A tutorial primer for getting started with RL in a seamless way

RL Bootcamp 2025

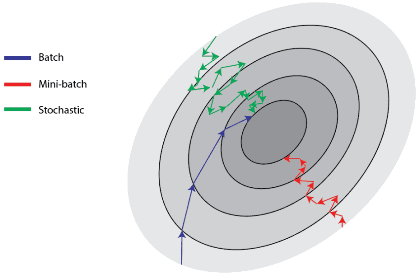
Thomas Gallien[1,2]

[1]Institute for Robotics and Flexible Production
JOANNEUM RESEARCH

[2]RL Community
AI Austria

18.09.2025

Batch Size vs. Learning Rate for Stochastic Gradient Decent



- We use stochastic gradient optimization, hence:
  - parameters are updated with unbiased gradient estimates
  - this is still a random variable $\Longrightarrow$ noisy!
  - variance of the gradient estimate is indirectly proportional to the **batch size**.
  - the larger the batch size, the fewer updates per epoch
  - tradeoff noise suppression and number updates
- Recommendation
  - keep batch size either default or at "usual" values (64,128,256)
  - adjust learning rate during hyperparameter search
  - reduces one dimension in search space

The Impact of the Trust Region

Line search
(like gradient ascent)

Trust region

- TRPO Objective:

$$\hat{\boldsymbol{\theta}}' = \arg\max_{\boldsymbol{\theta}'} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi_{\boldsymbol{\theta}}}\left[\frac{\pi_{\boldsymbol{\theta}'}(\mathbf{a}|\mathbf{s})}{\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})} A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s},\mathbf{a})\right]$$

subject to

$$\mathbb{E}_{\mathbf{s}}\left[D_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})||\pi_{\boldsymbol{\theta}'}(\mathbf{a}|\mathbf{s})\right)\right] \leq \delta$$

- Analytical Solution: [1]

$$\Delta\boldsymbol{\theta}_{\mathrm{TRPO}} = \sqrt{\frac{2\delta}{\mathbf{g}^T \mathbf{F}_{\pi_{\boldsymbol{\theta}}}^{-1}\mathbf{g}}}\mathbf{F}_{\pi_{\boldsymbol{\theta}}}^{-1}\mathbf{g}$$

- Parameter Update: $\Delta\boldsymbol{\theta} \propto \alpha\sqrt{\delta}$ (α ... learning rate)
- $\Longrightarrow$ parameter step size is also influenced by trust region hyper parameter
- Similar dynamics also for PPO through clipped importance ratios (ϵ)

[1] image credit: `https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9`

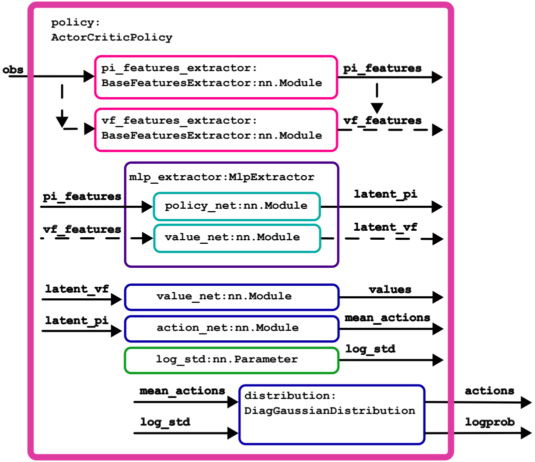
Composition of Stable Baselines 3's PPO

```
PPO:
  policy: ActorCriticPolicy
  rollout_buffer: RolloutBuffer

collect_rollouts()
train()
learn(callbacks)
```

- Rollout Buffer:
  - stores observations, actions, rewards, values and logprobs
  - calculates advantages via generalized advantage estimation
- Policy:
  - defines value and policy networks
  - defines model distribution (Gaussian, Categorical or state dependent Exploration (Gaussian))
- Methods:
  - learn(): main interface method
  - collect_rollouts(): interfaces with the environment and buffers trajectories, rewards, values and logprobs
  - train(): core method to update network parameters

Looking under the Hood: Policy



- Feature Extractor:
  - neural network(s) processing observations and outputs intermediate feature representations
  - if **share_features_extractor=True** only one network is used
- MLP Extractor:
  - some simple dense layers producing intermediate latent space
- value_net/action_net: final value and (mean) action heads
- log_std: **free running parameter** representing the logarithm of the standard deviation

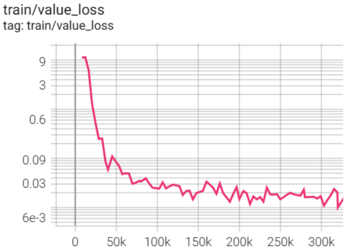
Rollout buffer

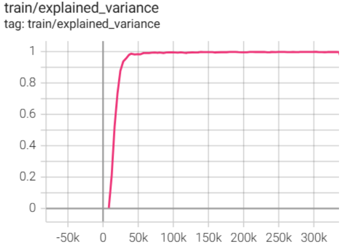
- Generalized Advantage Estimator:

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l},$$
$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- Discounting γ controls how "farsighted" the agent behaves
- λ controls how long past TD errors remain **eligible**
- Recommendation;
    - Start with defaults values for both hyper-parameters
    - Modify γ (**gamma**) to improve credit assignment
    - Touch λ (**gae_lambda**) only if you really know what you are doing

How the Value Estimator Should Behave





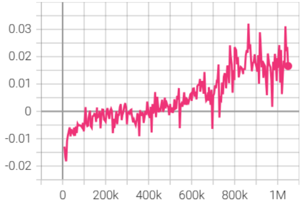
- PPO Objective:

$$\mathcal{L}(\theta') = \mathbb{E}_t \left[\min \left(\frac{\pi_{\theta'}(\mathbf{a}|\mathbf{s})}{\pi_{\theta}(\mathbf{a}|\mathbf{s})} {\color{red}A^{\pi_\theta}}, \mathrm{clip} \left(\frac{\pi_{\theta'}(\mathbf{a}|\mathbf{s})}{\pi_{\theta}(\mathbf{a}|\mathbf{s})}, 1-\epsilon, 1+\epsilon \right) {\color{red}A^{\pi_\theta}} \right) \right]$$

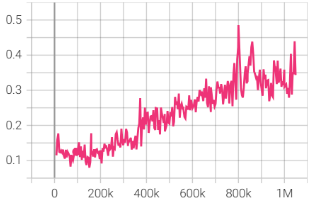
- apparently, policy updates heavily depend on advantage estimates
- $\Longrightarrow$ Value estimator is required to converge **fast**
- per default we use share the feature extractor for both networks, so
  - take care the losses are somehow balances
  - if unbalanced, the value loss usually "overwhelms" gradients for the policy
  - mitigation: downscale value loss parameter (**vf_coef**) or clip the value estimator (**clip_range_vf**)

Policy Loss

train/policy_gradient_loss
tag: train/policy_gradient_loss

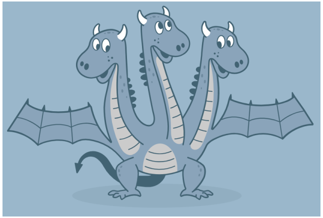


train/clip_fraction
tag: train/clip_fraction



- Early stage:
  - policy loss tends to be more negative (large improvement signals)
  - KL-divergence is less sensitive due to larger entropy
  - consequently, important ratios less affected by clipping
  - $\implies$ less regularized and more natural policy gradient
- Mid to late stage:
  - policy loss magnitude decreases
  - clipping signifies due to decreasing entropy
  - $\implies$ more and more regularized policy updates

Managing the Mess



- Fun fact: writing the code isn't the hardest part
  - Managing your experiments is because:
  - hierarchical configuration is challenging
  - code dependencies are hard to manage
  - managing hyperparameter search is challenging
  - ...
- Suggestion:
  - keep your code base clean (**avoid bugs at any cost**)
  - prefer established code bases over your own
    - it's a good idea to write your own algorithm to learn
    - it's a very bad idea to actually solve a practical problem
  - take leverage of configuration management systems
  - take leverage of plugins like launchers or hyperparameter samplers
  - **hydra** solve the majority of our problems