Proximal Policy Optimization
Christopher Mutschler
Proximal Policy Optimization (PPO)

- The main motivation behind PPO is the same as for TRPO:
  - Make the biggest possible improvement step
  - Do not step too far such that the performance accidentally collapses

- PPO addresses the shortcomings of TRPO:
  - PPO uses 1\textsuperscript{st} order methods with a few tricks
  - Significantly simpler to implement
  - Shows similar performance to TRPO empirically

- There are two variants:
  - PPO-penalty: TRPO with KL-penalization instead of constraint (penalty coefficient is adjusted and scaled automatically over the course of training: \textit{Adaptive KL Penalty Coefficient})
  - PPO-clip: no constraints! Adds a clipping to the objective function to remove incentives to move too far

\textit{Spoiler: PPO is (1) much simpler to understand and to implement, and (2) much better (empirically)}
Proximal Policy Optimization (PPO)

Where are we so far?

- TRPO maximizes a “surrogate” objective subject to a constraint on the size of the policy update:

\[
\max_{\theta} \mathbb{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right], \quad \text{subject to } \mathbb{E}_t \left[ KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right] \leq \delta
\]

with \(\theta_{old}\) being the policy parameters before the update.

- We did not explicitly formulate it like this, but the intuition behind it is:
  - We want to measure how \(\pi_\theta\) performs relative to \(\pi_{\theta_{old}}\) (using data from the old policy).
  - The original objective (see TRPO slides) can be exactly reformulated to this one.
  - We can solve this with CG after making a linear approximation to the objective and a quadratic approximation to the KL-constraint.
Proximal Policy Optimization (PPO)

Where are we so far?

• TRPO maximizes a “surrogate” objective subject to a constraint on the size of the policy update:

$$\max_{\theta} \mathbb{E}_t \left[ \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right], \quad \text{subject to} \quad \mathbb{E}_t \left[ KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_{\theta}(\cdot|s_t)] \right] \leq \delta$$

with $\theta_{old}$ being the policy parameters before the update

• Let us define the probability ratio $r_t(\theta)$:

$$r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

i.e., $r_t(\theta_{old}) = 1$.

• In other words, TRPO maximizes the following objective:

$$L_{CP1}(\theta) = \mathbb{E}_t [r_t(\theta)\hat{A}_t]$$

penalizing changes to the policy that move $r_t(\theta)$ (too far) away from 1.
Proximal Policy Optimization (PPO)

- The PPO objective we want to maximize is given by

\[
L(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right],
\]

where \( \epsilon \) is a hyperparameter (i.e., 0.1 or 0.2) that defines how far \( \pi_{new} \) may go away from \( \pi_{old} \)

- First term inside the min is \( L^{CPI}(\theta) \)
- Second term inside the min clip the probability ratio
  \( \rightarrow \) removes the incentive for moving \( r_t \) outside of the interval \([1 - \epsilon, 1 + \epsilon]\)
- We take the minimum of the clipped and unclipped objective
  \( \rightarrow \) the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective
Proximal Policy Optimization (PPO)

- The clipping operator is a pessimistic bound of the unclipped objective

![Graph](image)

- Plot show a single timestep of the surrogate function $L^{CLIP}$ as a function of $r$
- The red circle shows the starting point for the optimization, i.e., $r = 1$
Proximal Policy Optimization (PPO)

• Automated Tuning of the gradient step \textit{without calculating the Hessian}

Algorithm 1 PPO-Clip
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, \ldots$ do
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_t\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
6: Update the policy by maximizing the PPO-Clip objective:

$$
\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, \frac{A_{\pi_{\theta_k}}(s_t, a_t)}{g(\epsilon, A_{\pi_{\theta_k}}(s_t, a_t))} \right),
$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$
\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left( V_{\phi}(s_t) - \hat{R}_t \right)^2,
$$

typically via some gradient descent algorithm.
8: end for

Advantage Clipping for conservative policy updates
Proximal Policy Optimization (PPO)

• Results of PPO-clip:
  • Against well-known competitors
  • On well-known environments
• Those results are impressive!
Proximal Policy Optimization (PPO)

Videos from https://openai.com/blog/openai-baselines-ppo/
Proximal Policy Optimization (PPO)

Practical Considerations

Algorithm 1 PPO, Actor-Critic Style

for iteration=1, 2, ⋯ do
  for actor=1, 2, ⋯, N do
    Run policy \( \pi_{\theta_{\text{old}}} \) in environment for \( T \) timesteps
    Compute advantage estimates \( \hat{A}_1, \ldots, \hat{A}_T \)
  end for
  Optimize surrogate \( L \) wrt \( \theta \), with \( K \) epochs and minibatch size \( M \leq NT \)
  \( \theta_{\text{old}} \leftarrow \theta \)
end for

• There is two alternating threads in PPO:
  1. Policy interacts with the environment, collects data and computes advantage estimates (using fitted baselines estimates)
  2. 2nd thread collects all the experiences and runs SGD to optimize the policy using the clipped objective
PPO in Action: OpenAI Five on DOTA II

Optimizer + Connected Rollout Workers (x256)

**Rollout Workers**
- ~500 CPUs
- Run episodes
  - 80% against current bot
  - 20% against mixture of past versions
- Randomized game settings
- Push data every 60s of gameplay
  - Discount rewards across the 60s using generalized advantage estimation

**Optimizer**
- 1 p100 GPU
  - Compute Gradients
    - Proximal Policy Optimization with Adam
    - Batches of 40% observations
    - BPPT over 16 observations

**Eval Workers**
- ~2500 CPUs
- Play in various environments for evaluation
  - vs hardcoded “scripted” bot
  - vs previous similar bots (used to compute TrueSkill)
  - vs self (for humans to watch and analyze)

---

PPO in Action: OpenAI Five on DOTA II

1. **Optimizer + Connected Rollout Workers (x256)**
   - **Rollout Workers**
     - ~500 CPUs
     - Run episodes
       - 80% against current bot
       - 20% against mixture of past versions
     - Randomized game settings
     - Push data every 60s of gameplay
       - Discount rewards across the 60s using generalized advantage estimation
   - **Optimizer**
     - 1 p100 GPU
     - Compute Gradients
       - Proximal Policy Optimization with Adam
       - Batches of 40% observations
       - BPPT over 16 observations

2. **Eval Workers**
   - ~2500 CPUs
   - Play in various environments for evaluation
     - vs hardcoded “scripted” bot
     - vs previous similar bots (used to compute TrueSkill)
     - vs self (for humans to watch and analyze)

---

https://openai.com/blog/openai-five/
Proximal Policy Optimization (PPO)

Practical Considerations
• One more thing…. 

• PPO combines a few more things in the final objective:

\[
L_t^{\text{CLIP+VF+S}}(\theta) = \mathbb{E}_t[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)]
\]

• Note: you likely need similar features to represent the policy and the state-values
  → OpenAI Five shares parameters between the policy network and the value network
  → Both error terms are combined in a single loss function
PPO in Action: OpenAI Five vs. DOTA II

Proximal Policy Optimization

https://intellabs.github.io/coach/components/agents/policy_optimization/ppo.html
PPO in Action: OpenAI Five vs. DOTA II

Bill Gates (@BillGates)

AI bots just beat humans at the video game Dota 2. That’s a big deal, because their victory required teamwork and collaboration—a huge milestone in advancing artificial intelligence.

via Twitter

---

June 30, 2018

OpenAI Five’s parameters initialized.

August 5, 2018

OpenAI Five defeats popular casters at the Benchmark in front of a live audience and 100k livestream viewers, with somewhat restricted 5v5.
PPO in Action: OpenAI Five vs. DOTA II

- **High-level info:**
  - 180 years of self-play per day and hero (~900 yrs/day), no human data
  - Running on 256 P100 GPUs and 128,000 CPU cores

- **Technical stats:**
  - Observation size: ~36.8kB @ ~7Hz
  - Batch size: 1,048,576 observations = 36 GB 😊
  - Separate single-layer, 1024 unit LSTM per hero
  - See [https://openai.com/blog/openai-five/](https://openai.com/blog/openai-five/) for an interactive demo!
  - Reward: net worth, kills, deaths, assist, last hits, etc.
  - "Team spirit" – trade own rewards over team reward (heroes do not communicate)

- **Challenge:** exploring combinatorial-vast space of combining actions w/ long planning horizons
  - 80% of games against itself, 20% against past selves (avoid “strategy collapse”)
  - After several hours: concepts such as laning, farming or fighting emerged
  - After several days: basic human strategies such as steal bounty runes from opponents, rotate heroes around the map to gain lane advantage etc.