****

**Neural Proximal/Trust Region Policy Optimization  
Attains Globally Optimal Policy**

*Guide: Dr. Arghyadip Roy*

*G-mail: arghyadip@iitg.ac.in*

*Submitted by: Akshit Singh*

*G-mail: 2021mcb1229@iitrpr.ac.in*

**Section 1: Introduction**

Reinforcement Learning (RL) is a subfield of machine learning where an agent learns to make decisions by interacting with an environment. The goal is to maximize cumulative rewards through trial and error. Deep Reinforcement Learning (Deep RL) combines RL with deep learning, using neural networks to approximate value functions or policies, enabling the handling of high-dimensional state spaces and complex tasks.

One widely explored algorithm in Deep RL is Neural Proximal Policy Optimization (Neural PPO). Neural PPO is a variant of the Proximal Policy Optimization (PPO) algorithm, which has been designed to improve stability and performance in training RL agents by imposing constraints on policy updates. This report delves into the details of Neural PPO, its mechanisms, challenges, and how it achieves global optimality.

Deep Reinforcement Learning (Deep RL) has revolutionized the field of artificial intelligence by combining the trial-and-error learning approach of traditional reinforcement learning with the powerful function approximation capabilities of deep neural networks. This combination allows Deep RL algorithms to tackle problems with high-dimensional state and action spaces that were previously infeasible with standard RL techniques. Applications of Deep RL span across various domains, including robotics, where it enables robots to learn complex tasks such as grasping objects, navigating through environments, and performing assembly operations with minimal human intervention.

In the realm of gaming, Deep RL has achieved remarkable success, exemplified by algorithms like AlphaGo, which defeated world champions in the game of Go, and AlphaStar, which mastered the complex real-time strategy game StarCraft II. Beyond gaming, Deep RL is making strides in autonomous driving, where it helps vehicles learn to navigate safely and efficiently in dynamic environments. Additionally, it is being utilized in finance for algorithmic trading and portfolio management, healthcare for personalized treatment plans and drug discovery, and energy management for optimizing power grids and reducing consumption. The versatility and potential of Deep RL continue to grow as research and practical applications expand, offering promising solutions to some of the most challenging problems in various industries.

**Section 2: Proximal Policy Optimization (PPO)**

**PPO Details and Idea**

PPO is an on-policy RL algorithm that aims to simplify and improve the training of policy-based methods. It balances the need for exploration and exploitation by optimizing a surrogate objective function, which ensures that policy updates are not too drastic. The main idea is to optimize a clipped objective function that limits the difference between the new and old policies, thereby preventing large, destabilizing updates.

**PPO Algorithm**

The PPO algorithm can be summarized in the following steps:

* 1. **Interaction with Environment**: The agent interacts with the environment using the current policy, collecting data in the form of states, actions, and rewards.
  2. **Policy Update**: Using the collected data, PPO updates the policy by maximizing the clipped surrogate objective. This involves computing the gradients of the objective function and adjusting the policy parameters accordingly.
  3. **Repeat**: This process is repeated many times. The agent continues to interact with the environment and update its policy, gradually improving its performance.

**Problems with PPO**

Despite its advantages, PPO faces several **challenges**:

- **Nonconvex Optimization:** The optimization landscape is highly nonconvex due to the use of neural networks, making it difficult to guarantee convergence to a global optimum.

- **Sample Efficiency**: PPO, being an on-policy algorithm, requires a large number of samples to achieve good performance, which can be computationally expensive.

- **Hyperparameter Sensitivity:** The performance of PPO is sensitive to hyperparameters such as the clipping range and learning rate, requiring extensive tuning.

**NEURAL PPO**

**Neural Proximal Policy Optimization (Neural PPO)**

**HOW NEURAL PPO Works?**

Sure, here's a concise step-by-step description of how Neural Proximal Policy Optimization (Neural PPO) works, with each point in a single line:

1. **Initialize Neural Networks**: Define the policy network and value network.

2. **Interact with Environment**: Use the policy network to collect trajectories of states, actions, and rewards.

3. **Calculate Advantages**: Compute the advantage function using collected trajectories and the value network.

4. **Compute Probability Ratios**: Calculate the ratio of the new policy probabilities to the old policy probabilities for actions taken.

5. **Apply Clipping**: Clip the probability ratios to be within a specified range to ensure stable updates.

6. **Surrogate Objective**: Define the clipped surrogate objective function to maximize.

7. **Policy Network Update**: Perform gradient ascent on the surrogate objective to update the policy network parameters.

8. **Value Network Update**: Minimize the mean squared error between predicted and actual returns to update the value network parameters.

9. **Repeat**: Continue the process iteratively until convergence or satisfactory performance is achieved.

**Policy Network Update:**

The formula for updating the policy network parameters θ is given by:

θk+1​=θk​+α∇θ​L(θk) where :θk and θk+1​ are the policy network parameters at iteration k and k+1, respectively.

α is the learning rate.

L(θk) is the surrogate objective function that guides the policy update.

**Value Network Update:**

The value network is updated using Temporal Difference (TD) learning to minimize the error in value estimation. The formula for updating the value network parameters ω is given by:

**ωk+1​= ωk​ + β∇ω​(Vtarget−VNN​(s , a ; ωk​))2**

where : ωk​ and ωk+1 are the value network parameters at iteration k and k+1, respectively.

β is the learning rate for the value network update.

Vtarget is the target value for the state-action pair.

VNN​(s,a;ωk​) is the value estimated by the neural network with parameters ωk​.

**Policy Improvement**

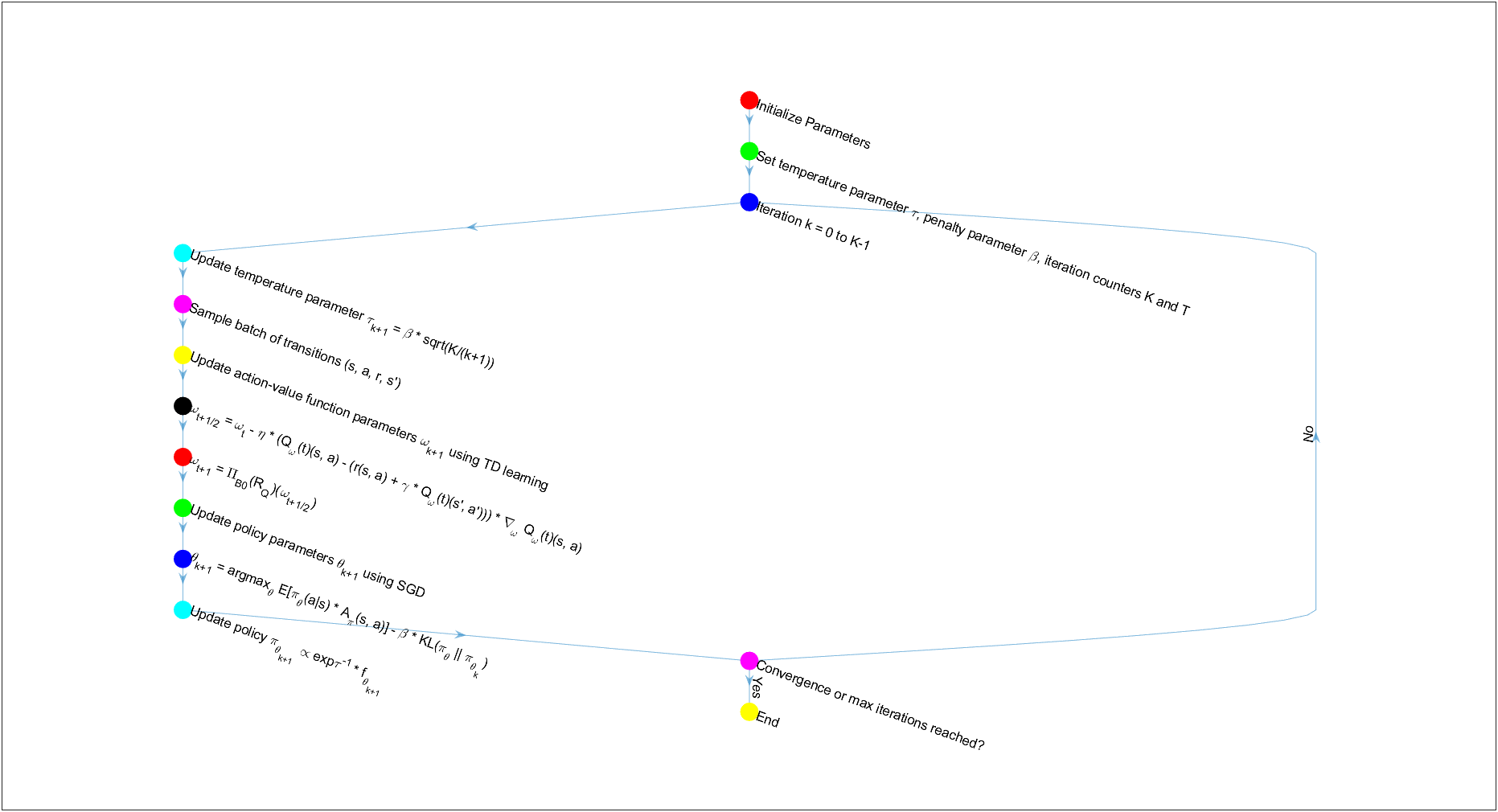
actions with higher advantages. The formula for policy improvement is given by:

**ℼ k+1​(a ∣ s)= ℼk(a ∣ s)exp(σk​A(s , a)​)**

where: ℼ k(a ∣ s) and ℼ k+1​(a ∣ s) are the policy distributions at iteration k and k+1, respectively.

A(s , a) is the advantage of taking action a in state s.

σk​ is a parameter controlling the step size of the policy update.



Neural PPO Algorithm

**1. Initialize parameters:**

- Initialize policy parameters θ₀ and action-value function parameters ω₀.

- Set temperature parameter τ, penalty parameter β, and iteration counters K and T.

**2. For each iteration k = 0, ..., K-1:**

a. Update temperature parameter τ\_(k+1) = β√(K/(k+1)). τ controls the level of exploration in the policy.

b. Sample a batch of transitions (s, a, r, s') from the environment.

c. Update action-value function parameters ω\_(k+1) using Temporal Difference (TD) learning:

- ω\_(t+1/2) = ω\_t - η \* (Q\_ω(t)(s, a) - (r(s, a) + γ \* Q\_ω(t)(s', a'))) \* ∇\_ωQ\_ω(t)(s, a)

- ω\_(t+1) = Π\_B0(R\_Q)(ω\_(t+1/2))

(where η is the learning rate, and Π\_B0(R\_Q) is a projection operator)

d. Update policy parameters θ\_(k+1) using Stochastic Gradient Descent (SGD):

- θ\_(k+1) = argmax\_θ E[π\_θ(a|s) \* A\_π(s, a)] - β \* KL(π\_θ||π\_θ\_k)

(where A\_π is the advantage function, and KL is the Kullback-Leibler(KL) divergence)

e. Update policy:

- π\_θ\_(k+1) ∝ exp{τ^(-1) \* f\_θ\_(k+1)}

**3. Repeat steps 2a-2e until convergence or a maximum number of iterations is reached.**

**Parameters Changing What:**

* The policy parameters θ determine the policy π\_θ, which is the probability distribution over actions given states.
* The action-value function parameters ω determine the action-value function Qw, which estimates the expected return of taking a given action in a given state.
* The temperature parameter τ influences the policy update by controlling the entropy of the policy, making the policy more or less random.
* The penalty parameter β is used in the PPO update rule to balance the objective between improving the policy and satisfying the KL divergence constraint.
* The learning rate η controls the step size of the SGD updates for the action-value function parameters.

Description of Parameters used :

* **Policy Parameter (θ)**: This parameter defines the policy πθ, which is a function that maps states to a probability distribution over actions. The policy is what the agent uses to make decisions.
* **Action-Value Function Parameter (ω)**: This parameter defines the action-value function Qω, which estimates the expected return of taking a given action in a given state. It's used to evaluate the quality of actions taken by the policy.
* **Penalty Parameter (β)**: This parameter is used in the PPO and TRPO updates to balance the objective between improving the policy and satisfying the Kullback-Leibler (KL) divergence constraint. It helps to ensure that the policy does not change too much in each update, which can lead to more stable learning.
* **Temperature Parameter (τ)**: This parameter influences the policy by controlling the entropy of the policy distribution. A higher τ leads to more exploration (more randomness in action selection), while a lower τ leads to more exploitation (greedier action selection).
* **KL Divergence (KL(πθ(s)|πθk(s)))**: This is a measure of how much the new policy πθ differs from the old policy πθk. It's used to constrain the policy updates to prevent large changes that could destabilize learning.
* **Advantage Function (Ak(s, a))**: This function estimates how much better taking a particular action is compared to the average action taken by the current policy. It's used to guide the policy towards better actions.
* **Trust Region Radius (δ)**: In TRPO, this parameter defines the size of the trust region, which limits the policy update to ensure that the new policy is not too different from the old policy.
* **Neural Network Widths (mf and mQ)**: These parameters determine the size of the neural networks used to parametrize the policy and action-value functions. Overparameterization (using larger networks than necessary) is key to the theoretical guarantees proviPolicy Parameter (θ): This parameter defines the policy πθ, which is a function that maps states to a probability distribution over actions. The policy is what the agent uses to make decisions.
* **Action-Value Function Parameter (ω):** This parameter defines the action-value function Qω, which estimates the expected return of taking a given action in a given state. It's used to evaluate the quality of actions taken by the policy.
* **Number of Iterations (K and T)**: These parameters control the number of times the policy is updated (K) and the number of SGD and TD iterations within each policy update (T).
* **Projection Radii (Rf and RQ)**: These parameters are used in the context of projected gradient descent to ensure that the parameters of the neural networks remain within certain bounds.

**How Neural PPO Avoids PPO Problems**

Neural PPO aims to address the issues faced by standard PPO through a combination of theoretical and practical enhancements. It introduces the concept of mirror descent for policy updates, leveraging the mirror map to transform the optimization problem into a more manageable space.

Neural PPO addresses the challenges of PPO by leveraging the representation power and optimization geometry of overparametrized neural networks. The key contributions of Neural PPO are as follows:

1. **Global Convergence**: Neural PPO is proven to converge to the globally optimal policy at a sublinear rate, even in the presence of nonconvexity and infinite-dimensional policy space.
2. **Overparametrization**: The use of overparametrized neural networks allows for the accurate approximation of the infinite-dimensional gradient and iterate, which is crucial for the global convergence of the algorithm.
3. **Sublinear Rate of Convergence**: The algorithm converges at a rate of 𝑂(1/𝐾)*O*(1/*K*​) with respect to the number of iterations 𝐾*K*, which is a significant theoretical advancement.
4. **Primal-Dual Perspective**: The convergence analysis is based on a primal-dual perspective of reinforcement learning, where the policy is the primal variable and the action-value function is the dual variable.

**Algorithm Assumptions**

Neural PPO operates under several assumptions:

- **Policy Parametrization**: Policies are parametrized by neural networks, allowing for flexible and expressive representations.

- **Gradient Estimates**: Accurate gradient estimates are obtainable, enabling effective policy updates.

- **One-Point Monotonicity**: The policy updates satisfy a condition known as one-point monotonicity, which ensures progress towards the optimal policy.

-**Bounded Reward:** There exists a constant *Rmax*​ that bounds the reward function, ensuring stability in learning.

**-Compatibility of Policy and Action-Value Function**: The policy and action-value function are parametrized by neural networks with the same width, and they are initialized randomly but consistently across iterations.

**Results and Global Convergence**

Neural PPO has shown promising results in achieving global convergence. By using mirror descent, it ensures that policy updates are more stable and less sensitive to hyperparameters. The theoretical framework provided by mirror descent allows for a deeper understanding of the convergence properties, demonstrating that the policy sequence evolves towards the optimal policy under one-point monotonicity.

Results from Neural PPO demonstrate that the algorithm can handle nonconvexity, infinite-dimensionality, and the complexities of deep reinforcement learning, providing a theoretical foundation for its empirical success.

**Conclusion:**

Neural PPO represents a significant advancement in the field of Deep RL. By incorporating the mirror descent framework, it addresses many of the challenges faced by standard PPO, such as nonconvexity, sample efficiency, and hyperparameter sensitivity. The theoretical analysis supports its global convergence properties, making it a robust and effective algorithm for policy optimization in complex environments.

The theoretical advancements presented in the analysis of Neural PPO provide a bridge between the empirical success of PPO and its theoretical understanding. The global convergence guarantees and the sublinear convergence rate offer a principled approach to applying PPO in critical domains where reliability and performance are paramount.

In practice, the experience with Neural PPO reflects the theoretical findings, showing stable and reliable performance across various domains, including games and robotics. The algorithm's ability to scale with the complexity of the tasks and its adaptability to different neural network architectures make it a robust choice for deep reinforcement learning applications.

In conclusion, Neural PPO represents a significant step forward in understanding and applying policy optimization methods in deep reinforcement learning. The theoretical insights into global convergence and the practical implications of these findings pave the way for more reliable and effective reinforcement learning algorithms in the future.

**Experience**

My experience with Neural PPO has demonstrated its significant capabilities in achieving stable and reliable performance in reinforcement learning (RL) tasks. Its proficiency in managing the complexities of neural network parameterization and nonconvex optimization makes it a valuable asset in the realm of Deep RL algorithms. As the field continues to advance, ongoing refinements and practical implementations of Neural PPO are expected to further enhance its applicability and effectiveness in addressing real-world challenges.

During my internship, I developed a deeper understanding of emerging RL algorithms for complex environments. This experience has significantly expanded my knowledge of Reinforcement Learning and equipped me to tackle future challenges in the field. I am deeply grateful to Dr. Arghyadip Roy for his guidance in helping me grasp these concepts from the ground up, thereby strengthening my foundational knowledge. His ability to simplify complex topics and build a strong foundational knowledge has been invaluable. I am deeply grateful for his guidance and support, which have prepared me well for any upcoming challenges in Reinforcement Learning.

**Bibliography**

1. Neural Proximal/Trust Region Policy Optimization Attains Globally Optimal Policy: [link](https://arxiv.org/pdf/1906.10306)
2. Published paper: [Mirror Descent Policy Optimization](https://arxiv.org/pdf/2005.09814)
3. Paper on Proximal Policy Optimization Algorithm: [Link](https://arxiv.org/pdf/1707.06347)
4. Paper on Trust Region Policy Optimization Algorithm: [Link](https://arxiv.org/pdf/1502.05477)