± 403.6	2784.4 ± 357.8	2981.3 ± 306.7	3083.5 ± 244.1	2155.9 ± 343.9	1763.0 ± 431.9	1400.8 ± 423.5	1103.4 ± 398.1
Walker2d	3442.7 ± 182.0	4080.3 ± 295.5	4189.1 ± 355.9	4253.8 ± 334.3	3381.5 ± 378.3	3539.7 ± 517.6	3160.7 ± 743.5	2722.8 ± 1033.0
Swimmer	102.1 ± 28.4	105.6 ± 29.2	106.6 ± 29.2	107.5 ± 29.2	89.5 ± 35.1	88.6 ± 34.4	84.5 ± 32.0	75.5 ± 30.9
HalfCheetah	4082.1 ± 659.9	4346.3 ± 649.1	4459.8 ± 638.8	4555.0 ± 643.6	3435.8 ± 742.4	3501.7 ± 748.2	3526.3 ± 734.5	3467.7 ± 741.1
InvertedPendulum	1000.0 ± 0.0	1000.0 ± 0.0	1000.0 ± 0.0	1000.0 ± 0.0	995.2 ± 9.6	998.7 ± 2.6	998.6 ± 2.8	998.2 ± 3.6
InvertedDoublePendulum	6301.4 ± 299.8	9347.6 ± 0.2	9353.2 ± 0.2	9356.9 ± 0.1	6286.6 ± 308.4	9244.3 ± 64.5	9159.7 ± 225.1	7929.4 ± 1498.2
Ant	1730.3 ± 225.7	2109.3 ± 329.7	2183.2 ± 324.8	2370.0 ± 371.3	1946.4 ± 266.1	2047.1 ± 202.7	2078.5 ± 197.5	2013.3 ± 279.1
Humanoid	747.8 ± 280.7	788.0 ± 355.6	817.8 ± 410.5	895.5 ± 566.3	933.1 ± 432.9	957.3 ± 449.2	919.2 ± 345.9	859.8 ± 272.9

Table 2: branching and backup experiment where $d = 2$ and $\lambda = 0.95$

In this section we explore the impact of using planning as the test policy and which backup scheme captures the highest returns. We consider a PTN trained with a depth equal to 2 and we evaluate both backup procedures found in Section 3.2.

As we can see in Table 2, nearly all environment suffer from an increased branching factor when using Q backup. None of the environments show a clear trend that increasing the branching benefits returns. In fact, both Hopper and Walker2d environments show the opposite, as b increases returns decrease. Intuitively this makes sense, as b increases to infinity the action taken no longer relies on the policy π but instead relies entirely on the Q backup from reward and value predictions.

When the backup scheme takes into account the likelihood of an action coming from the policy, it is no longer true that as b increases to infinity the action selected by decision-time planning is independent from the policy. As we can see in Table 2, every environment benefits from increasing branching used in decision-time planning when the backup scheme is π-Q backup discussed in Section 3.2.2. Comparing Q-backup and π-Q-backup in Table 2 it's obvious to see that the policy weighted Q backup offers much more robust returns.

Now we clearly see that decision-time planning is indeed helpful. Furthermore, we now know that a π-Q backup offers a reliable approach. Going forward we will ask if it's possible to learn from this decision-time planning policy.

4.4 Baseline Comparison

To test our model, we chose to benchmark against PPO2 and PPN on eight MuJoCo (**?**) environments. We include our $d = 2$ $b = 5$ model in our baseline comparison. However, we note that it is possible that other configurations perform better on some environments.

As can be seen in Figure 5, we find that PTN finds a better if not comparable policy in all of the environments. PTN has large performance gains over PPO's model-free approach, even though PTN's implicit planning module only looks a short distance into the future.

Environments with larger meaningful observation spaces, such as Humanoid, we suspect could benefit from a bigger latent space. In Humanoid, the latent space (128) is far smaller than the observation dimensions (376) and makes learning a useful implicit transition model infeasible. Given more compute we would explore increasing the latent space.

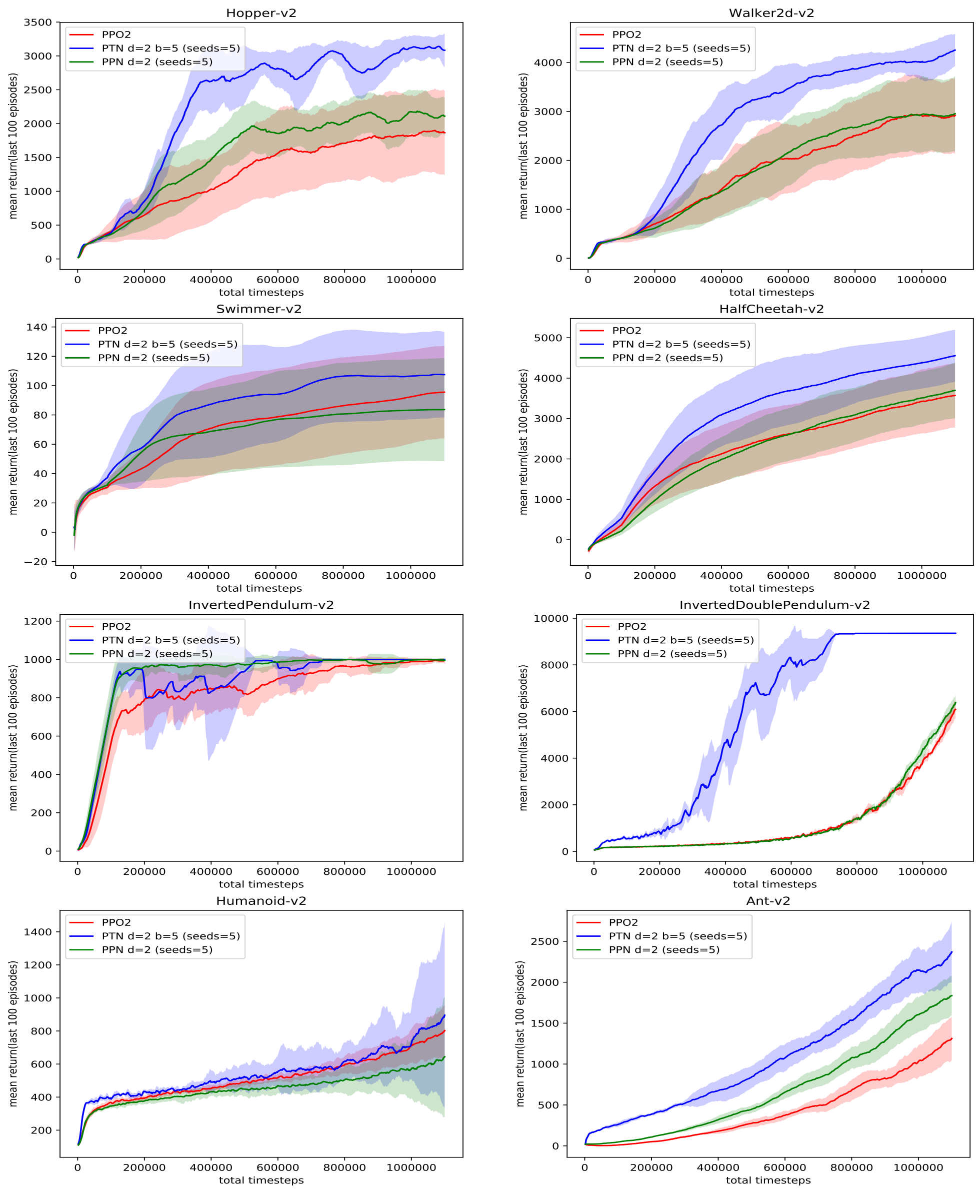

Figure 5: Results on 1 million step MuJoCo benchmark. Dark lines represent the mean return and the shaded region is a standard deviation above and below the mean.

5 Conclusion

In this work we present for the first time an approach to decision-time planning for implicit dynamics models in continuous action space. We provide a theoretical justifications for our design choices and show strong empirical results which are improvements over previous related work. In future work we would like to investigate distilling the tree policy into the model-free policy, find a better sampling procedure for expansion, and incorporate research from recurrent networks to avoid issues with vanishing and exploding gradients in the implicit dynamics model.

REFERENCES

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree planning for deep reinforcement learning. *CoRR*, abs/1710.11417, 2017. URL `http://arxiv.org/abs/1710.11417`.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pp. 2455–2467, 2018.

Elsayed AE Habib. Geometric mean for negative and zero values. *International Journal of Research and Reviews in Applied Sciences*, 11(3):419–32, 2012.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *CoRR*, abs/1707.03497, 2017. URL `http://arxiv.org/abs/1707.03497`.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David P. Reichert, Neil C. Rabinowitz, André Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. *CoRR*, abs/1612.08810, 2016. URL `http://arxiv.org/abs/1612.08810`.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pp. 216–224. Elsevier, 1990.

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Zac Wellmer and James Kwok. Policy prediction network: Model-free behavior policy with model-based learning in continuous action space. In *European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2019.

A Appendix

A.1 Decision-Time Planning as the Behavior Policy

Including branching in Policy Tree Network's behavior policy is appealing because it would allow for directly optimizing the test policy we intend to use and *appears* to be simple since an implicit transition model is already learned. However, as we will soon see this is not the case.

A natural first thought is just to recursively sample π b times up to a depth d, perform a backup on the expanded tree, and then take the action associated with the maximal base branch. This is the approach described in Section 3.2.

However, this can not be used as the behavior policy because you are changing the distribution of how rollout actions are chosen in a way that is difficult to directly measure. The rollout policy $\pi_{\theta',F}$ is not equal to $\pi_{\theta'}$ used to do importance sampling in the policy gradient loss. When $b = 1$ then $\pi_{\theta',F}$ is the same as $\pi_{\theta'}$. However as soon as $b > 1$ we begin to rely on the reward and value network to help decide which actions to take.

The previously mentioned issues can be avoided if $\pi_{\theta',F}$ is not used as the behavior policy. Instead with decision-time planning we can only use $\pi_{\theta',F}$ as our test policy. Interestingly, there is no guarantee that this should work because the policy we are optimizing for (π_θ) is not the same as the test policy. We note that a theoretical issue does arise when using $\pi_{\theta',F}$ as the test policy. The $f^v_{\theta'}$ network parameters are trained to predict the value for policy $\pi_{\theta'}$ not $\pi_{\theta',F}$. However, in practice this turns out not to be a large issue.

A.2 PPN Training Algorithm

Algorithm 2 Policy Prediction Network(PPN) (Wellmer & Kwok, 2019)

```
Initialize parameters $\theta$
$\theta' = \theta$
for iteration=1, 2, ... do
    Run policy $\pi_{\theta'}$ in environment for $n$ time steps
    Compute advantage estimates $A_1^{GAE}, \dots, A_n^{GAE}$
    for epoch= $1, \dots, K$ do
        Shuffle $n$ samples into mini-batches of size $M \leq n$
        for each mini-batch do
            $T$ is the set of samples selected for the mini-batch
            $\mathcal{L}_{mb} = \frac{1}{M} \sum_{t \in T} \mathcal{L}_t$
            Optimize $\mathcal{L}_{mb}$ w.r.t. $\theta$
    $\theta' = \theta$
```

PPO, PPN, and PTN use similar training algorithms, the main difference stems from how $\mathcal{L}_t$ is defined. Where in Algorithm 2, n time steps are used to collect trajectories, K is the number of epochs, M is mini-batch size, and T represents the randomly sampled time steps in a specific mini-batch.

A.3 Clipping

$$\begin{aligned}
\text{ratio}_t^i &= \frac{\pi_\theta(a = a_{t+i} | s = \hat{s}^i_{t,\theta})}{\pi_{\theta'}(a = a_{t+i} | s = \hat{s}^0_{t+i,\theta'})} \\
\text{ratio}^i_{t,\text{clip}} &= \text{clip}(\text{ratio}^i_t, 1-\epsilon, 1+\epsilon), \\
\hat{v}^i_{t,\text{clip}} &= \text{clip}(\hat{v}^i_{t,\theta} - v^0_{t+i,\theta'}, -\epsilon, \epsilon) + \hat{v}^0_{t+i,\theta'}, \\
\hat{r}^i_{t,\text{clip}} &= \text{clip}(\hat{r}^i_{t,\theta} - \hat{r}^0_{t+i,\theta'}, -\epsilon, \epsilon) + \hat{r}^0_{t+i,\theta'}.
\end{aligned}$$

During training we follow the grounded clipping approach shown in PPN (Wellmer & Kwok, 2019). The clipped estimates are defined above. Where ϵ is a hyperparameter that defines the size of the clipping region. Clipping all the network heads turns out to be imperative to the learning process (Wellmer & Kwok, 2019).

A.4 Proofs

Proof of Lemma 1. Note that $cov_U(\pi(X|s), Q(X,s)) = E_U[Q(X,s)\pi(X|s)] - E_U[Q(X,s)]E_U[\pi(X|s)]$. Let R be a bounded action space. By definition, the covariance equals $\frac{1}{area(R)}\int_{x\in R}(Q(x,s)\pi(x|s)) - E_U[Q(X,s)]\frac{1}{area(R)}\int_{x\in R}\pi(x|s)$, where $\int_{x\in R}\pi(x|s) = 1$ and $\int_{x\in R}(Q(x,s)\pi(x|s)) = E_\pi[Q(X,s)]$. The result follows because the covariance is positive.

□

Proof of Theorem 1. We prove this by showing the existence of Q^0 functions for which the inequalities holds. We drop s from our notation for convenience and write $Q(a,s)$ as $Q(a)$ and $\pi(a|s)$ as $\pi(a)$.

Let $X_F \sim \pi_F$, i.e., $X_F = \arg\max_i Q^0(X_i)$, where $X_i \sim \pi$. Consider a $Q^0(a)$ that is strictly decreasing as a increases. Then $\int_{-\infty}^{c}\pi_F(x)dx = Pr(X_F \le c) = Pr(\exists i, X_i \le c)$=$1 - Pr(\forall_i, X_i > c)$. Since X_i's are i.i.d, $Pr(\forall_i, X_i > c) = Pr(X > c)^b = \big(\frac{\text{erfc}(\frac{c-\mu}{\sqrt{2\sigma^2}})}{2}\big)^b$ (setting $X = X_1$). Therefore, $\pi_F(x) = b\big(\frac{\text{erfc}(\frac{c-\mu}{\sqrt{2\sigma^2}})}{2}\big)^{b-1}\pi(c)$.

Then $\mu_F = b\int_{-\infty}^{\infty} c\big(\frac{\text{erfc}(\frac{c-\mu}{\sqrt{2\sigma^2}})}{2}\big)^{b-1}\pi(c)dc$. Substitute $c = \sigma z + \mu$, we have

$$\mu_F = b\sigma\int_{-\infty}^{\infty} z\big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)^{b-1}\phi(z)dz + b\mu\int_{-\infty}^{\infty}\big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)^{b-1}\phi(z)dz$$

where,

$$\begin{aligned} b\mu\int_{-\infty}^{\infty}\big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)^{b-1}\phi(z)dz &= \mu\big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)^{b}\Big|_{-\infty}^{\infty} \\ &= 0 - (-\mu) \end{aligned}$$

and when $b = 2$,

$$\begin{aligned} (2)\sigma\int_{-\infty}^{\infty} z\big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)^{(2)-1}\phi(z)dz &= 2\sigma\Big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)\phi(z) - \frac{\text{erf}(z)}{4\sqrt{\pi}}\Big)\Big|_{-\infty}^{\infty} \\ &= -\frac{\sigma}{\sqrt{\pi}} \end{aligned}$$

Observe that $b\int_{-\infty}^{\infty}\big(\frac{\text{erfc}(\frac{z}{\sqrt{2}})}{2}\big)^{b-1}\phi(z)dz = 1$, which gives the first bound.

Setting $b = 2$, we have that $\mu_F - \mu = -\frac{\sigma}{\sqrt{\pi}} + \mu - \mu = -\frac{\sigma}{\sqrt{\pi}}$.

A similar argument but setting Q^0 to be a strictly increasing function in a gives the inequality for when μ_F is larger than μ.

□

A.5 Numerical evaluation of bound in Theorem 1

Fig. 6 shows the numerical values for the bound in Theorem 1 for several branching values. Note that the bound becomes larger than $1\times\sigma$ when $b = 4$. This means that when $b = 4$, the difference between the mean of π_F and π can become larger than one standard deviation of π, which is a significant difference.

A.6 Learning Modifications

Notice that these returns are worse than PTN(with decision-time planning) shown in Figure 5. This shows performance benefits from the modifications shown in Section 3.1(ex: β). The point of

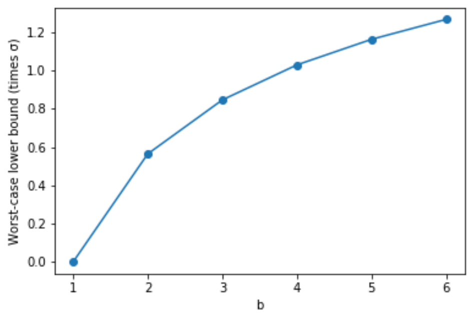

Figure 6: Numerical evaluation of bound in Theorem 1 for various branching values

	train depth = 5		train depth = 10	
	PTN	PPN	PTN	PPN
Hopper-v2	1944.0 $\pm$ 258.5	**2191.6** $\pm$ **183.3**	1672.7 $\pm$ 457.5	**1752.3** $\pm$ **625.4**
Walker2d-v2	**2936.9** $\pm$ **893.2**	2808.4 $\pm$ 647.3	**3054.8** $\pm$ **434.0**	2565.9 $\pm$ 327.9
Swimmer-v2	**89.4** $\pm$ **35.3**	73.3 $\pm$ 34.5	**80.3** $\pm$ **35.7**	57.5 $\pm$ 30.3
HalfCheetah-v2	**3439.8** $\pm$ **698.6**	3410.5 $\pm$ 840.9	**3638.9** $\pm$ **545.2**	3632.8 $\pm$ 819.0
InvertedPendulum-v2	971.7 $\pm$ 56.5	**979.6** $\pm$ **25.7**	1000.0 $\pm$ 0.0	1000.0 $\pm$ 0.0
InvertedDoublePendulum-v2	**4332.6** $\pm$ **192.9**	3344.5 $\pm$ 234.4	**4360.2** $\pm$ **145.1**	3186.8 $\pm$ 87.7

Table 3: Returns from PTN(no π_F/decision-time planning) and PPN using only the model-free policies

β(found in PTN but not PPN) is to stabilize returns over different values of training depth. In PPN, Wellmer & Kwok (2019) showed that optimal depth is highly dependent on the environment and returns can drastically differ. While the modifications certainly do not entirely fix this, we find that it mitigates large differences in returns from different training depths.