<https://arxiv.org/pdf/1707.06347v2>  
<https://paperswithcode.com/method/ppo>  
<https://github.com/bnelo12/PPO-Implemnetation>

<https://www.researchgate.net/publication/339651408_Federated_Reinforcement_Learning_for_Training_Control_Policies_on_Multiple_IoT_Devices/figures?lo=1>

<https://openai.com/index/roboschool/>



**A Comprehensive Tutorial on Proximal Policy Optimization (PPO) in Reinforcement Learning:**

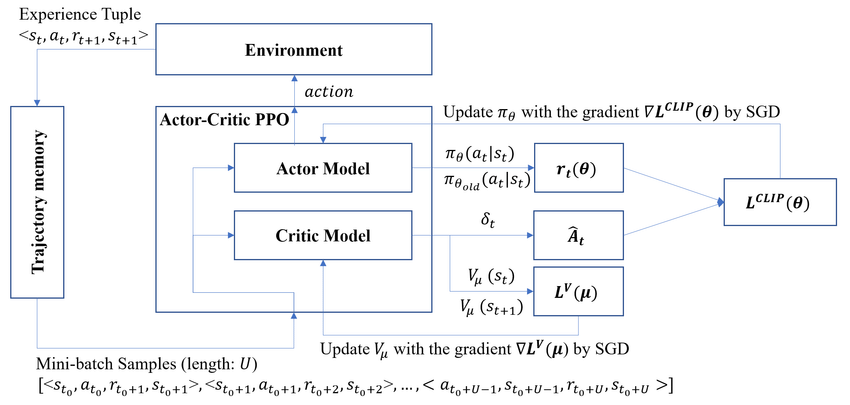
**Introduction:**

Proximal Policy Optimization, or PPO, was introduced by OpenAI back in 2017. It is a reinforcement learning method whereby an agent learns to make decisions by interacting with an environment to accomplish a goal. It is an on-policy gradient algorithm and unlike value-based methods such as Q-learning, policy gradients optimize continuous and probabilistic policies directly by optimizing expected rewards through gradient-based methods hence making them appropriate for tasks with complex or continuous action spaces.  
  
It extends a number of on-policy algorithms, such as Trust Region Policy Optimisation. TRPO's are good at making stable policy updates, but their reliance on computationally expensive second-order optimization techniques makes scaling and implementing them problematic. PPO simplifies this process by making first-order gradient updates while still maintaining stability and performance with the following two techniques:

1. Actor-Critic Architecture: Combines a policy network (actor) to decide actions and a value network (critic) to evaluate the actions
2. Clipped surrogate objective: Stabilizes learning by restricting overly large updates to the policy.
3. Advantage function : determines the direction and magnitude of updates for the policy.

How these 3 interact:

* **The Clipped Surrogate Objective**: Uses to determine how the policy should be updated. Positive advantages increase the probability of the corresponding actions, while negative advantages decrease it.
* **Critic Network**: Trains to accurately estimate , which is then used to calculate ​. This enables the actor (policy) to improve based on accurate feedback.

Visually:

PPO’s simplicity and versatility allow it to address a wide range of practical challenges across various domains:

**Simulated Robotic Locomotion**: PPO has great efficiency in continuous control problems and outperformed similar algorithms in extensive benchmarks of simulated robotic control tasks, such as Hopper, Walker2d, HalfCheetah using the MuJoCo physics engine.

**3D Humanoid**: PPO showcases great performance on some high-dimensional humanoid control tasks such as running, steering, and recovery while getting knocked down during interaction with obstacles (e.g. cubes) particularly proving efficiency in Roboschool’s Humanoid, HumanoidFlagrun, and HumanoidFlagrunHarder

**Atari Game Playing**: PPO shows competitive performance in game AI benchmarks like Atari games. It does better than state-of-the-art algorithms such as A2C in 30 out of 49 games tested.  It also outperformed ACER in terms of sample efficiency and ease of tuning, making it a practical choice for training agents in diverse and challenging gaming environments, especially in long-term strategy games like Assault and Atlantis.

**Challenges Addressed by PPO:**

**Stability vs. Performance Trade-off**:

In many RL methods, large updates to the policy can destabilize learning, while small updates slow convergence. PPO addresses this with a clipped objective function that constrains the size of policy updates, ensuring stable learning while maintaining high performance.

**Sample Efficiency**:

Unlike methods like TRPO, which require second-order optimization techniques and large computational resources, PPO uses simple first-order gradient updates, making it computationally efficient and scalable.

**Flexibility Across Tasks**:

PPO works well for both discrete action tasks (e.g., games) and continuous action tasks (e.g., robotics), without requiring significant modifications.

**Hyperparameter Sensitivity**:

PPO is less sensitive to hyperparameters compared to other RL algorithms, making it easier to implement and tune in real-world scenarios.

**Theoretical Explanation**

**Provide a conceptual explanation of the method, outlining its key mechanisms. Include sufficient**

**mathematical detail to explain its functionality (e.g., loss functions or core algorithm steps), but avoid**

**complex derivations. The focus should remain on accessibility and clarity.**

**Key mechanisms in PPO:**

**I THINK ITS WORTH WRITING ABOUT THE GRADIENT HERE.**

**Neural Network maybe**

***The policy gradient theorem states that the gradient of the expected return with respect to the policy parameters* \theta  *is proportional to the expected value of the gradient of the log policy multiplied by the advantage function:***

**Policy gradient loss**

**ENTROPY**

**Surrogate objective Function:** This is a function used to approximate the true objective of maximizing cumulative rewards. In PPO we use a probability of old and new policies, this encourages to prioritise actions with higher probabilities.

PPO uses first-order gradient updates (e.g. adam optimizer or stochastic gradient descent) eliminating the need of computationally expensive conjugate gradient methods like KL divergence used in TRPO. It also utilises clipping**,** which is a ‘clip’ of the surrogate objective in PPO to ensure policy updates are not too large to keep the policy stable and is a hallmark feature in modern day PPOs. It basically sets a boundary or ‘clipping range’ which if the surrogate objective function crosses, the clipped term is used hence penalizing updates that move too far. Following the equation:

It alsoleverages the actor-critic architecture, where two components (using neural networks) work together to determine the policy and evaluate actions using the help of an advantage function which measures how much better (or worse) a specific action  a  is compared to the average actions the agent would take in that state under the policy, π.

Where,

is the likelihood ratio of old and new policies,

At is the advantage function that indicates the relative value of an action (more on this later)

Clipping range is 1 to 1.

The advantage function, ​, quantifies the relative quality of an action compared to the expected action:

Where:

* is the state-action value function. (OR IS IT THE REWARD)
* is the value function estimating the expected return from .

This helps focus updates on actions that are significantly better or worse than average. In practice, is often approximated using **Generalized Advantage Estimation (GAE)**, which balances bias and variance.

In the context of PPO we have:

* The **actor (Policy Network)** determines the policy (responsible for deciding the actions based on current state).
* The **critic (Value Network)** evaluates the quality of actions by estimating the value function, providing feedback to improve the policy. (The value network predicts the value of a given state  V(s) , which is used to compute the advantage function).

**Where ​ is the discounted return.**

**The actor and critic networks are trained simultaneously, ensuring efficient learning.**

Core Algorithm Steps

1. Initialization:
   * Initialize policy ​ (actor) and value function ​ (critic).
   * Set hyperparameters like clipping range , learning rates, and batch size.`
2. Rollout Collection:
   * Run the policy in the environment for T timesteps to collect trajectories .
3. Advantage Estimation:
   * Compute advantage estimates ​ using a suitable method (e.g., Generalized Advantage Estimation, GAE).
4. Optimization:
   * Update the policy network by maximizing the clipped surrogate objective .
5. Update the value network by minimizing the value loss .
6. Policy Update:
   * Use a first-order optimizer (e.g., Adam) for stable convergence.
7. Repeat:

* Iterate over multiple epochs until convergence or the budgeted interactions are exhausted.

****

****

****

**Code Demonstration/visualisations**

**Clipping function graphs:**

We will implement PPO to train an agent on the **CartPole-v1** environment from OpenAI Gym using PyTorch. The agent learns to balance a pole on a cart by applying forces to the left or right.

[**https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2**](https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2)

**Learning Curve**

**Interpretation**

* **Initial Performance**: The agent starts with a low total reward, indicating random or suboptimal actions.
* **Learning Progress**: The total reward increases over episodes, showing the agent's improving ability to balance the pole.
* **Stability**: The curve demonstrates steady learning without significant drops in performance, highlighting PPO's stability.

**Bioengineering Contextualization (Bonus)**

**Application in Prosthetic Control**

In bioengineering, controlling prosthetic limbs requires adapting to the user's intentions and the environment. PPO can be used to train control policies that interpret sensor data (e.g., electromyography signals) to produce smooth and responsive movements.

* **Adaptive Learning**: PPO's stability ensures that the prosthetic can adjust to changes without erratic behavior.
* **Real-Time Performance**: Efficient policy updates allow for near real-time adjustments, crucial for user comfort.

**Optimization in Drug Delivery Systems**

PPO can optimize dosing strategies in automated drug delivery devices, such as insulin pumps for diabetics.

* **Personalization**: The agent learns optimal dosing schedules based on patient-specific data.
* **Safety**: The clipped objective in PPO prevents drastic changes in dosing, enhancing patient safety.

**Conclusion**

Proximal Policy Optimization is a robust and efficient algorithm in the reinforcement learning landscape. Its balance between simplicity and performance makes it a valuable tool for both research and practical applications.

In this tutorial, we've:

* **Explained the theoretical foundations** of PPO, focusing on its clipped surrogate objective and policy update mechanism.
* **Demonstrated an implementation** in a classic control problem, providing insights into practical aspects of training an RL agent.
* **Visualized and interpreted results**, highlighting the strengths of PPO in stable and efficient learning.
* **Connected the method to bioengineering**, showcasing potential real-world applications that can benefit from PPO's capabilities.

By leveraging PPO, bioengineers and researchers can develop intelligent systems that adapt and learn from interactions, pushing the boundaries of what's possible in technology and healthcare.