Implementing Trust Region Policy Optimization

Vivswan Shitole
Final Report
CS519 - Convex Optimization

December 3, 2018

Abstract

The reinforcement learning (RL) setting involves an agent interacting with an environment to maximize a reward function. The agent's action at a given state in the environment are determined by its policy, which is a probability distribution over all the possible actions. Trust Region Policy Optimization (TRPO) [Schulman et al.2015] is a recent policy optimization algorithm that guarantees monotonic policy improvement. We reformulate the optimization problem, develop the standard offline algorithm to solve the optimization problem and introduce a new online version of the algorithm that is applicable to certain environments. We borrow results for the standard offline algorithm from [Schulman et al.2015] and append our results for the new online version generated on a custom environment.

1 Introduction

In this section we provide a brief introduction of the model-free reinforcement learning framework over a Markov Decision Process (MDP). We then introduce TRPO as an iterative policy optimization method and state the optimization problem that it tries to solve.

1.1 Model-free Reinforcement Learning

A Markov Decision Process M is defined as the tuple (S, A, P, R, γ), where, S is a finite set of Markov states, A is a finite set of actions, $P : S \times A \times S \to \Re$ is the state transition probability distribution, $R : S \to \Re$ is the immediate reward function and $\gamma \in (0, 1)$ is the discount factor.

In a Markov Decision Process (MDP), all states follow the Markov property: $P(s_{t+1}|s_t, s_{t-1}, .., s_0) = P(s_{t+1}|s_t)$, i.e., the conditional probability of next state s_{t+1} depends only on the current state s_t and not on the sequence of states $\{s_{t-1}, .., s_0\}$ that preceded it. The solution to an MDP is a policy, which is a mapping from states to actions that optimizes a desired objective function. In general, policies are stochastic in that they map states and actions to the probability of taking that action in that state. Typically, the objective function is an expectation of the total reward received when following the policy. In problems with infinite horizon, the reward is geometrically discounted by the time at which it is received to encourage early accumulation of rewards and to keep the total finite.

In model-free reinforcement learning approach to solving an MDP, an agent samples states from MDP M by interacting with environment E. The agent selects its action a from a stochastic policy $\pi(a|s, \theta) : S \times A \to [0, 1]$, parameterized by θ (as a linear function or a neural network). The environment E can be assumed as an oracle generating the next state s_{t+1} using an unknown transition probability matrix P, based on the agent's action a_t (obtained from agent's policy π) at the current state s_t. In addition to the next state s_{t+1}, the environment generates a reward $R(s_t)$ for the agent's action a_t at state s_t. Thus the agent's interaction with the environment generates a sequence of state-action-reward tuples, terminating at a terminal state s_T. This sequence $\{s_0, s_1, .., s_T\}$ constitutes an episode. The cumulative discounted sum of

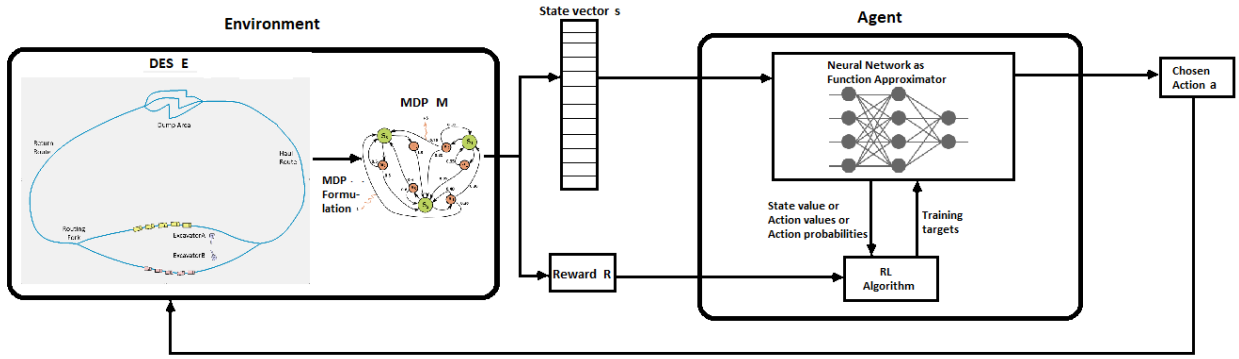


Figure 1: The deep reinforcement learning approach for solving Markov decision problems. On the left is a discrete event simulator which simulates the environment by taking actions on a state and producing the next state and the reward. The state vector is fed into the neural network function approximator, which proposes an action. The RL algorithms update the weights of the neural network based on the difference between the predicted long-term objectives in successive time-steps.

all rewards $\sum_{t=0}^{T} \gamma^t R(s_t) = G$ is the return from the episode. Our goal is to find a stochastic policy π^* that maximizes the expected return G over all the episodes (samples) of the MDP experienced by the agent, i.e.,

$$\pi^* = \arg\max_{\pi} \quad \mathbb{E}_{s_0,a_0,\ldots,s_T,a_T}\left[\sum_{t=0}^{T} \gamma^t R(s_t)\right] \tag{1}$$

where, $s_t \sim P(s_t|s_{t-1}, a_{t-1}), \quad a_t \sim \pi(a_t|s_t)$

1.2 Trust Region Policy Optimization

Trust Region Policy Optimization [Schulman et al.2015] is an iterative method for policy optimization that ensures monotonic policy improvement by using a minorization-maximization (MM) algorithm. It optimizes a surrogate function representing a lower bound on policy improvement, subject to a trust region constraint between the new policy and the old policy:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \mathbb{E}_{s\sim\rho_{\theta_{old}},a\sim q}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{\theta_{old}}(s,a)\right] \\ \text{subject to} \quad & \mathbb{E}_{s\sim\rho_{\theta_{old}}}\left[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))\right] \leq \delta \end{aligned} \tag{2}$$

Thus, the surrogate function to be maximized is an expectation over the parameterized Q-value function $Q_{\theta_{old}}(s,a)$ (defined later) multiplied by an importance sampling weight of the current policy $\pi_\theta(a|s)$ over the action sampling distribution $\pi_{\theta_{old}}(a|s)$. The trust region constraint is that the KL divergence between the old policy $\pi_{\theta_{old}}$ and the current policy π_θ should be within a value δ (a hyperparameter). This ensures that the state distribution visited by the current policy is not too far from that of the old policy so that the value estimates can be trusted.

2 Problem Formulation

In this section, we reformulate TRPO as an optimization problem by stepping through the derivation of Equation 2

Let $\eta(\pi)$ denote the expected return of following policy π for an episode. Then from Equation 1 we have:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots, s_T, a_T} \left[\sum_{t=0}^{T} \gamma^t R(s_t) \right] \tag{3}$$

Similarly, for an infinite horizon MDP, we have:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right] \tag{4}$$

Let $V_\pi(s_t)$ denote the expected return on following policy π from state s_t. This is known as the value function of state s_t and is given as:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+l}) \right] \tag{5}$$

Let $Q_\pi(s_t, a_t)$ denote the expected return on taking action a_t from state s_t and following policy π thereon. This is known as the action-value function of the state-action pair (s_t, a_t) and is given as:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+l}) \right] \tag{6}$$

Then the advantage of taking action a_t from state s_t is given by the advantage function $A_\pi(s_t, a_t)$ as:

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t) \tag{7}$$

In order to obtain monotonic policy improvement guarantee, we want to operate in the space of policy distributions. Expressing the expected return $\eta(\tilde{\pi})$ of a policy $\tilde{\pi}$ in terms of the expected return $\eta(\pi)$ of another policy π would be essential for this scenario. This expression can be obtained using the expected advantage of policy $\tilde{\pi}$ over policy π as follows:

$$\mathbb{E}_{s'_0, a'_0, \ldots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \tag{8}$$

$$= \mathbb{E}_{\tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t) \right) \right] \tag{9}$$

$$= -\mathbb{E}_{s_0} \left[V_\pi(s_0) \right] + \mathbb{E}_{\tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right] \tag{10}$$

$$= -\eta(\pi) + \eta(\tilde{\pi}) \tag{11}$$

On rearranging, we get the desired expression:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s'_0, a'_0, \ldots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \tag{12}$$

Let ρ_π be the discounted state visitation frequencies:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots \tag{13}$$

Then Equation 12 can be written as:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s)\gamma^t A_\pi(s,a) \tag{14}$$

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) A_\pi(s,a) \tag{15}$$

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s,a) \tag{16}$$

This equation implies that any policy update $\pi \to \tilde{\pi}$ that has a nonnegative expected advantage at every state s, i.e., $\sum_a \tilde{\pi}(a|s)A_\pi(s,a) \geq 0$, is guaranteed to increase the policy performance η or leave it constant. The complex dependency of $\rho_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ makes Equation 16 difficult to optimize directly. So we introduce the following local approximation to η, ignoring changes to state visitation frequency $\rho_\pi(s) \to \rho_{\tilde{\pi}}(s)$ due to change in policy $\pi \to \tilde{\pi}$:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_\pi(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s,a) \tag{17}$$

If we have a parameterized policy π_θ, where $\pi_\theta(a|s)$ is a differentaible function of the parameter vector θ, then L_π matches θ to first order [Kakade and Langford2002]. That is for any parameter value θ_0,

$$\begin{aligned} L_{\pi_{\theta_0}}(\pi_{\theta_0}) &= \eta(\pi_{\theta_0}) \\ \nabla_\theta L_{\pi_{\theta_0}}(\pi_{\theta_0})|_{\theta=\theta_0} &= \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_0} \end{aligned} \tag{18}$$

Using this approximation, [Kakade and Langford2002] proposed a mixture of policies update scheme:

$$\begin{aligned} \pi_{new}(a|s) &= (1-\alpha)\pi_{old}(a|s) + \alpha\pi'(a|s) \\ where, \pi' &= \underset{\pi'}{argmax} L_{\pi_{old}}(\pi') \end{aligned} \tag{19}$$

They showed that for the forementioned policy update, the following lower bound can be derived:

$$\begin{aligned} \eta(\pi_{new}) &\geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \\ where, \epsilon &= \max_s |\mathbb{E}_{a\sim\pi'(a|s)}[A_\pi(s,a)]| \end{aligned} \tag{20}$$

[Schulman et al.2015] extended this analysis from the class of mixture policies to the general class of stochastic policies by replacing α with KL divergence as a distance metric between π and π'. They derived the following lower bound:

$$\begin{aligned} \eta(\tilde{\pi}) &\geq L_\pi(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{max}(\pi, \tilde{\pi}) \\ where, \epsilon &= \max_{s,a} |A_\pi(s,a)| \end{aligned} \tag{21}$$

Thus, to optimize the performance $\eta(\tilde{\pi})$, our optimization objective is obtained as (overloading notation of parameterized policy $\pi_\theta \to \theta$):

$$\begin{aligned} \underset{\theta}{maximize} \, [L_{\theta_{old}}(\theta) - CD_{KL}max(\theta_{old}, \theta)] \\ where, C = \frac{4\epsilon\gamma}{(1-\gamma)^2} \end{aligned} \tag{22}$$

Replacing the penalty term in Equation 22 with a constrain on KL divergence, we obtain a more robust optimization problem as:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & L_{\theta_{old}}(\theta) \\ \text{subject to} \quad & D_{KL}max(\theta_{old}, \theta) \leq \delta \end{aligned} \tag{23}$$

The only problem with using Equation 23 as the final formulation is that the objective $L_{\theta_{old}}(\theta)$ involves summing over all states and actions (Equation 17). We want to convert it into an equivalent sampling based objective function. This can be done by using the following two notation replacements:

$$\sum_{s} \rho_{\theta}(s) \rightarrow \mathbb{E}_{s \sim \rho_{\theta}} \tag{24}$$

$$\sum_{a} \pi_{theta}(a|s) A_{\theta_{old}}(s,a) \rightarrow \mathbb{E}_{a \sim \pi_{\theta_{old}}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{\theta_{old}}(s,a) \right] \tag{25}$$

Hence, substituting the changes given in Equation 24 and Equation 25 to Equation 23, our final optimization problem becomes:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \mathbb{E}_{s \sim \rho_{\theta_{old}}, a \sim q} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{\theta_{old}}(s,a) \right] \\ \text{subject to} \quad & \mathbb{E}_{s \sim \rho_{\theta_{old}}} \left[D_{KL}(\pi_{\theta_{old}}(\cdot|s) || \pi_{\theta}(\cdot|s)) \right] \leq \delta \end{aligned} \tag{26}$$

which is the same as Equation 2

3 Developing The Solving Algorithm

In this section, we develop the TRPO algorithm to solve the optimization problem formulated in Equation 26.

3.1 Finding best direction of ascent

From Equation 23 we have:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & L_{\theta_K}(\theta) \\ \text{subject to} \quad & D_{KL}max(\theta_K, \theta) \leq \delta \end{aligned} \tag{27}$$

The hessian of KL divergence is the Fisher Information matrix. This can be shown as:

$$KL\left[p(x|\theta_K)||p(x|\theta)\right] = \underset{p(x|\theta_K)}{\mathbf{E}}[\log p(x|\theta_k)] - \underset{p(x|\theta_K)}{\mathbf{E}}[\log p(x|\theta)] \tag{28}$$

The first derivative with respect to θ is:

$$\nabla_{\theta} KL\left[p(x|\theta_K)||p(x|\theta)\right] = \nabla_{\theta} \underset{p(x|\theta_K)}{\mathbf{E}}[\log p(x|\theta_k)] - \nabla_{\theta} \underset{p(x|\theta_K)}{\mathbf{E}}[\log p(x|\theta)] \tag{29}$$

$$\nabla_{\theta} KL\left[p(x|\theta_K)||p(x|\theta)\right] = - \underset{p(x|\theta_K)}{\mathbf{E}}[\nabla_{\theta} \log p(x|\theta)] \tag{30}$$

$$\nabla_{\theta} KL\left[p(x|\theta_K)||p(x|\theta)\right] = - \int p(x|\theta_K) \nabla_{\theta} \log p(x|\theta) dx \tag{31}$$

The second derivative is:

$$\nabla^2_\theta KL\left[p(x|\theta_K)||p(x|\theta)\right] = -\int p(x|\theta_K)\nabla^2_\theta \log p(x|\theta)dx \tag{32}$$

Therefore, the hessian with respect to θ at $\theta = \theta_K$ is:

$$\mathbf{H}_{KL[p(x|\theta_K)||p(x|\theta)]} = -\int p(x|\theta_K)\nabla^2_\theta \log p(x|\theta)|_{\theta=\theta_K}dx \tag{33}$$

$$\mathbf{H}_{KL[p(x|\theta_K)||p(x|\theta)]} = -\int p(x|\theta_K)\mathbf{H}_{\log p(x|\theta_K)}dx \tag{34}$$

$$\mathbf{H}_{KL[p(x|\theta_K)||p(x|\theta)]} = \mathbf{F} \tag{35}$$

Moreover, the first derivative of KL divergence at $\theta = \theta_K$ is zero:

$$\nabla_\theta KL\left[p(x|\theta_K)||p(x|\theta)\right]|_{\theta=\theta_K} = -\mathop{\mathbf{E}}_{p(x|\theta_k)}[\nabla_{\theta_K}\log p(x|\theta_K)] = 0 \tag{36}$$

Hence, the second order Taylor series approximation of KL divergence is given as:

$$KL\left[p(x|\theta_K)||p(x|\theta)\right] \approx KL\left[p(x|\theta_K)||p(x|\theta_k)\right] + \left(\nabla_\theta KL\left[p(x|\theta_K)||p(x|\theta)|\right]_{\theta=\theta_K}\right)^T(\theta-\theta_K) + \frac{1}{2}(\theta-\theta_K)^T\mathbf{F}(\theta-\theta_K) \tag{37}$$

Using Equation 35 and Equation 36, we have:

$$KL\left[p(x|\theta_K)||p(x|\theta)\right] \approx \frac{1}{2}(\theta-\theta_K)^T\mathbf{F}(\theta-\theta_K) \tag{38}$$

Substituting this in Equation 27, our optimization problem reduces to:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & L_{\theta_K}(\theta) \\ \text{subject to} \quad & \frac{1}{2}(\theta-\theta_K)^T\mathbf{F}(\theta-\theta_K) \le \delta \end{aligned} \tag{39}$$

Using first order Taylor series approximation for the objective:

$$L_{\theta_K}(\theta) \approx L_{\theta_K}(\theta_K) + (\nabla_\theta L_{\theta_K}(\theta))^T(\theta-\theta_K) \tag{40}$$

$$L_{\theta_K}(\theta) \approx (\nabla_\theta L_{\theta_K}(\theta))^T(\theta-\theta_K) \tag{41}$$

The optimization problem reduces to:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & (\nabla_\theta L_{\theta_K}(\theta))^T(\theta-\theta_K) \\ \text{subject to} \quad & \frac{1}{2}(\theta-\theta_K)^T\mathbf{F}(\theta-\theta_K) \le \delta \end{aligned} \tag{42}$$

This is equivalent to:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & -(\nabla_\theta L_{\theta_K}(\theta))^T(\theta-\theta_K) \\ \text{subject to} \quad & \frac{1}{2}(\theta-\theta_K)^T\mathbf{F}(\theta-\theta_K) \le \delta \end{aligned} \tag{43}$$

Let $d = \theta - \theta_K$. Then the direction of steepest gradient ascent is given as:

$$d^* = \underset{d}{argmin} \left[-(\nabla_\theta L_{\theta_K}(\theta))^T d + \lambda(d^T F d - \delta) \right] \tag{44}$$

Thus d^* is calculated by setting derivative with respect to d to zero:

$$-\nabla_\theta L_{\theta_K}(\theta) + \lambda F d^* = 0 \tag{45}$$

$$d^* = \frac{1}{\lambda} \mathbf{F}^{-1} \nabla_\theta L_{\theta_K}(\theta) \tag{46}$$

The factor $\frac{1}{\lambda}$ can be incorporated in the learning rate. Thus,

$$d^* = \mathbf{F}^{-1} \nabla_\theta L_{\theta_K}(\theta) \tag{47}$$

But calculating $\mathbf{F}^{-1}$ is computationally very expensive.
So instead of using Equation 47 to find d^*, we solve:

$$\mathbf{F} d^* = \nabla_\theta L_{\theta_K}(\theta) \tag{48}$$

using conjugate gradient descent.

Conjugate gradient descent allows us to approximately solve Equation 48 without forming the full $\mathbf{F}$ matrix. It merely requires access to a function that computes matrix-vector products $y \to \mathbf{F}y$. Such a mapping can be obtained as:

$$\mathbf{F}y = \nabla(\nabla KL)y = \nabla(diag(\nabla KL)y) \tag{49}$$

Using the above mapping in the standard conjugate gradient descent algorithm, we calculate the best ascent direction d^*. But we still need to find the best step size.

3.2 Finding best step size

Let β be the step size, i.e., $\theta - \theta_k = \beta d^*$. This implies βd^* must satisfy the constrain in Equation 39:

$$\frac{1}{2}(\theta - \theta_K)^T \mathbf{F}(\theta - \theta_K) \leq \delta \tag{50}$$

$$\frac{1}{2}(\beta d^*)^T \mathbf{F}(\beta d^*) \leq \delta \tag{51}$$

maximally,

$$\frac{1}{2}(\beta d^*)^T \mathbf{F}(\beta d^*) = \delta \tag{52}$$

$$\beta = \sqrt{\frac{2\delta}{(d^*)^T \mathbf{F} d^*}} \tag{53}$$

This is the biggest valid step size. However, using this step size sometimes leads to catastrophic degradation of performance. So we use this value as a starting point and use line search with shrinking the step size exponentially until the objective improves.

3.3 TRPO Algorithm

Here is the final algorithm:

Algorithm 1 Trust Region Policy Optimization

Input: initial policy parameters θ_0

1: **for** k = 0,1,2,... **do**
2: Collect set of trajectories on policy $\pi_K = \pi(\theta_K)$
3: Estimate advantages A_{π_K}
4: Form sample estimates for policy gradient $\nabla_\theta L_{\theta_K}(\theta)$ and KL-divergence hessian vector product $\mathbf{F}y$
5: Use conjugate gradient for 10 iterations to obtain $d_k^* \approx \mathbf{F}^{-1}\nabla_\theta L_{\theta_K}(\theta)$
6: Estimate proposed step $\beta d^* = \sqrt{\frac{2\delta}{(d^*)^T \mathbf{F} d^*}} d^*$
7: Perform line search with exponential decay to obtain final step for update: $\Delta_K = \beta' d^*$
8: Update: $\theta_{K+1} = \theta_K + \alpha\Delta_K$

4 Offline TRPO and its extension to Online version [New]

4.1 Offline version and Results

The original TRPO algorithm is offline, i.e., it does not perform any learning *while* an agent is interacting with the environment during an episode, but *after* the agent has completed an episode, generating a trajectory of states and actions $\tau = \{s_0, a_0, s_1, a_1, \cdots s_T\}$. This is because the TRPO algorithm requires an existing set of trajectories to sample from and estimate an average of the advantage $A_{\tilde{\pi}}$ over all sampled trajectories (steps 2 and 3 in Algorithm 1):

$$A_{\tilde{\pi}} = \mathbb{E}_{s'_0, a'_0, \ldots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \tag{54}$$

The original paper [Schulman et al.2015] accomplished this sampling over trajectories using two approaches: *(i) single path* and *(ii) vine*. In single path approach, they collect a sequence of states by sampling $s_0 \sim \rho_0$ and then simulating the policy π for some number of time steps to generate a trajectory $s_0, a_0, s_1, a_1, \cdots s_T$. In vine approach, they first sample $s_0 \sim \rho_0$ and simulate the policy π to generate a number of trajectories. Then they choose a subset of N states (called a "rollout set") along these trajectories, denoted $s_0, s_1, \cdots s_N$. For each state s_n in the rollout set, they sample K actions according to $a_{n,k} \sim \pi(\cdot|s_n)$. For each action $a_{n,k}$ sampled at each state s_n, they estimate $A_{\pi_t}(s_n, a_{n,k})$ by performing a rollout (i.e., a short trajectory) starting with state s_n and action $a_{n,k}$. The two approaches are illustrated in Figure 2

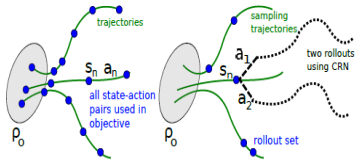


Figure 2: Left: illustration of single path procedure. Here, we generate a set of trajectories via simulation of the policy and incorporate all state-action pairs (s_n; a_n) into the objective. Right: illustration of vine procedure. We generate a set of trunk trajectories, and then generate branch rollouts from a subset of the reached states. For each of these states s_n, we perform multiple actions (a_1 and a_2 here) and perform a rollout after each action.

Using the above approaches, [Schulman et al.2015] implemented the algorithm over 4 different environments and generated the learning curves. The description of these environments along with their results is given below:

- *Cartpole*: A movable cart has to balance a pole standing on it by moving itself left or right on a 1-d line. 4-dimensional state space, 2-dimensional action space, linear reward for duration of an episode.
- *Swimmer*: The swimmer can propoel itself forward by making an undulating motion. 10-dimensional state space, linear reward for forward progress and a quadratic penalty on joint effort to produce the reward $r(x, u) = v_x - 10^{-5}||u||^2$
- *Hopper*: 12-dimensional state space, same reward as swimmer, with a bonus of +1 for being in a non-terminal state. The episode ends when the hopper falls over, which is defined by thresholds on the torso height and angle.
- *Walker*: 18-dimensional state space. For the walker, there is a penalty for strong impacts of the feet against the ground to encourage a smooth walk.

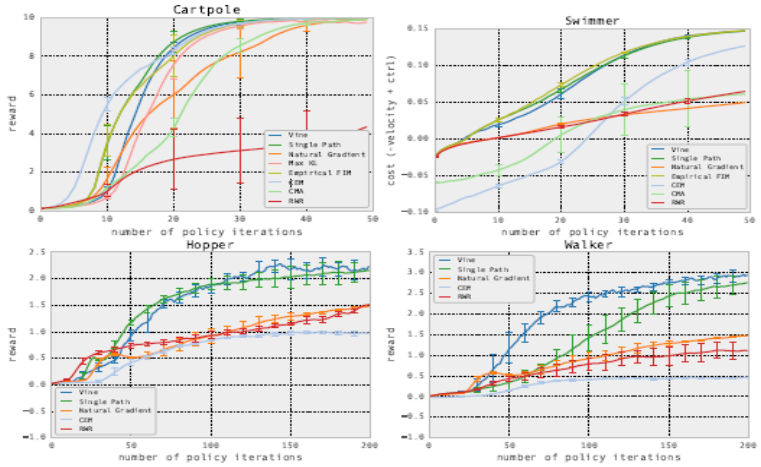


Figure 3: Learning curves for the four locomotion tasks.

4.2 Online version and Results [New]

4.2.1 The idea

In this report, we implemented an online version of the TRPO algorithm in which the agent learns a policy *while* interacting with the environment within an episode. This is achieved by maintaining a moving estimate of the advantage $A_{\tilde{\pi}_t}$ at each state-action pair (s_t, a_t) in an episode:

$$A_{\tilde{\pi}_t} = \left(\frac{t-1}{t}\right)\left[\gamma A_{\tilde{\pi}_{t-1}}\right] + \left(\frac{1}{t}\right)\left[R(s_t) + \gamma V_{\tilde{\pi}}(s_{t+1}) - V_{\tilde{\pi}}(s_t)\right] \tag{55}$$

Note that this is a biased estimate of advantage $A_{\tilde{\pi}}$ as it is estimated on a correlated sequence of trajectories instead of an average over uncorrelated trajectories. But in implementation, we have shown that this biased estimate can be used to learn optimal policies, provided the environment does not have a high branching factor, i.e., does not generate drastically different episode trajectories. We have not proved this proposition mathematically but have shown it via implementation on a simple custom environment with a small branching factor.

4.2.2 Custom environment

We created a custom environment involving an Earthmoving operation, which has a small branching factor. The operation requires the loading and transportation of material from a load area to a dump area by a fleet of trucks. Loading of the trucks is carried out by two excavators operating at two different load sites within the same general loading area. Loaded trucks then go to the dumping site via a common haul route. The dump site has an upper limit on the number of trucks that can unload simultaneously. The unloaded trucks then return to the loading area via a common return route. The environment layout in Fig.4 illustrates the operational model.

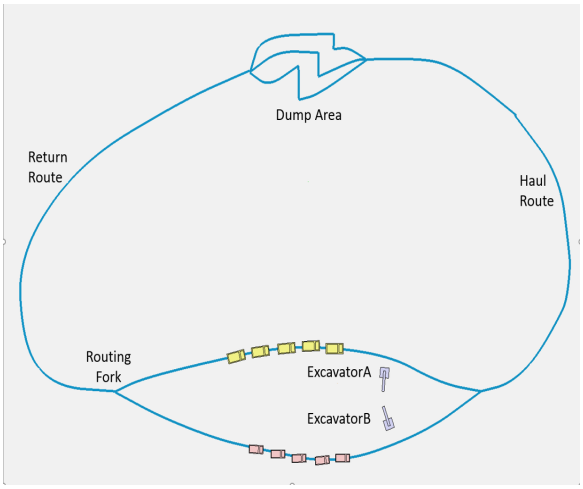


Figure 4: Earthmoving operation layout

To formulate an MDP, the state of the system is represented by a (12×1) vector, with each element representing a different feature of the DES at a given time in the simulation. This state vector, as described in Fig.5, provides a complete snapshot of the DES at a given time, thus respecting the Markov property. The reward function is formulated as the negative of the time of cycle of a truck (the time taken by a truck to exit the routing fork, go through the excavator, the haul route, the dump area, the return route and arrive back at the routing fork). The presence of two excavators in the operation necessitates the routing fork as the decision making element which decides the excavator to which an empty truck should queue up for loading. Thus the routing fork constitutes the RL agent which has to select between the two actions in the action space (Fig.5) and learn an optimal routing strategy as its policy. The production rate at the end of the operation constitutes the return G of the episode.

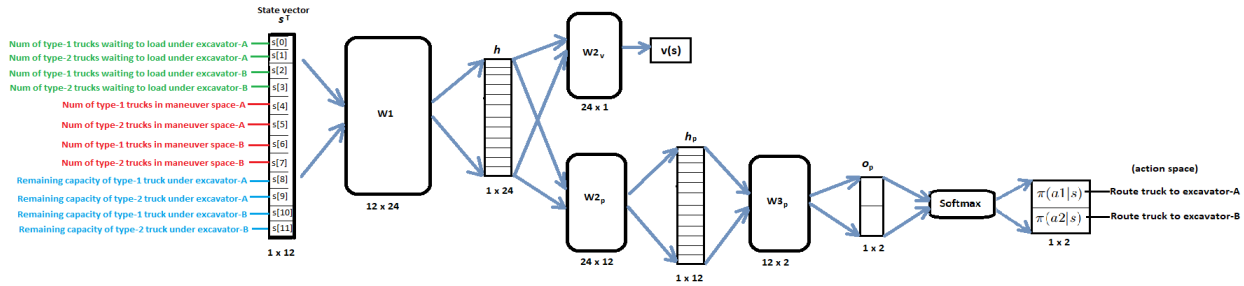
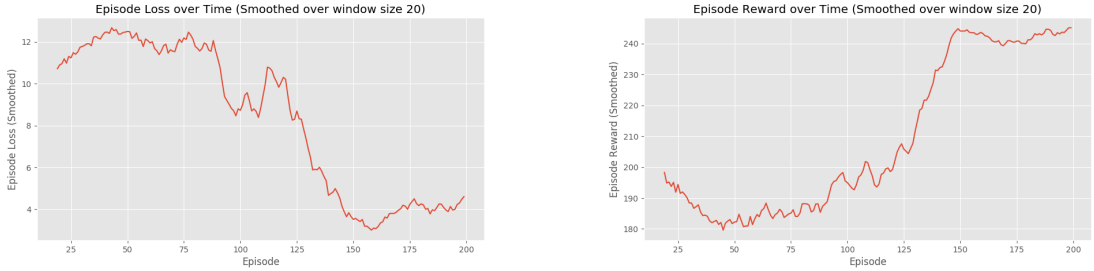


Figure 5: Neural Network architecture used to predict the value of the state and the action probabilities from state vector.

4.2.3 Results

We evaluated the performance of Online TRPO for the forementioned custom environment by generating two performance curves for 200 episodes based on episode loss and episode reward. These results are shown in Figure 6. The complete implementation code for generating these results is uploaded at **https://github.com/viv92/convoptfinal**.



(a) Episode loss

(b) Production rate

Figure 6: Performance of online TRPO on custom environment. The X-axis shows the number of episodes. The Y-axis shows the average episode loss on left (the smaller the better) and the average production rate on right (the higher the better).

Thus we see from the above results that the agent is able to optimize the TRPO objective iteratively and learn policies that increase the episode reward in an online fashion, using a moving estimate of the advantage function.

References

[Kakade and Langford2002] Kakade, S., and Langford, J. 2002. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 267–274.

[Schulman et al.2015] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.