Policy Prediction Network: Model-Free Behavior Policy with Model-Based Learning in Continuous Action Space

Zac Wellmer[1] (✉) and James T. Kwok[1]

Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
zac@1984.ai

Abstract. This paper proposes a novel deep reinforcement learning architecture that was inspired by previous tree structured architectures which were only useable in discrete action spaces. Policy Prediction Network offers a way to improve sample complexity and performance on continuous control problems in exchange for extra computation at training time but at no cost in computation at rollout time. Our approach integrates a mix between model-free and model-based reinforcement learning. Policy Prediction Network is the first to introduce implicit model-based learning to Policy Gradient algorithms for continuous action space and is made possible via the empirically justified clipping scheme. Our experiments are focused on the MuJoCo environments so that they can be compared with similar work done in this area.

1 Introduction

Reinforcement learning algorithms can be model-free or model-based. Model-free reinforcement learning attempts to find a policy through interacting with the environment and improving the policy based on previous states and rewards. Model-based reinforcement learning attempts to learn the dynamics of the environment and uses the model to improve the policy through various methods such as planning, exploration, and even training on generated data [5, 19]. Though historically, model-based methods capable of predicting near perfect observations [2, 10] usually have the benefit of reduced sample complexity, they still struggle to perform as well as model-free methods [8, 9, 17]. It is therefore appealing to explore achieving the best of both worlds, as collecting the large amount of experience required by model-free methods is oftentimes expensive or infeasible.

Model-based agents traditionally learn a model of the environment that predicts future observations conditioned on previous actions and observations. This approach is sometimes referred to as observation-prediction models [11]. Recreating the original observation is sometimes a questionable objective as the original observation can be dominated by irrelevant information. For example, if the observation is an image and contains a complex background, a large part of the model's capacity can be spent on modeling the background even though it may be irrelevant to information necessary for planning.

As a response to the issues faced by observation-prediction models, several implicit model-based methods [4, 11, 18] were introduced and learn an implicit transition module that predicts the value/reward of future states without being subjected to observation reconstruction. Value Prediction Networks (VPN) [11] and TreeQN [4] operate by expanding a tree of predicted reward and value estimates. However, this is feasible only because each branch is linked to a discrete action. ATreeC [4] introduces a policy gradient method but is still not applicable to continuous action spaces because their policy is a multinomial distribution parameterized by the Q-values associated with each branch. Implicit models have seen success in Q-learning approaches, but are not straightforward to apply to policy gradient methods. Many real-world problems, such as robotics or autonomous vehicle applications, lie in continuous action spaces and Q-learning approaches do not naturally extend to continuous action spaces like policy gradient methods.

Policy gradient methods are of primary interest in this paper because of their inherent flexibility in terms of their application to both discrete and continuous action spaces. In particular, we will focus on a model-free policy gradient algorithm called Proximal Policy Optimization (PPO) [17]. PPO is of high interest because of its high performance on popular benchmarks and simplicity.

We propose Policy Prediction Networks (PPN), where the value, reward, policy, and abstract-state are predicted by leveraging a transition model. PPN uses an implicit model-based approach at training time but a model-free approach at rollout time. An implicit model-based approach at training time helps accelerate feature learning via predicting future policies, values, and rewards. All of which encourage the dynamics model to learn features that are well aligned with our objective of finding a policy that maximizes returns. To the best of our knowledge, this is the first work on developing implicit model-based learning for policy gradient methods.

Our contribution is a training procedure that leverages model-based learning for policy gradient algorithms to improve performance and does not trade off computational costs at rollout time. This work introduces implicit transition models for Policy Gradient methods, depth-based objectives, auxiliary reward objectives, and an empirically justified clipping scheme. Furthermore, our work lays down the foundation for future research on using implicit transition models to perform decision-time planning. Empirical results demonstrate the advantage of PPN over the model-free baseline (PPO), which suggests that PPN finds a better state embedding and reduces sample complexity.

2 Background and Related Work

In this section, we will give a brief review of related work on model-based and model-free reinforcement learning. We also introduce terminologies to differentiate between two general approaches in model-based reinforcement learning.

Notations: s_t abstract state, a_t action, r_{t+1} reward from taking a_t at s_t, γ discount, $v_\theta(s_t)$ value of state s_t with respect to parameters θ, A_t advantage, H_t policy entropy, and the subscript t denotes timesteps.

2.1 Policy Gradient Methods

Policy Gradient Methods [21] are a type of reinforcement learning algorithm that directly optimizes policy parameters to maximize expected returns. Policy Gradient methods are more naturally applied to environments with continuous action spaces in comparison to Q-learning approaches. Generally, the policy gradient loss is of the shape:

$$\mathcal{L}_t^\pi = -\log \pi_\theta(a_t|s_t)A_t - H_t,$$

where A_t is an advantage estimate [16, 20, 21], π_θ is the policy with parameter θ, H_t is the policy entropy, s_t is the state at time t, and a_t is the action taken in state s_t.

2.2 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) attempts to generate monotonically improving policies following inspiration from conservative policy iteration [7]. However, TRPO contains a few theoretical relaxations that are required to make a practical algorithm. TRPO is left with a hard KL divergence constraint to be less than or equal to δ. This hard constraint can be seen as a trust region on the mean KL divergence.

Let θ' be the old parameters, θ be the new proposed parameters, A_t^{GAE} be the generalized advantage estimate [16]. Which is defined as

$$A_t^{GAE} = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{n-t+1}\delta_{n-1}, \tag{1}$$

where λ is a hyperparameter controlling the bias-variance trade-off, and $\delta_t = r_{t+1} + \gamma v_{\theta'}(s_{t+1}) - v_{\theta'}(s_t)$.TRPO's optimization problem is then formulated as:

$$\pi_\theta = \max_{\pi_\theta} L_{\pi_{\theta'}}(\pi_\theta) \quad : \quad \bar{D}_{KL}(\pi_{\theta'}, \pi_\theta) \leq \delta,$$

where $L_{\pi_{\theta'}}(\pi_\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}A^{GAE}$ is the objective using importance sampling to estimate expected advantage under the new policy, and $\bar{D}_{KL}(\pi_{\theta'}, \pi_\theta)$ is the mean KL divergence between the new and old policy. At this point TRPO offers theoretical inspiration but does not actually offer theoretical guarantees for monotonically improving policies.

2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [17] was introduced as offering similar benefits as TRPO [15], but via a simpler approach. PPO replaces a KL divergence constraint in TRPO via a clipped policy gradient loss:

$$\mathcal{L}_t^\pi = \max(-\text{ratio}_t \cdot A_t^{GAE}, -\text{ratio}_{t,\text{clip}} \cdot A_t^{GAE}), \tag{2}$$

where

$$\text{ratio}_t = \frac{\pi_\theta(a = a_t|s = s_t)}{\pi_{\theta'}(a = a_t|s = s_t)},$$
$$\text{ratio}_{t,\text{clip}} = \text{clip}\left(\frac{\pi_\theta(a = a_t|s = s_t)}{\pi_{\theta'}(a = a_t|s = s_t)}, 1-\epsilon, 1+\epsilon\right). \tag{3}$$

Clipping no longer guarantees $\bar{D}_{KL}(\pi_{\theta'}, \pi_\theta) \leq \delta$. Instead, it serves to approximate it (see [6] for further details).

PPO2 [3] is a GPU implementation from OpenAI that offers a key difference from PPO, namely, that the critic is also clipped. More specifically, the critic loss ($\mathcal{L}_t^v$) is now:

$$\mathcal{L}_t^v = \max((v_\theta(s_t) - R_t)^2, (v_{t,\text{clip}} - R_t)^2) \tag{4}$$

Where $v_{t,\text{clip}} = \text{clip}(v_\theta(s_t) - v_{\theta'}(s_t), -\epsilon, \epsilon) + v_{\theta'}(s_t)$ is the clipped value estimate, $R_t = \gamma^n v_{\theta'}(s_{t+n}) + \sum_{i=1}^{n} \gamma^{i-1} r_{t+i}$ is the bootstrapped n-step return at time t, and n is the number of steps in the bootstrapped estimate. At this point theoretical guarantees in Conservative Policy Iteration have been dropped to make TRPO a practical algorithm, and the theoretical justifications in TRPO have again been weakened to make the more versatile and empirically superior PPO.

2.4 Model-based Reinforcement Learning

The essence of model-based reinforcement learning revolves around using a model or learning a model of the environment, and using this to improve a policy. We will be focused on the challenging class of problems where the environment dynamics are unknown and must be learned. In this case, the problem can be broken down further into dynamics models that are learned implicitly and dynamics models that are learned explicitly. It was not until recent years that implicitly learned model-based algorithms received attention [4, 11, 18].

Explicit Model-based Methods Cases of explicit model-based methods involve some form of directly predicting future observations and including this in the loss function, as in:

$$L_t^{model} = \frac{1}{2}\|\hat{x}_t - x_t\|^2,$$

where $\hat{x}_t$ is the predicted observation at time t and x_t is the ground truth observation. Several variations exist that involve learning to predict in an abstract state space [2, 5, 13, 14], and predicting the grounded observation over multiple time steps [10]. This has seen some success and can be useful for learning, planning, and exploration.

These methods are particularly useful when observations contain entirely useful information. However, it can be misleading in the class of problems where

parts of the observation do not include useful information. For example, when generating an image frame, a large part of the network's capacity could be dedicated to learning less useful information like the background or objects that are not well aligned with the agent's interest [13].

Implicit Model-based Methods Implicit model-based methods are interesting because they are not explicitly tied to reproducing original observations or an encoded observation. Rather, the dynamics model is indirectly learned by finding parameters that allow for an agent to perform optimally. This is done by learning to predict future characteristics such as the value or reward [4, 11], but without having a constraint on predicting the ground truth observation. Unfortunately, a downside to implicit approaches is that it is difficult to know what is actually taking place during planning since it is hard to reconstruct the predicted observations.

VPN [11] involve expanding a Q-tree, performing a linear backup along the maximal path, and selecting the maximal backed-up path. At training time, the loss is computed along the tree path followed by rollout actions. The Predictron [18] is similar to VPN except learning is done entirely in an abstract space, whereas VPN is grounded to transitions experienced by the rollout policy. Predictron also offers a meta-objective called consistency which lines up individual estimates with respect to backed-up estimates. We do not explore this in our work, but note that it could serve as an orthogonal improvement. TreeQN and ATreeC [4] introduce a Q learning approach (TreeQN) and a policy gradient approach (ATreeC) that make use of a differentiable tree structure. ATreeC involves expanding a pseudo Q-tree. This is pseudo because the nested value predictions are not directly constrained to represent the value. The backed-up pseudo Q-values are treated as logits and are used to parameterize and sample from a multinomial distribution. These samples are then used as the actions. VPN, ATreeC, and TreeQN are limited to only operating in discrete action spaces. Predictron was used in a continuous action space, the MuJoCo pool environment [22], but was done through discretizing the action space.

3 Policy Prediction Network

Policy Prediction Network uses a combination of model-free and model-based techniques. Actions are made with a model-free approach by the behavior policy at rollout time. However, learning is done with a model-based approach that follows the rollout trajectory. A latent space transition model is embedded into the architecture so that we are able to backpropagate from multiple simulation steps into the future back to a grounded observation. Backpropagation from predictions through the dynamics model, and back to a grounded observation enables the dynamics model to learn features which align with accurate reward predictions, accurate value predictions, and maximizing advantage. This is as opposed to maximizing observation reconstruction as is traditionally done in explicit model-based reinforcement learning.

Our novel contribution is a training scheme that integrates model-free and model-based reinforcement learning to improve sample complexity and performance in exchange for extra computation at training time but at no extra cost in computation at rollout time. Additionally our work offers a foundation for decision-time planning for policy gradient methods and implicit transition models. Our empirical results in Section 4 demonstrate the advantage of PPN over model-free baseline (PPO), which suggests that PPN finds a better state embedding and reduces sample complexity.

3.1 Architecture

PPN is comprised of a few components. The components are parameterized by $\theta = \{\theta^{enc}, \theta^{v}, \theta^{r}, \theta^{tr}, \theta^{\pi}\}$ described below. In the following, a hat over variables represents that it is an estimate as opposed to a grounded observation or reward. The superscript represents the forward step predictions. The depth-rollout is expanded to a depth d. For example, $\hat{s}_t^i$ is the predicted state i steps (where $0 \leq i \leq d$) forward in time from t.

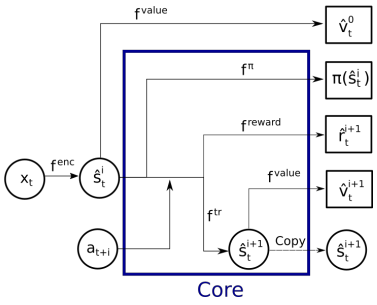


Fig. 1: PPN learns to predict policies, rewards, abstract states, and the value of the abstract states.

Encoding $(f_\theta^{enc}(x_t) = \hat{s}_{t,\theta}^0)$ function embeds the observation (x_t) in an abstract state $(\hat{s}_{t,\theta}^0 \in \mathbb{R}^y)$.

Value $(f_\theta^v(\hat{s}_{t,\theta}^i) = \hat{v}_{t,\theta}^i)$ function estimates the value $(\hat{v}_{t,\theta}^i \in \mathbb{R})$ of the abstract state.

Policy $(\mathcal{N}(f_\theta^\mu(\hat{s}_{t,\theta}^i), f^\Sigma(T)) = \pi_\theta(\hat{s}_{t,\theta}^i))$ function parameterizes a distribution over actions to take given a state $\hat{s}_{t,\theta}^i$. The policy module has two parts. The first (f_θ^μ) producing estimates of the mean $(\mu \in \mathbb{R}^z$ where z is dimensionality of action space) and the second (f^Σ) producing estimates of a diagonal covariance matrix $(\Sigma \in \mathbb{R}^{z \times z})$ to parameterize a normal distribution for the policy (π). This is further described in Section 3.2.

Reward $(f_\theta^r(\hat{s}_{t,\theta}^i, a_{t+i}) = \hat{r}_{t,\theta}^{i+1})$ function predicts the reward $(\hat{r}_{t,\theta}^{i+1} \in \mathbb{R})$ for executing the action a_{t+i} at abstract state $\hat{s}_{t,\theta}^i$.

Transition $(f_\theta^{tr}(\hat{s}_{t,\theta}^i, a_{t+i}) = \hat{s}_{t,\theta}^{i+1})$ function transforms the abstract state given an action to the next abstract state $(\hat{s}_{t,\theta}^{i+1} \in \mathbb{R}^y)$ by predicting $\Delta = \hat{s}_{t,\theta}^{i+1} - \hat{s}_{t,\theta}^i$.

We adopt a similar convention from VPN [11] which defines a core module. Figure 1 shows the core module which performs a depth-1 rollout by composing the modules:

$$
\begin{aligned}
f_\theta^{enc}(x_t) &= \hat{s}_{t,\theta}^0, \\
f_\theta^{v}(\hat{s}_{t,\theta}^0) &= \hat{v}_{t,\theta}^0, \\
f_\theta^{core}(\hat{s}_{t,\theta}^i, a_{t+i}) &= (\pi(\hat{s}_{t,\theta}^i), \hat{r}_{t,\theta}^{i+1}, \hat{v}_{t,\theta}^{i+1}, \hat{s}_{t,\theta}^{i+1}).
\end{aligned}
$$

There are 4 subtle but important differences between PPN depth rollouts and Value Prediction Network depth rollouts.

1. PPN estimates the policy based on the abstract state ($\pi(\hat{s}^i_{t,\theta})$) at each step of the core module; while in VPN there is no need to predict a policy π because it is a Q-learning method. However, this means it does not naturally apply to continuous action-spaces.
2. PPN produces a value estimate ($\hat{v}^0_{t,\theta}$) at the base of the depth rollout and uses this as the critic. In VPN, this is not necessary because it is not an actor-critic method.
3. The actions used in PPN come from samples generated by the behavior policy ($\pi_{\theta'}$), seen later in Equation (5); while in VPN, the actions are chosen by exhaustively simulating all possible actions. Simulating all possible actions is only feasible in a discrete action space.
4. PPN only uses the depth-based rollout at training time. VPN's behavior policy can use decision-time planning [20]. However, this is not straightforward to apply to continuous action spaces and we leave this for future work.

If $d > 1$, the PPN recursively calls the core function (f^{core}_θ) to generate a trajectory of simulated rewards, policies, values, and abstract states conditioned on an initial abstract state ($\hat{s}^0_{t,\theta}$) and action trajectory ($a_t, \ldots, a_{t+d-1}$). Each recursive call passes on the predicted abstract state ($\hat{s} = \hat{s}'$).

3.2 Planning

Here we introduce our approach to background planning [20] in continuous action spaces performed at training time. PPN has the ability to predict the future abstract states and based on these predicted future abstract states make additional predictions of future rewards, values, and policies. We use a basic planning method which simulates up to a certain depth d collecting reward, value, and policy estimates along the way.

Background planning is done by following the actions performed by the behavior policy and recursively calling f^{core} with the predicted abstract state. Action generation by the rollout policy ($\pi_{\theta'}$) is done by sampling from a normal distribution defined as follows:

$$a_t \sim N(\mu = f^{\mu}_{\theta'}(\hat{s}^0_{t,\theta'}),\ \Sigma = f^{\Sigma}(T)|\hat{s}^0_{t,\theta'}, T). \tag{5}$$

$f^{\Sigma}(T)$ is a function of the number of samples T seen since the beginning of training, and does not depend on the model parameters. In our experiments the standard deviation used to parameterize a diagonal covariance matrix is exponentially decayed with respect to the number of samples seen, as is done in PPO [17].

3.3 Learning

PPN is trained in a similar manner to typical Policy Gradient algorithms. The novel differences we introduce are depth-based losses, a latent transition model

Algorithm 1 Policy Prediction Network(PPN), PPO style.

Initialize parameters θ
$\theta' = \theta$
for iteration=1, 2, ... **do**
  Run policy $\pi_{\theta'}$ in environment for n time steps
  Compute advantage estimates $A_1^{GAE}, \ldots, A_n^{GAE}$
  for epoch= 1, ..., 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. θ
    end for
  end for
  $\theta' = \theta$
end for

(f^{tr}) embedded into the architecture, auxiliary reward objectives, and a clipping scheme for depth-based losses. Depth-based losses are necessary to train the implicit transition model. The implicit transition model and auxiliary reward help with feature learning via background-planning.

PPN seeks to optimize auxiliary objectives and perform multiple updates on a batch. Trust regions as seen in TRPO can not be directly applied to both cases described above [17] and thus we introduce a clipping approach. Clipping all the network heads is crucial because the parameters of the reward network and value network all share parameters ($f^{\text{tr}}, f^{\text{enc}}$) with the policy network. For a visual reference of parameter sharing please see Figure 1. This means that if any of the networks are updated in an uncontrolled fashion, it can also cause dramatic changes to the policy.

In addition, in Algorithm 1, we show that PPN performs a similar learning algorithm as was done in PPO.

The major differences in training between PPN and PPO come from the loss formulation ($\mathcal{L}$). The behavior policy with parameters (θ') generates an n-step trajectory $(x_1, a_1, x_2, a_2, r_2, \ldots, x_{n+1}, r_{n+1})$. The depth-$i$ predictions are grounded based on the generated n-step action trajectories. The loss at time t accumulates error over the planned trajectory up to a depth d and is defined as:

$$\mathcal{L}_t = \mathcal{L}_t^{\pi} + \alpha_v \mathcal{L}_t^{v} + \alpha_r \mathcal{L}_t^{r}, \tag{6}$$

where minimizing $\mathcal{L}_t^{\pi}$ corresponds to maximizing expected advantage, $\mathcal{L}_t^{v}$ results in an accurate critic, and $\mathcal{L}_t^{r}$ leads to reward predictions that represent the environment's actual reward for a state action pair. α_v, α_r are the penalty coefficients for the value loss and reward loss respectively.

Specifically, we define

$$\mathcal{L}_t^{\pi} = \frac{1}{d_\pi} \sum_{i=0}^{d_\pi - 1} \max(-\text{ratio}_t^i A_{t+i}^{\text{GAE}}, -\text{ratio}_{t,\text{clip}}^i A_{t+i}^{\text{GAE}}) - \alpha_h H, \tag{7}$$

where $\text{ratio}_t^i = \frac{\pi_\theta(a=a_{t+i}|s=\hat{s}_{t,\theta}^i)}{\pi_{\theta'}(a=a_{t+i}|s=\hat{s}_{t+i,\theta'}^0)}$ is the importance sampling ratio between the new policy and the old policy at depth i, $\text{ratio}_{t,\text{clip}}^i$ is the clipped ratio used to ensure the new parameter's estimate to be near the old parameter's estimate. We offer two possible formulations to clipping in Section 3.4. A_t^{GAE} is the generalized advantage estimate defined in (1), and α_h is a hyperparameter for the entropy coefficient.

As for the critic objective, we have the critic loss

$$\mathcal{L}_t^v = \frac{1}{d_v+1}\sum_{i=0}^{d_v}\frac{1}{2}\max((\hat{v}_{t,\theta}^i - R_{t+i})^2, (\hat{v}_{t,\text{clip}}^i - R_{t+i})^2), \tag{8}$$

which encourages the current value estimate $\hat{v}_{t,\theta}^i$ to be close to the bootstrapped return R_{t+i} at each depth i without moving closer to the target than the clipped estimate ($\hat{v}_{t,\text{clip}}^i$). The clipped estimate is guaranteed to be near the old parameter's estimate. Notice that $\mathcal{L}_t^v$ is over an extra iteration of the summation. This is because value estimates are made at every state $(\hat{s}_{t,\theta}^0, \ldots, \hat{s}_{t,\theta}^d)$ in the forward plan.

Similarly, the reward loss is

$$\mathcal{L}_t^r = \frac{1}{d_r}\sum_{i=0}^{d_r-1}\frac{1}{2}\max((\hat{r}_{t,\theta}^i - r_{t+i})^2, (\hat{r}_{t,\text{clip}}^i - r_{t+i})^2), \tag{9}$$

encourages the reward estimate $\hat{r}_{t,\theta}^i$ to be close to the reward r_{t+i} at each depth i without moving closer to the target than the clipped estimate ($\hat{r}_{t,\text{clip}}^i$).

The maximum in Equations (7)-(9) is taken between the unclipped surrogate objective and clipped surrogate objective. In the case of the critic and reward losses ($\mathcal{L}_t^v$ and $\mathcal{L}_t^r$), this means that updates only take place when the estimate from the new parameters (θ) are farther from the target (R_{t+i} in Equation 8 and r_{t+i} in Equation 9) than the clipped estimate. When the new parameter's estimate is closer, the max in Equation 8 and 9 will select the clipped surrogate. The gradient of the clipped surrogate with respect to parameters (θ) will be zero, and thus will not change any parameters. This is desirable because it attempts to prevent destructive updates that push estimates made by θ far from estimates made by θ'.

Remark 1. PPN can be reduced to PPO2 if ($\alpha_r = 0$), $d_\pi = 1$, and $d_v = 0$.

It's possible to use different values of depth for the objectives but unless otherwise noted $d = d_\pi = d_v = d_r$.

3.4 Clipping

We present two approaches to clipping called grounded and ungrounded clipping. In this case, grounded and ungrounded refer to whether we have access to the ground truth observation (x_t). Grounded clipping offers a less strict clipping

region, while ungrounded clipping is more aligned with theoretical justifications found in Conservative Policy Iteration [7], TRPO [15], and PPO [17]. Our clipped objectives are advantageous for two reasons. First, they allow for auxiliary reward and depth based updates. Second, they allow us to share parameters between the transition, policy, value, reward, and embedding networks. Both of these are essential to learn the implicit transition model and are helpful with feature learning.

Grounded Clipping The clipping region is grounded with respect to both the action trajectory $(a_t, \ldots, a_{t+d})$ and the latent state $(\hat{s}^0_{t,\theta'}, \ldots, \hat{s}^0_{t+d+1,\theta'})$. The three grounded clipped estimates are shown in Equations (10), (11), and (12):

$$\text{ratio}^i_{t,\text{clip}} = \text{clip}(\text{ratio}^i_t, 1-\epsilon, 1+\epsilon), \tag{10}$$

$$\hat{v}^i_{t,\text{clip}} = \text{clip}(\hat{v}^i_{t,\theta} - v^0_{t+i,\theta'}, -\epsilon, \epsilon) + \hat{v}^0_{t+i,\theta'}, \tag{11}$$

$$\hat{r}^i_{t,\text{clip}} = \text{clip}(\hat{r}^i_{t,\theta} - \hat{r}^0_{t+i,\theta'}, -\epsilon, \epsilon) + \hat{r}^0_{t+i,\theta'}. \tag{12}$$

The clipping region is based on the grounded estimates from the old parameters (θ') rather than predicted estimates from old parameters.

Ungrounded Clipping The clipping region in this case is grounded with respect to only the action trajectory, but ungrounded with respect to the latent state. The ungrounded clipping estimates are defined as:

$$\text{ratio}^i_{t,\text{clip}} = \text{clip}\left(\frac{\pi_\theta(a = a_{t+i}|s = \hat{s}^i_{t\theta})}{\pi_{\theta'}(a = a_{t+i}|s = \hat{s}^i_{t,\theta'})}, 1-\epsilon, 1+\epsilon\right)\frac{\pi_{\theta'}(a = a_{t+i}|s = \hat{s}^i_{t,\theta'})}{\pi_{\theta'}(a = a_{t+i}|s = \hat{s}^0_{t+i\theta'})},$$

$$\hat{v}^i_{t,\text{clip}} = \text{clip}(\hat{v}^i_{t,\theta} - v^i_{t,\theta'}, -\epsilon, \epsilon) + \hat{v}^i_{t,\theta'}, \tag{13}$$

$$\hat{r}^i_{t,\text{clip}} = \text{clip}(\hat{r}^i_{t,\theta} - \hat{r}^i_{t,\theta'}, -\epsilon, \epsilon) + \hat{r}^i_{t,\theta'}, \tag{14}$$

and $\text{ratio}^i_t = \frac{\pi_\theta(a=a_{t+i}|s=\hat{s}^i_{t,\theta})}{\pi_{\theta'}(a=a_{t+i}|s=\hat{s}^0_{t+i,\theta'})}$. Notice the change in how $\text{ratio}^i_{t,\text{clip}}$ is defined. We first clip the ratio between new and old ungrounded policies to be no more or less $1 \pm \epsilon$ and then perform importance sampling to account for the advantage being calculated with respect to the rollout policy.

4 Experiments

Our experiments seek to answer the following questions: (1) Is clipping necessary? If so, which type of clipping performs best(Section 4.2)? (2) Does PPN outperform model-free baselines(Section 4.3)? (3) What effect does depth have on performance(Section 4.4)? (4) Is the implicit transition module actually predicting abstract states that are useful to the policy(Section 4.5)?

	observation dimensions	action dimensions
Hopper-v2	11	3
Walker2d-v2	17	6
Swimmer-v2	8	2
HalfCheetah-v2	17	6
InvertedPendulum-v2	4	1
InvertedDoublePendulum-v2	11	1
Humanoid-v2	376	17
Ant-v2	111	8

Table 1: Summary of the MuJoCo environments used.

4.1 Experimental Setup

Our experiments investigate the comparison of PPN to PPO2 [3] on the OpenAI Gym MuJoCo environments [1, 22]. Preprocessing was done similarly to that of PPO [17]. Both PPO2 and PPN were implemented in Pytorch [12].

The comparison against PPO2 is run over all 8 environments listed in Table 1. Due to returns being subject to high variance, we run tests over 15 seeds (which is 3-5 times more than the related works in [11, 17]). [1] Due to computational constraints in other experiments we run over 5 seeds on the Walker2d-v2 and Ant-v2 environments.

Our PPO2 implementation uses the same hyperparameters as the baselines implementation [3]. The largest difference in our PPO2 implementation is that we do not perform orthogonal initialization. We did not include orthogonal initialization because it was not mentioned in the original PPO and we did not notice any clear performance benefits. For example, our implementation receives roughly double the returns on the HalfCheetah-v2 environment than the results reported by the baselines [3] implementation of PPO2 using orthogonal initialization.

Our PPN implementation uses similar hyperparameters: 2 fully connected layers for the embedding. 2 fully connected residual layers with unit length projections of the abstract-state [4] for the transition module. 1 fully connected layer for the policy mean, value and reward. All hidden layers have 128 hidden units and tanh activations. In practice we use Huber losses instead of L2 losses, as was done in related implicit model based works [11].

4.2 Clipping

We first look into the effect grounded clipping the network heads has on returns gathered by PPN agents. Here we test to see if a strong policy can still be learned if the other network heads (f^r, f^v) are not clipped.

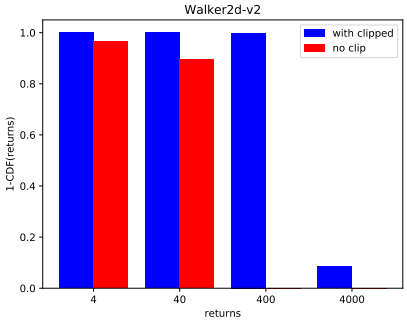
[1] Due to computational constraints InvertedDoublePendulum-v2 only uses 5 seeds

	Grounded	Ungrounded
Hopper-v2	2172.28	1356.14
Walker2d-v2	2937.20	1717.23
Swimmer-v2	83.22	85.27
HalfCheetah-v2	3509.34	3485.59
InvertedPendulum-v2	996.44	998.47
InvertedDoublePendulum-v2	4336.93	4071.19
Humanoid-v2	574.15	676.31
Ant-v2	1602.15	1566.06

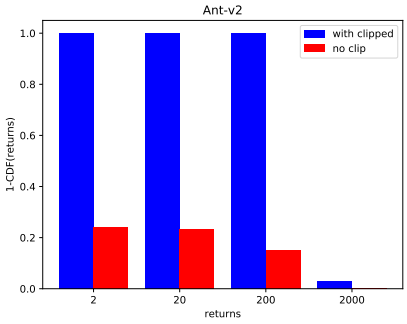
Table 2: Returns using grounded and ungrounded clipping

As similarly done by Ilyas *et al.* [6], we fit a normal distribution to the returns achieved by the random seeds. Then we compare points on the cumulative distribution functions (CDF) that correspond to returns of $2, 20, 200, 2000$ for Ant-v2 and $4, 40, 400, 4000$ for Walker2d-v2.

In Figure 2a and 2b we can see that clipping all the network heads turns out to be imperative to learn a useful policy. As stated in Section 3.3 clipping all the network heads is imperative because they all share parameters $(f^{\text{tr}}, f^{\text{enc}})$ with the policy. Additionally, we look into which type of clipping performs best. For the most part Table 2 shows grounded clipping offers the most robust returns. For all other PPN experiments we use the grounded clipping scheme.



(a) Walker2d-v2



(b) Ant-v2

Fig. 2: Comparison of returns with and without (grounded) clipping of reward and critic.

4.3 Baseline Comparison

To test our model, we chose to benchmark against PPO2 and use the environments in Table 1. As is done in related works [4], we include depth $d = 1$ and $d = 2$ in our baseline comparison. However, we note that it is possible that larger depth values could be better on other environments.

As can be seen in Figure 3, we find that PPN finds a better if not comparable policy in all of the environments. We notice that PPN tends to do well in complex environments such as Ant-v2. Indeed Humanoid-v2 is an exception to this observation. Perhaps this is because Humanoid-v2's number of observation

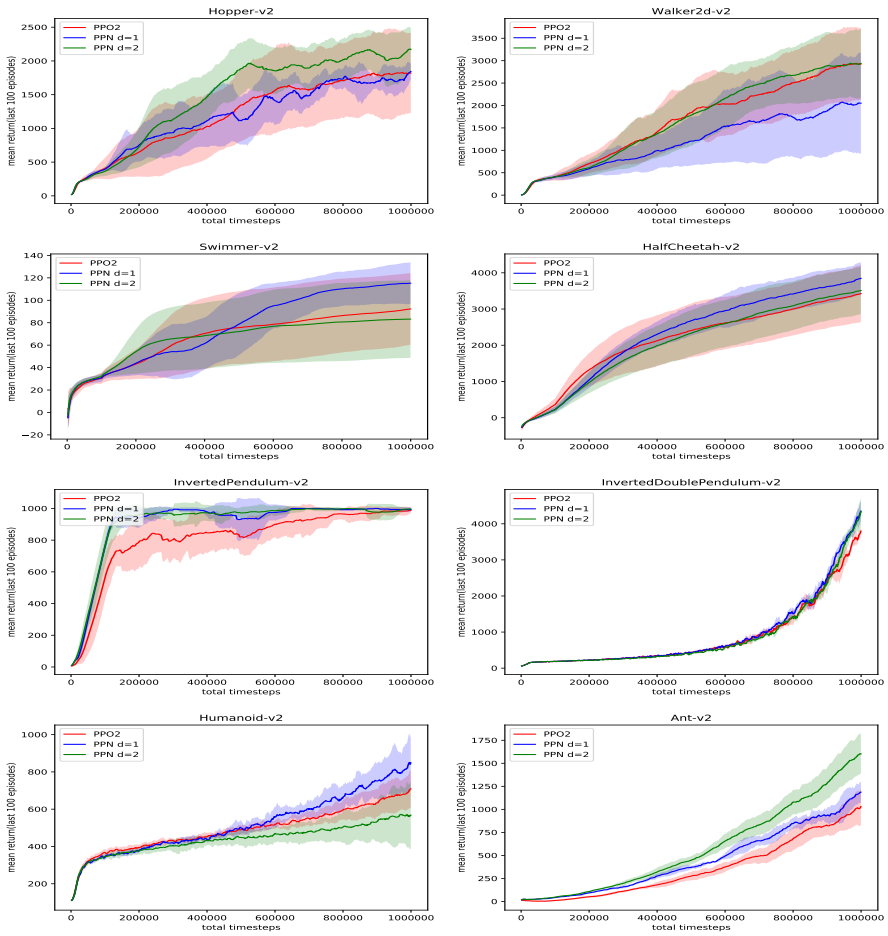


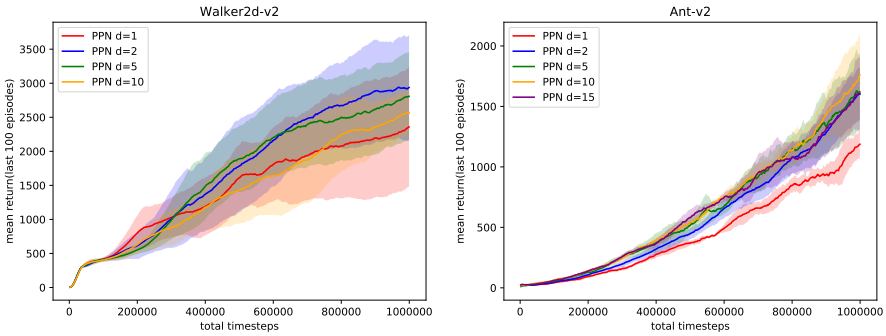
Fig. 3: Results on 1 million step MuJoCo benchmark. Dark lines represent the mean return and the shaded region is a standard deviation above and below the mean.

dimensions(376) are far larger than the latent space(128). Additionally, we notice optimal depth is environment dependent and is further studied in Section 4.4.

4.4 Depth

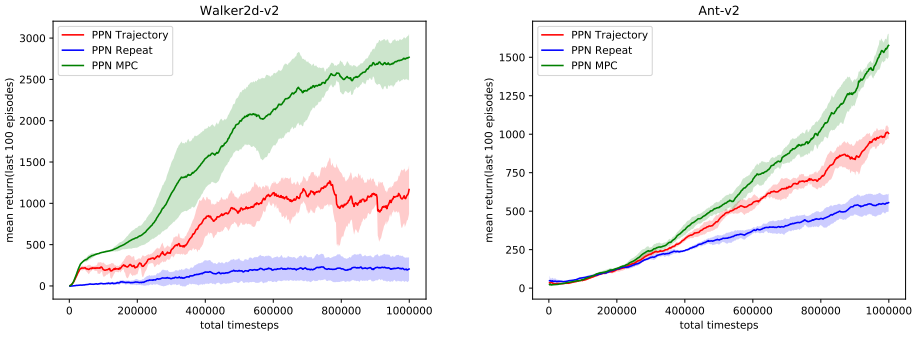
In this section, we explore the effect depth has on returns gathered by PPN agents. Increasing depth forces the agent to learn abstract state which contains information relevant to longer-term environment dynamics.

As seen in Figure 4 increasing depth (d) offers performance improvements but only up to a certain point. As the depth grows, we become more reliant on having a good transition function and eventually leads to a worse policy.



(a) Walker2d-v2 (b) Ant-v2

Fig. 4: Returns with respect to d values of 1, 2, 5, and 10.



(a) Walker2d-v2 (b) Ant-v2

Fig. 5: Returns with respect to three difference action selection approaches.

In Walker2d-v2 (Figure 4a) we can clearly see a depth of 2 offers performance gains over a depth of 1. However after this point returns decrease as we increase depth. We suspect that optimal depth for Walker2d-v2 may be less than Ant-v2 because the implicit transition module is less accurate. A similar conclusion can be drawn from our observation in Section 4.5. Optimal depth is a recurring issue in implicit model-based approaches [4, 11].

4.5 Transition Ablation

We are curious whether the implicit transition module is actually predicting abstract states that resemble reality closely enough to actually be useful to the policy. To test this we perform an ablation study of 3 different types of policy prediction networks. The first type Model Predictive Control (MPC) represents a perfect implicit transition module as the policy in this case has access to the ground truth observation. The standard MPC approach is where only the first

action is followed and the rest are replanned. The second type, "trajectory", represents the strength of the transition module. In this case, every d steps a new trajectory is generated by recursively calling f^{core} with the predicted abstract states and the sampled actions the predicted policy. The third type, "repeat", represents a meaningless transition module. In this case, every d steps a new action is generated by the policy and repeated for d steps. If the implicit transition module is bad we expect the returns from trajectory and repeat to be more or less the same. If the implicit transition module is good we expect returns somewhere in between the MPC and repeat curves. Note that all 3 of these approaches are trained in the same manner and have exactly the same parameters.

In Figures 5a and 5b we see that the trajectory approach performs much better than repeat but not quite as well as MPC. This is interesting because the trajectory approach only has access to the grounded observation and must simulate d-steps into the future, where as in the MPC approach the action taken at time t always has access to the observation from time t. These results show that the implicit transition module is indeed useful and could be used in future work for decision-time planning.

5 Conclusion

Introduced in this work is a learning scheme for Policy Gradient methods which integrates model-free and model-based learning that reduces sample complexity at no extra cost in computation at rollout time. Additionally, PPN's implicit transition model acts as a first step towards decision-time planning with tree structured architectures in continuous action-spaces. It is interesting to note that while we only explored continuous action spaces in this work it is also possible to extend this to discrete action spaces.

For future work we would like to adapt PPNs to be less sensitive to planning depth and to leverage the transition model for decision-time planning. Decision-time planning is interesting but not straight forward to apply because it changes the behavior policy distribution in ways that are hard to measure.

References

1. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
2. Chiappa, S., Racaniere, S., Wierstra, D., Mohamed, S.: Recurrent environment simulators. arXiv preprint arXiv:1704.02254 (2017)
3. Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P.: Openai baselines. `https://github.com/openai/baselines` (2017)
4. Farquhar, G., Rocktaeschel, T., Igl, M., Whiteson, S.: TreeQN and ATreec: Differentiable tree planning for deep reinforcement learning. In: International Conference on Learning Representations (2018), `https://openreview.net/forum?id=H1dh6Ax0Z`

5. Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: Advances in Neural Information Processing Systems. pp. 2455–2467 (2018)
6. Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., Madry, A.: Are deep policy gradient algorithms truly policy gradient algorithms? arXiv preprint arXiv:1811.02553 (2018)
7. Kakade, S., Langford, J.: Approximately optimal approximate reinforcement learning. In: ICML. vol. 2, pp. 267–274 (2002)
8. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International conference on machine learning. pp. 1928–1937 (2016)
9. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
10. Oh, J., Guo, X., Lee, H., Lewis, R.L., Singh, S.: Action-conditional video prediction using deep networks in atari games. In: Advances in neural information processing systems. pp. 2863–2871 (2015)
11. Oh, J., Singh, S., Lee, H.: Value prediction network. In: Advances in Neural Information Processing Systems. pp. 6118–6128 (2017)
12. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
13. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning (ICML). vol. 2017 (2017)
14. Schmidhuber, J.: On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models. arXiv preprint arXiv:1511.09249 (2015)
15. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897 (2015)
16. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. In: Proceedings of the International Conference on Learning Representations (ICLR) (2016)
17. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017), `http://arxiv.org/abs/1707.06347`
18. Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D.P., Rabinowitz, N.C., Barreto, A., Degris, T.: The predictron: End-to-end learning and planning. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. pp. 3191–3199 (2017), `http://proceedings.mlr.press/v70/silver17a.html`
19. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Machine Learning Proceedings 1990, pp. 216–224. Elsevier (1990)
20. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
21. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems. pp. 1057–1063 (2000)
22. Todorov, E., Erez, T., Tassa, Y.: 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)