


PPO, R2D2, Agent57

Milan Straka

 April 22, 2024



EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



unless otherwise stated

Neural networks usually rely on SGD for finding a minimum, by performing

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}).$$

A disadvantage of this approach (so-called **first-order method**) is that we need to specify the learning rates by ourselves, usually using quite a small one, and perform the update many times. However, in some situations, we can do better.

Newton's Root-Finding Method

Assume we have a function $f : \mathbb{R} \rightarrow \mathbb{R}$ and we want to find its root. An SGD-like algorithm would always move “towards” zero by taking small steps.

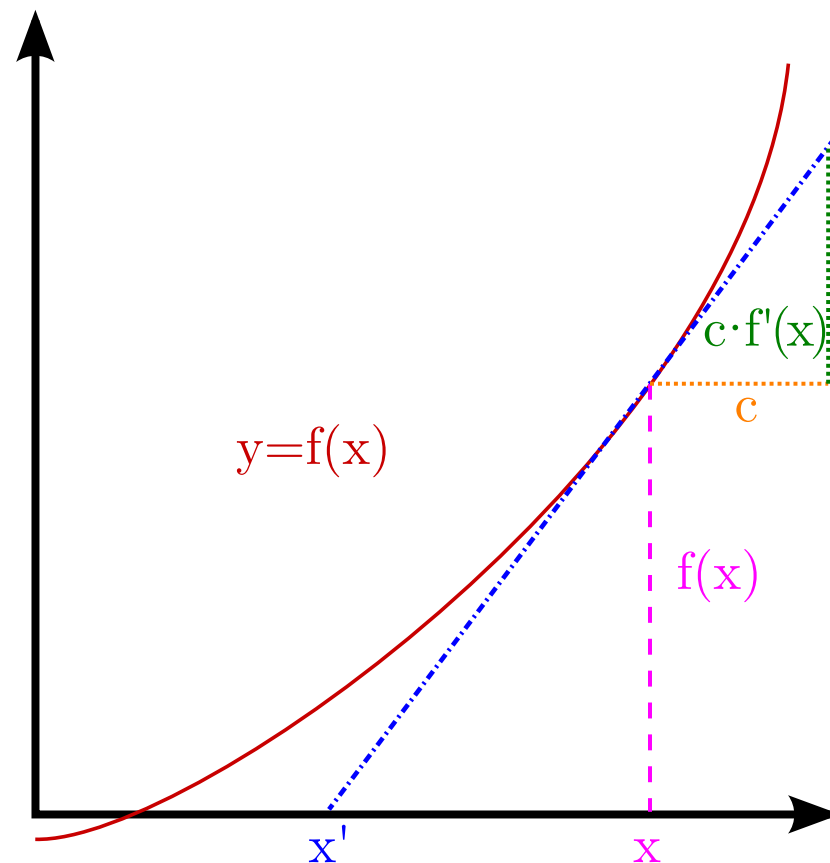
Instead, we could consider the linear local approximation (i.e., consider a line “touching” the function in a given point) and perform a step so that our linear local approximation has a value 0:

$$x' \leftarrow x - \frac{f(x)}{f'(x)}.$$

Finding Minima

The same method can be used to find minima, because a minimum is just a root of a derivative, resulting in:

$$x' \leftarrow x - \frac{f'(x)}{f''(x)}.$$



Modification of https://commons.wikimedia.org/wiki/File:Newton-Raphson_method.svg

The following update is the Newton's method of searching for extremes:

$$x' \leftarrow x - \frac{f'(x)}{f''(x)}.$$

It is a so-called **second-order** method, but it is just an SGD update with a learning rate $\frac{1}{f''(x)}$.

Derivation from Taylor's Expansion

The same update can be derived also from the Taylor's expansion

$$f(x + \varepsilon) \approx f(x) + \varepsilon f'(x) + \frac{1}{2} \varepsilon^2 f''(x) + \mathcal{O}(\varepsilon^3),$$

which we can minimize for ε by

$$0 = \frac{\partial f(x + \varepsilon)}{\partial \varepsilon} \approx f'(x) + \varepsilon f''(x), \text{ obtaining } x + \varepsilon = x - \frac{f'(x)}{f''(x)}.$$

Note that the second-order methods (methods utilizing second derivatives) are impractical when training MLPs with many parameters. The problem is that there are too many second derivatives – if we consider weights $\boldsymbol{\theta} \in \mathbb{R}^D$,

- the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ has D elements;
- however, we have a $D \times D$ matrix with all second derivatives, called the **Hessian** \mathbf{H} :

$$H_{i,j} \stackrel{\text{def}}{=} \frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}.$$

The Taylor expansion of a multivariate function then has the following form:

$$f(\mathbf{x} + \boldsymbol{\varepsilon}) = f(\mathbf{x}) + \boldsymbol{\varepsilon}^T \nabla f(\mathbf{x}) + \frac{1}{2} \boldsymbol{\varepsilon}^T \mathbf{H} \boldsymbol{\varepsilon},$$

from which we obtain the following second-order method update:

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}^{-1} \nabla f(\mathbf{x}).$$

Fisher Information Matrix

Assume we have a model computing a distribution $p(y|\mathbf{x}; \boldsymbol{\theta})$.

We define **score** $s(\boldsymbol{\theta}; \mathbf{x}, y)$ as

$$s(\boldsymbol{\theta}; \mathbf{x}, y) \stackrel{\text{def}}{=} \nabla_{\boldsymbol{\theta}} \log p(y|\mathbf{x}; \boldsymbol{\theta}).$$

Given the formula for a derivative of a logarithm, the score can also be written as

$$s(\boldsymbol{\theta}; \mathbf{x}, y) \stackrel{\text{def}}{=} \frac{\nabla_{\boldsymbol{\theta}} p(y|\mathbf{x}; \boldsymbol{\theta})}{p(y|\mathbf{x}; \boldsymbol{\theta})}.$$

Note that the expectation of the score with respect to the model output y is zero:

$$\mathbb{E}_{y \sim p(\mathbf{x}; \boldsymbol{\theta})} [s(\boldsymbol{\theta}; \mathbf{x}, y)] = \sum_y p(y|\mathbf{x}; \boldsymbol{\theta}) \frac{\nabla_{\boldsymbol{\theta}} p(y|\mathbf{x}; \boldsymbol{\theta})}{p(y|\mathbf{x}; \boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} \sum_y p(y|\mathbf{x}; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} 1 = 0.$$

Fisher Information Matrix

Let \mathcal{D} be the data generating distribution, and \mathbb{D} a dataset sampled from \mathcal{D} .

Assuming $\boldsymbol{\theta} \in \mathbb{R}^D$, we define the **Fisher Information Matrix** $\mathbf{F}(\boldsymbol{\theta}) \in \mathbb{R}^{D \times D}$ as the covariance matrix of the score:

$$\mathbf{F}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{y|\mathbf{x} \sim \mathcal{D}} \left[\left(s(\boldsymbol{\theta}; \mathbf{x}, y) - \mathbb{E} s(\boldsymbol{\theta}; \mathbf{x}, y) \right) \left(s(\boldsymbol{\theta}; \mathbf{x}, y) - \mathbb{E} s(\boldsymbol{\theta}; \mathbf{x}, y) \right)^T \right].$$

Because the expectation of the score is zero, the definition simplifies to

$$\mathbf{F}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{y|\mathbf{x} \sim \mathcal{D}} \left[s(\boldsymbol{\theta}; \mathbf{x}, y) s(\boldsymbol{\theta}; \mathbf{x}, y)^T \right].$$

The first expectation is usually computed over \mathbb{D} . When the $y|\mathbf{x}$ is taken from $p(y|\mathbf{x}; \boldsymbol{\theta})$, the matrix is called **Empirical Fisher**, and it can be computed with a single forward and backward pass through the model.

Now consider the usual NLL loss $\mathcal{L}(y|\mathbf{x}; \boldsymbol{\theta}) = -\log p(y|\mathbf{x}; \boldsymbol{\theta})$. Its Hessian is

$$H_{\mathcal{L}(y|\mathbf{x}; \boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} - \log p(y|\mathbf{x}; \boldsymbol{\theta}).$$

We get that

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{y|\mathbf{x} \sim \mathcal{D}} H_{\mathcal{L}(y|\mathbf{x}; \boldsymbol{\theta})} = \mathbf{F}(\boldsymbol{\theta}).$$

Lets consider

$$D_{\text{KL}}(p(y|\mathbf{x}; \boldsymbol{\theta}) \| p(y|\mathbf{x}; \boldsymbol{\theta} + \mathbf{d}))$$

The expectation over $\mathbf{x} \sim \mathcal{D}$ of the Taylor expansion to the second order gives

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} D_{\text{KL}}(p(y|\mathbf{x}; \boldsymbol{\theta}) \| p(y|\mathbf{x}; \boldsymbol{\theta} + \mathbf{d})) \approx \frac{1}{2} \mathbf{d}^T \mathbf{F} \mathbf{d} + \mathcal{O}(\|\mathbf{d}\|^3).$$

Consider the

$$\lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \arg \min_{\mathbf{d}, \|\mathbf{d}\| \leq \varepsilon} \mathcal{L}(\mathbb{D}; \boldsymbol{\theta} + \mathbf{d}).$$

We get that

$$\mathbf{d} \propto -\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbb{D}; \boldsymbol{\theta}),$$

therefore,

$$\mathbf{d} = \varepsilon \frac{-\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbb{D}; \boldsymbol{\theta})}{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbb{D}; \boldsymbol{\theta})\|}.$$

Consider now

$$\arg \min_{\mathbf{d}, D_{\text{KL}}(p(y|\mathbf{x};\boldsymbol{\theta}) \| p(y|\mathbf{x};\boldsymbol{\theta} + \mathbf{d})) \leq \varepsilon} \mathcal{L}(\mathbb{D}; \boldsymbol{\theta} + \mathbf{d}).$$

We get that

$$\mathbf{d} \propto -\mathbf{F}(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbb{D}; \boldsymbol{\theta}).$$

Note that if we consider just a diagonal of $\mathbf{F}(\boldsymbol{\theta})^{-1}$, the resulting algorithm is similar to Adam.

Natural Policy Gradient

Kakade (2002) introduced natural policy gradient, a second-order method utilizing the Fisher Information Matrix.

Using policy gradient theorem, we are able to compute ∇v_π . Normally, we update the parameters by using directly this gradient. This choice is justified by the fact that a vector \mathbf{d} which maximizes $v_\pi(\mathbf{s}; \boldsymbol{\theta} + \mathbf{d})$ under the constraint that $\|\mathbf{d}\|^2$ is bounded by a small constant is exactly the gradient ∇v_π .

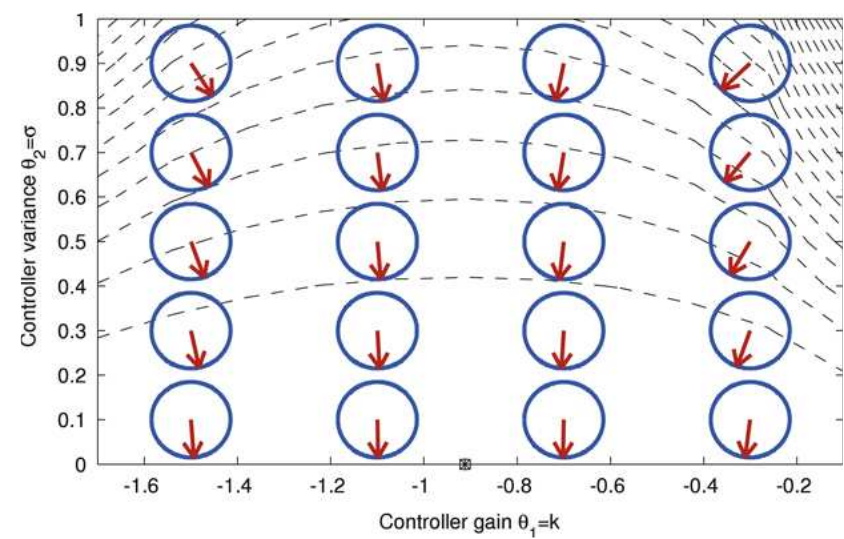
However, for the Fisher information matrix

$$\mathbf{F}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_s \mathbb{E}_{\pi(a|s;\boldsymbol{\theta})} \left[(\nabla_{\boldsymbol{\theta}} \log \pi(a|s;\boldsymbol{\theta})) (\nabla_{\boldsymbol{\theta}} \log \pi(a|s;\boldsymbol{\theta}))^T \right],$$

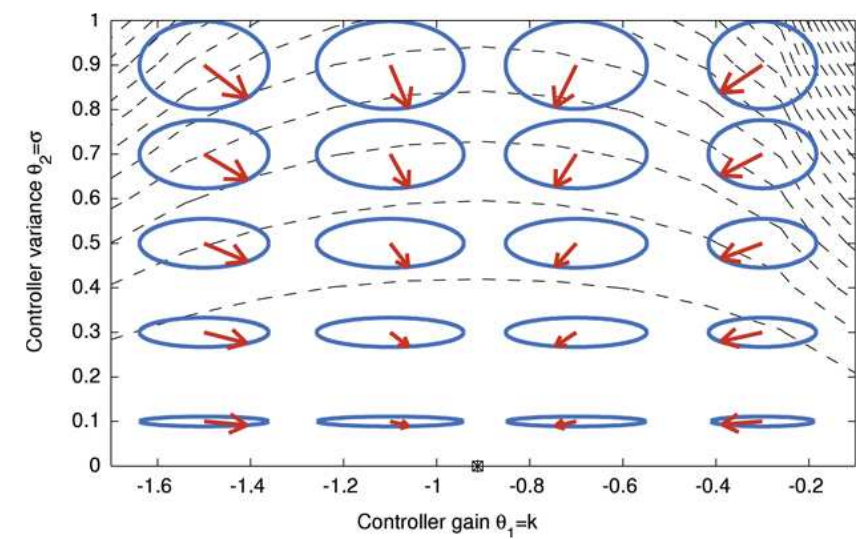
we might to update the parameters using $\mathbf{d}_F \stackrel{\text{def}}{=} \mathbf{F}(\boldsymbol{\theta})^{-1} \nabla v_\pi$.

It can be shown that the Fisher information metric is the only Riemannian metric (up to rescaling) invariant to change of parameters under sufficient statistic.

Natural Policy Gradient



(a) Vanilla policy gradient.



(b) Natural policy gradient.

Figure 3 of "Reinforcement learning of motor skills with policy gradients" by Jan Peters et al.

An interesting property of using the \mathbf{d}_F to update the parameters is that

- updating θ using ∇v_π will choose an arbitrary *better* action in state s ;
- updating θ using $\mathbf{F}(\theta)^{-1} \nabla v_\pi$ chooses the *best* action (maximizing expected return), similarly to tabular greedy policy improvement.

However, computing \mathbf{d}_F in a straightforward way is too costly.

Duan et al. (2016) in paper *Benchmarking Deep Reinforcement Learning for Continuous Control* propose a modification to the NPG to efficiently compute \mathbf{d}_F .

Following Schulman et al. (2015), they suggest to use *conjugate gradient algorithm*, which can solve a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ in an iterative manner, by using \mathbf{A} only to compute products $\mathbf{A}\mathbf{v}$ for a suitable \mathbf{v} .

Therefore, \mathbf{d}_F is found as a solution of

$$\mathbf{F}(\boldsymbol{\theta})\mathbf{d}_F = \nabla v_\pi$$

and using only 10 iterations of the algorithm seem to suffice according to the experiments.

Furthermore, Duan et al. suggest to use a specific learning rate suggested by Peters et al (2008) of

$$\frac{\alpha}{\sqrt{(\nabla v_\pi)^T \mathbf{F}(\boldsymbol{\theta})^{-1} \nabla v_\pi}}.$$

Schulman et al. in 2015 wrote an influential paper introducing TRPO as an improved variant of NPG.

Considering two policies $\pi, \tilde{\pi}$, we can write

$$v_{\tilde{\pi}} = v_{\pi} + \mathbb{E}_{s \sim \mu(\tilde{\pi})} \mathbb{E}_{a \sim \tilde{\pi}(a|s)} a_{\pi}(a|s),$$

where $a_{\pi}(a|s)$ is the advantage function $q_{\pi}(a|s) - v_{\pi}(s)$ and $\mu(\tilde{\pi})$ is the on-policy distribution of the policy $\tilde{\pi}$.

Analogously to policy improvement, we see that if $a_{\pi}(a|s) \geq 0$, policy $\tilde{\pi}$ performance increases (or stays the same if the advantages are zero everywhere).

However, sampling states $s \sim \mu(\tilde{\pi})$ is costly. Therefore, we instead consider

$$L_{\pi}(\tilde{\pi}) = v_{\pi} + \mathbb{E}_{s \sim \mu(\pi)} \mathbb{E}_{a \sim \tilde{\pi}(a|s)} a_{\pi}(a|s).$$

$$L_{\pi}(\tilde{\pi}) = v_{\pi} + \mathbb{E}_{s \sim \mu(\pi)} \mathbb{E}_{a \sim \tilde{\pi}(a|s)} a_{\pi}(a|s)$$

Using $L_{\pi}(\tilde{\pi})$ is usually justified by $L_{\pi}(\pi) = v_{\pi}$ and $\nabla_{\tilde{\pi}} L_{\pi}(\tilde{\pi})|_{\tilde{\pi}=\pi} = \nabla_{\tilde{\pi}} v_{\tilde{\pi}}|_{\tilde{\pi}=\pi}$.

Schulman et al. additionally proves that if we denote $\alpha = D_{\text{KL}}^{\max}(\pi_{\text{old}} \parallel \pi_{\text{new}}) = \max_s D_{\text{KL}}(\pi_{\text{old}}(\cdot|s) \parallel \pi_{\text{new}}(\cdot|s))$, then

$$v_{\pi_{\text{new}}} \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2} \alpha \quad \text{where} \quad \varepsilon = \max_{s,a} |a_{\pi}(s,a)|.$$

Therefore, TRPO maximizes $L_{\pi_{\theta_0}}(\pi_{\theta})$ subject to $D_{\text{KL}}^{\theta_0}(\pi_{\theta_0} \parallel \pi_{\theta}) < \delta$, where

- $D_{\text{KL}}^{\theta_0}(\pi_{\theta_0} \parallel \pi_{\theta}) = \mathbb{E}_{s \sim \mu(\pi_{\theta_0})} [D_{\text{KL}}(\pi_{\text{old}}(\cdot|s) \parallel \pi_{\text{new}}(\cdot|s))]$ is used instead of D_{KL}^{\max} for performance reasons;
- δ is a constant found empirically, as the one implied by the above equation is too small;
- importance sampling is used to account for sampling actions from π .

$$\text{maximize } L_{\pi_{\theta_0}}(\pi_{\theta}) = \mathbb{E}_{s \sim \mu(\pi_{\theta_0}), a \sim \pi_{\theta_0}(a|s)} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_0}(a|s)} a_{\pi_{\theta_0}}(a|s) \right] \text{ subject to } D_{\text{KL}}^{\theta_0}(\pi_{\theta_0} \parallel \pi_{\theta}) < \delta$$

The parameters are updated using $\mathbf{d}_F = \mathbf{F}(\boldsymbol{\theta})^{-1} \nabla L_{\pi_{\theta_0}}(\pi_{\theta})$, utilizing the conjugate gradient algorithm as described earlier for TNPG (note that the algorithm was designed originally for TRPO and only later employed for TNPG).

To guarantee improvement and respect the D_{KL} constraint, a line search is in fact performed.

We start by the learning rate of $\sqrt{\delta / (\mathbf{d}_F^T \mathbf{F}(\boldsymbol{\theta})^{-1} \mathbf{d}_F)}$ and shrink it exponentially until the constraint is satisfied and the objective improves.

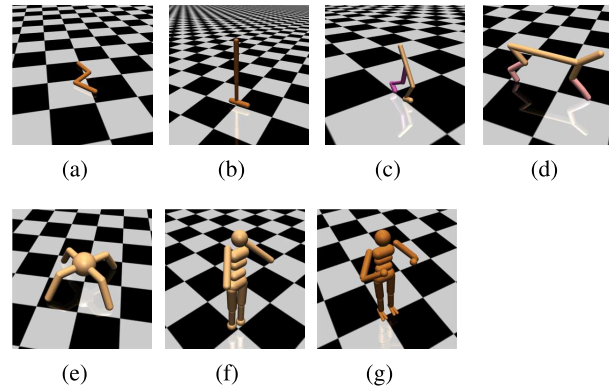


Figure 1. Illustration of locomotion tasks: (a) Swimmer; (b) Hopper; (c) Walker; (d) Half-Cheetah; (e) Ant; (f) Simple Humanoid; and (g) Full Humanoid.

Figure 1 of "Benchmarking Deep Reinforcement Learning for Continuous Control" by Duan et al.

Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	3986.4 ± 748.9	4861.5 ± 12.3	565.6 ± 137.6	4869.8 ± 37.6	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	-153.4 ± 0.2	13.4 ± 18.0	209.7 ± 55.5	84.7 ± 13.8	-113.3 ± 4.6	247.2 ± 76.1	38.2 ± 25.7	-40.1 ± 5.7	40.0 ± 244.6
Mountain Car	-415.4 ± 0.0	-67.1 ± 1.0	-66.5 ± 4.5	-79.4 ± 1.1	-275.6 ± 166.3	-61.7 ± 0.9	-66.0 ± 2.4	-85.0 ± 7.7	-288.4 ± 170.3
Acrobot	-1904.5 ± 1.0	-508.1 ± 91.0	-395.8 ± 121.2	-352.7 ± 35.9	-1001.5 ± 10.8	-326.0 ± 24.4	-436.8 ± 14.7	-785.6 ± 13.1	-223.6 ± 5.8
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	4455.4 ± 37.6	3614.8 ± 368.1	446.7 ± 114.8	4412.4 ± 50.4	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	-1.7 ± 0.1	92.3 ± 0.1	96.0 ± 0.2	60.7 ± 5.5	3.8 ± 3.3	96.0 ± 0.2	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	1155.1 ± 57.9	553.2 ± 71.0	86.7 ± 17.6	1183.3 ± 150.0	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	-1.7 ± 0.0	506.5 ± 78.8	1382.6 ± 108.2	136.0 ± 15.9	-37.0 ± 38.1	1353.8 ± 85.0	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	-90.8 ± 0.3	1183.1 ± 69.2	1729.5 ± 184.6	376.1 ± 28.2	34.5 ± 38.0	1914.0 ± 120.1	330.4 ± 274.8	441.3 ± 107.6	2148.6 ± 702.7
Ant*	13.4 ± 0.7	548.3 ± 55.5	706.0 ± 127.7	37.6 ± 3.1	39.0 ± 9.8	730.2 ± 61.3	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	255.0 ± 24.5	93.3 ± 17.4	28.3 ± 4.7	269.7 ± 40.3	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	288.4 ± 25.2	46.7 ± 5.6	41.7 ± 6.1	287.0 ± 23.4	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2

Table 1 of "Benchmarking Deep Reinforcement Learning for Continuous Control" by Duan et al.

Proximal Policy Optimization

A simplification of TRPO which can be implemented using a few lines of code.

Let $r_t(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \frac{\pi(A_t|S_t;\boldsymbol{\theta})}{\pi(A_t|S_t;\boldsymbol{\theta}_{\text{old}})}$. PPO maximizes the objective (i.e., you should minimize its negation)

$$L^{\text{CLIP}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_t \left[\min \left(r_t(\boldsymbol{\theta}) \hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right].$$

Such a $L^{\text{CLIP}}(\boldsymbol{\theta})$ is a lower (pessimistic) bound.

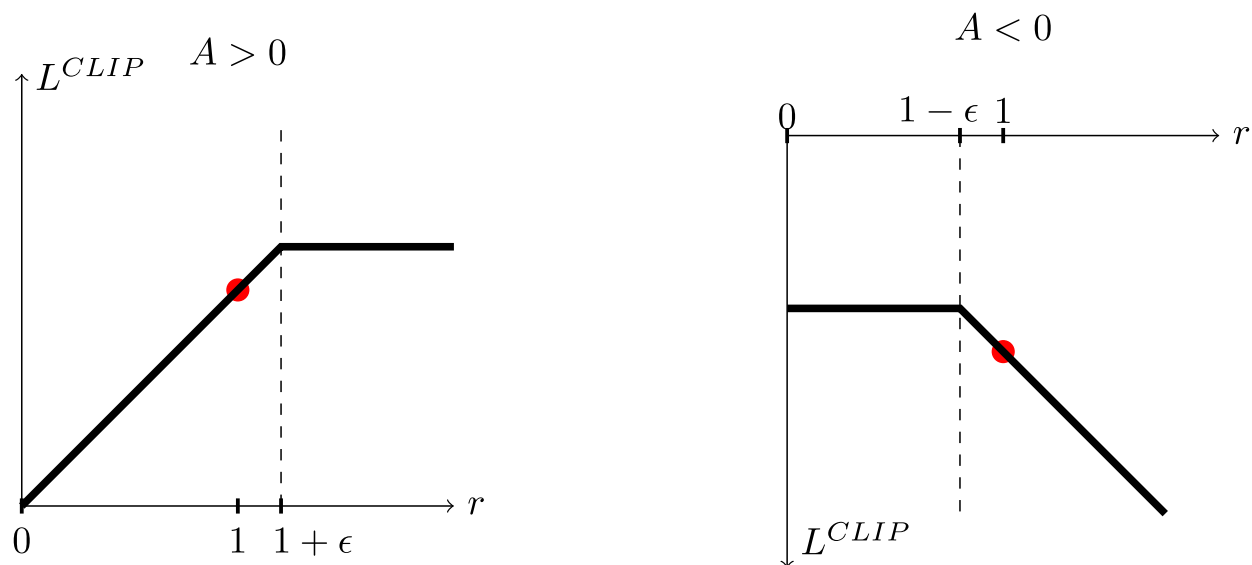


Figure 1 of "Proximal Policy Optimization Algorithms" by Schulman et al.

Proximal Policy Optimization

The advantages \hat{A}_t are additionally estimated using the so-called *generalized advantage estimation*, which is just an analogue of the truncated n-step lambda-return:

$$\hat{A}_t = \sum_{i=0}^{n-1} \gamma^i \lambda^i (R_{t+1+i} + \gamma V(S_{t+i+1}) - V(S_{t+i})).$$

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  $\hat{A}_1, \dots, \hat{A}_T$ 
  end for
  Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
   $\theta_{\text{old}} \leftarrow \theta$ 
end for
```

Algorithm 1 of "Proximal Policy Optimization Algorithms" by Schulman et al.

- The rollout phase should be usually performed using vectorized environments.
- It is important to correctly handle episodes that did not finish in a rollout, using bootstrapping to estimate the return from the rest of the episode. That way, PPO can learn in long-horizon games with T much smaller than episode length.
- Increasing N increases parallelism, while increasing T increase the number of steps that must be performed sequentially.

Proximal Policy Optimization

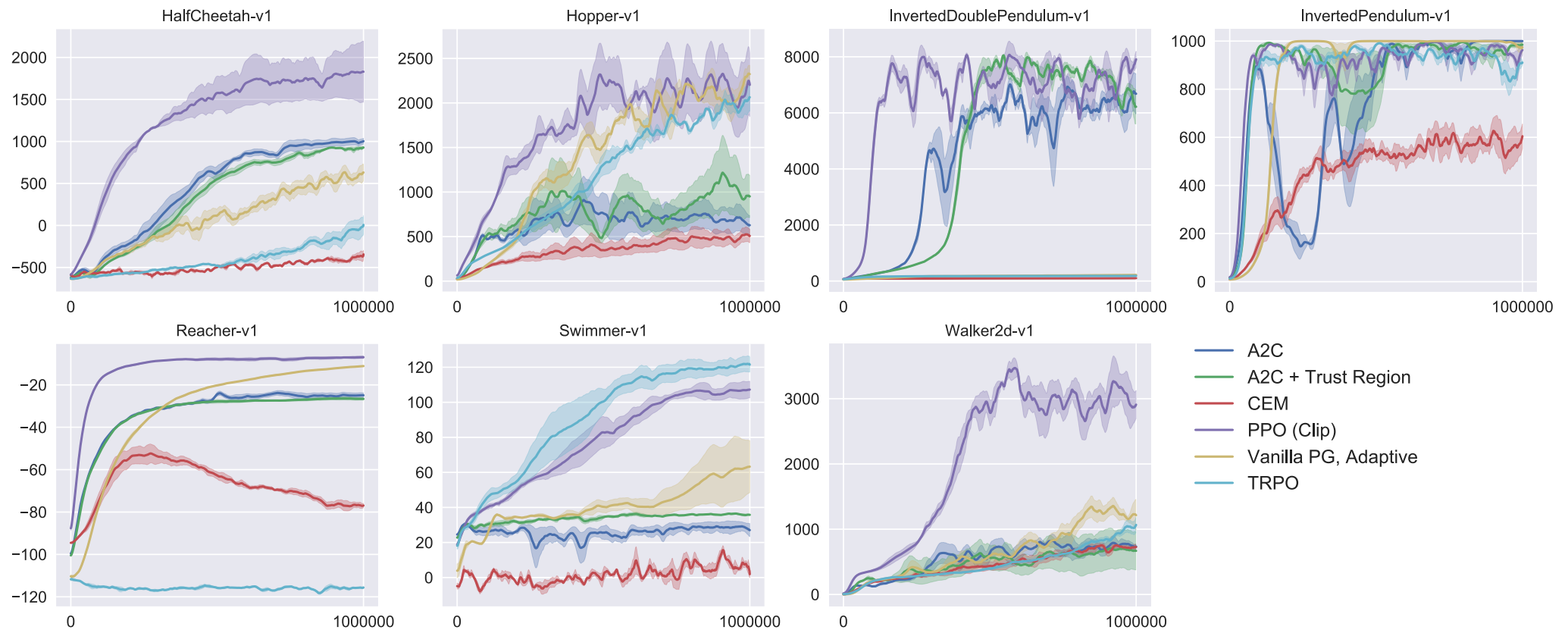


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

Figure 3 of "Proximal Policy Optimization Algorithms" by Schulman et al.

Results from the SAC paper:

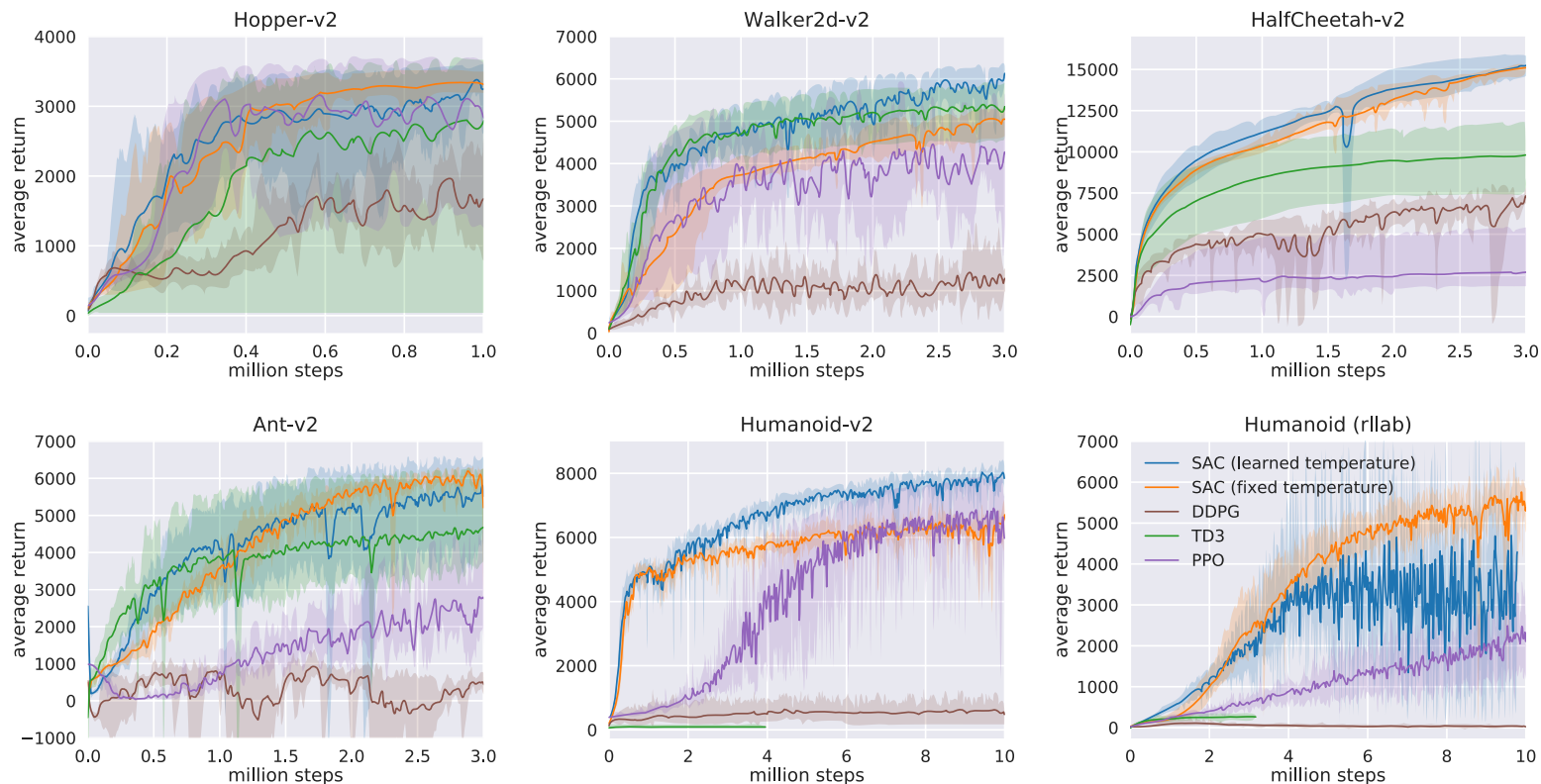


Figure 1: Training curves on continuous control benchmarks. Soft actor-critic (blue and yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

Figure 1 of "Soft Actor-Critic Algorithms and Applications" by Tuomas Haarnoja et al.

- There are a few tricks that influence the performance of PPO significantly; see the following nice blogpost about many of them:

<https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>

- The paper *What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study* <https://openreview.net/forum?id=nIAxjsniDzg> performs an evaluation of many hyperparameters of the PPO algorithm.

Main takeaways:

- Start with clipping threshold 0.25, but try increasing/decreasing it.
- Initialization of the last policy layer influences the results considerably; recommendation is to use 100 times smaller weights.
- Use softplus to parametrize standard deviation of actions, use a negative offset to decrease initial standard deviation of actions, tune it if possible.
- Use \tanh to transform the action distribution.
- Do not share weights between the policy and value network; use a wide value network.
- Always normalize observations; check if normalizing value function helps.
- Use GAE with $\lambda = 0.9$, do not use Huber loss. Adam with $3e-4$ is a safe choice.
- Perform multiple passes over the data, recompute advantages at the beginning of every one of them.
- The discount factor γ is important, tune it per environment starting with $\gamma = 0.99$.

Transformed Rewards

So far, we have clipped the rewards in DQN on Atari environments.

Consider a Bellman operator \mathcal{T}

$$(\mathcal{T}q)(s, a) \stackrel{\text{def}}{=} \mathbb{E}_{s', r \sim p} \left[r + \gamma \max_{a'} q(s', a') \right].$$

Instead of clipping the magnitude of rewards, we might use a function $h : \mathbb{R} \rightarrow \mathbb{R}$ to reduce their scale. We define a transformed Bellman operator \mathcal{T}_h as

$$(\mathcal{T}_h q)(s, a) \stackrel{\text{def}}{=} \mathbb{E}_{s', r \sim p} \left[h \left(r + \gamma \max_{a'} h^{-1} (q(s', a')) \right) \right].$$

It is easy to prove the following two propositions from a 2018 paper *Observe and Look Further: Achieving Consistent Performance on Atari* by Tobias Pohlen et al.

1. If $h(z) = \alpha z$ for $\alpha > 0$, then $\mathcal{T}_h^k q \xrightarrow{k \rightarrow \infty} h \circ q_* = \alpha q_*$.

The statement follows from the fact that it is equivalent to scaling the rewards by a constant α .

2. When h is strictly monotonically increasing and the MDP is deterministic, then $\mathcal{T}_h^k q \xrightarrow{k \rightarrow \infty} h \circ q_*$.

This second proposition follows from

$$h \circ q_* = h \circ \mathcal{T}q_* = h \circ \mathcal{T}(h^{-1} \circ h \circ q_*) = \mathcal{T}_h(h \circ q_*),$$

where the last equality only holds if the MDP is deterministic.

For stochastic MDP, the authors prove that if h is strictly monotonically increasing, Lipschitz continuous with Lipschitz constant L_h , and has a Lipschitz continuous inverse with Lipschitz constant $L_{h^{-1}}$, then for $\gamma < \frac{1}{L_h L_{h^{-1}}}$, \mathcal{T}_h is again a contraction. (Proof in Proposition A.1.)

For the Atari environments, the authors propose the transformation

$$h(x) \stackrel{\text{def}}{=} \text{sign}(x) \left(\sqrt{|x| + 1} - 1 \right) + \varepsilon x$$

with $\varepsilon = 10^{-2}$. The additive regularization term ensures that h^{-1} is Lipschitz continuous.

It is straightforward to verify that

$$h^{-1}(x) = \text{sign}(x) \left(\left(\frac{\sqrt{1 + 4\varepsilon(|x| + 1 + \varepsilon)} - 1}{2\varepsilon} \right)^2 - 1 \right).$$

In practice, discount factor larger than $\frac{1}{L_h L_{h^{-1}}}$ is being used – however, it seems to work.

Recurrent Replay Distributed DQN (R2D2)

Proposed in 2019, to study the effects of recurrent state, experience replay and distributed training.

R2D2 utilizes prioritized replay, n -step double Q-learning with $n = 5$, convolutional layers followed by a 512-dimensional LSTM passed to duelling architecture, generating experience by a large number of actors (256; each performing approximately 260 steps per second) and learning from batches by a single learner (achieving 5 updates per second using mini-batches of 64 sequences of length 80).

Rewards are transformed instead of clipped, and no loss-of-life-as-episode-end heuristic is used. Instead of individual transitions, the replay buffer consists of fixed-length ($m = 80$) sequences of (s, a, r) , with adjacent sequences overlapping by 40 time steps.

The prioritized replay employs a combination of the maximum and the average absolute 5-step TD errors δ_i over the sequence: $p = \eta \max_i \delta_i + (1 - \eta) \bar{\delta}$, for both η and the priority exponent set to 0.9.

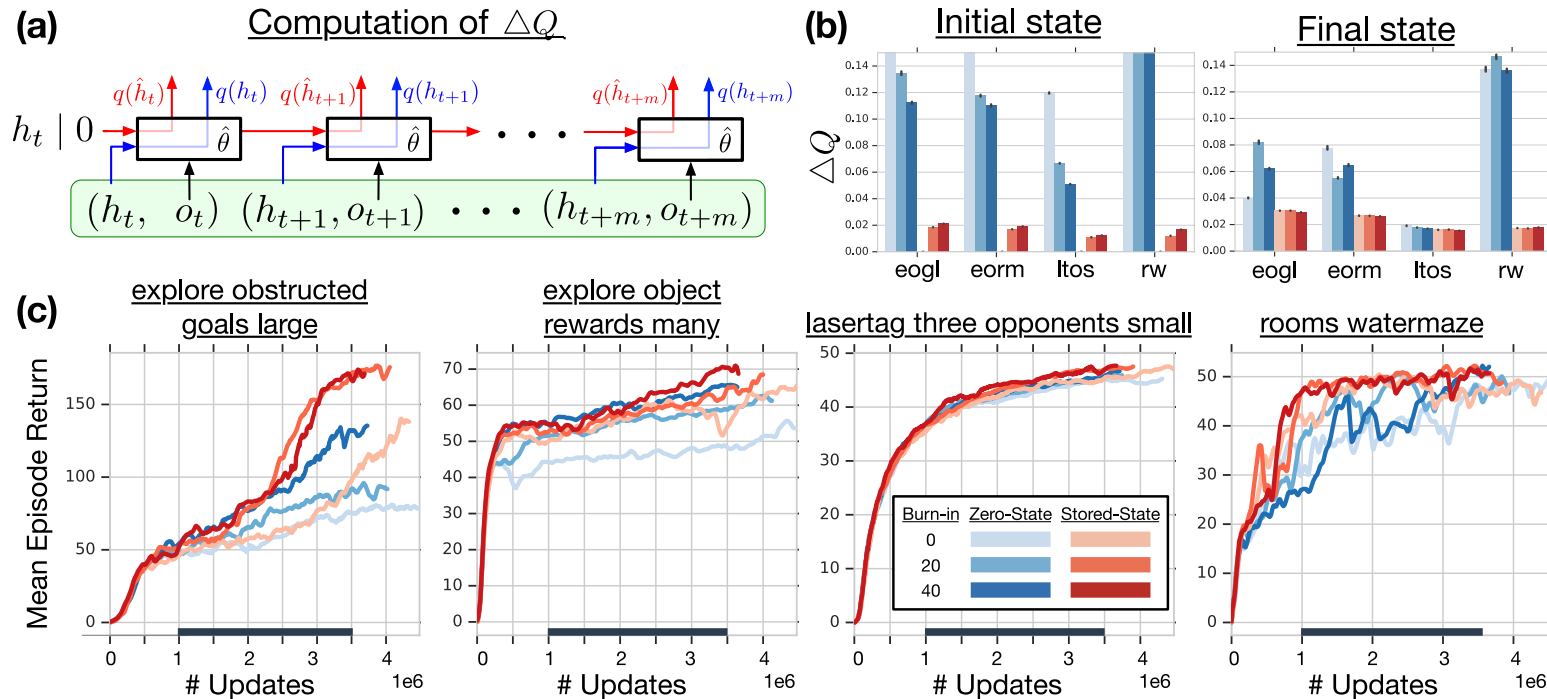


Figure 1: Top row shows Q-value discrepancy ΔQ as a measure for recurrent state staleness. **(a)** Diagram of how ΔQ is computed, with green box indicating a whole sequence sampled from replay. For simplicity, $l = 0$ (no burn-in). **(b)** ΔQ measured at first state and last state of replay sequences, for agents training on a selection of DMLab levels (indicated by initials) with different training strategies. Bars are averages over seeds and through time indicated by bold line on x-axis in bottom row. **(c)** Learning curves on the same levels, varying the training strategy, and averaged over 3 seeds.

Figure 1 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Recurrent Replay Distributed DQN (R2D2)

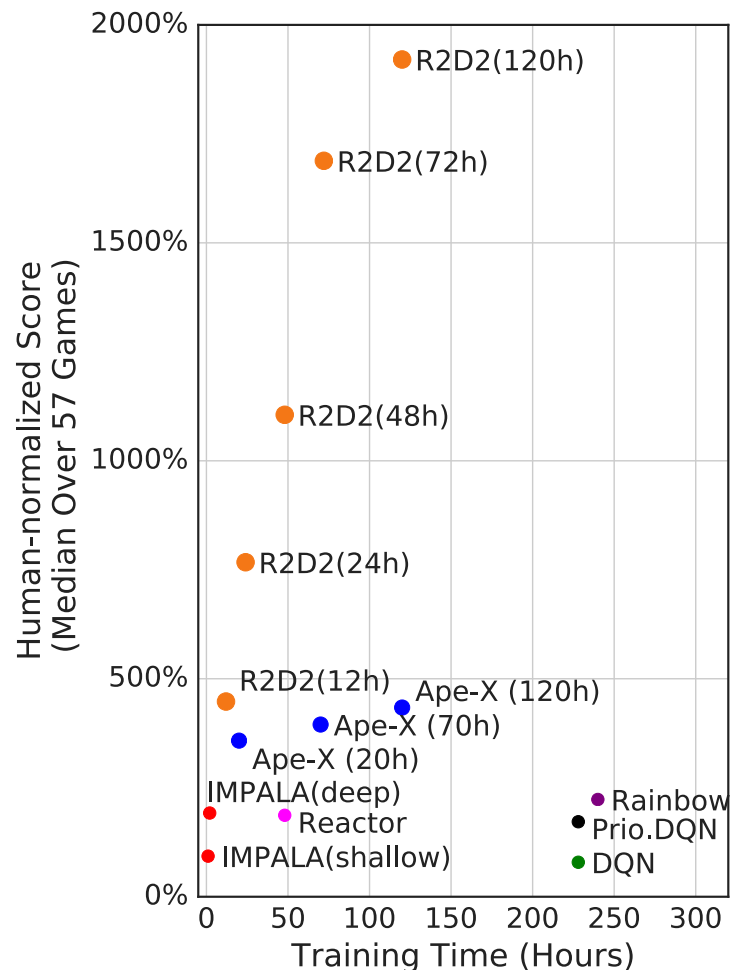


Figure 2 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Human-Normalized Score	Atari-57		DMLab-30	
	Median	Mean	Median	Mean-Capped
Ape-X (Horgan et al., 2018)	434.1%	1695.6%	–	–
Reactor (Gruslys et al., 2018)	187.0%	–	–	–
IMPALA, deep (Espeholt et al., 2018)	191.8%	957.6%	49.0%	45.8%
IMPALA, shallow (re-run)	–	–	89.7%	73.6%
IMPALA, deep (re-run)	–	–	107.5%	85.1%
R2D2+	–	–	99.5%	85.7%
R2D2, feed-forward	589.2%	1974.4%	–	–
R2D2	1920.6%	4024.9%	96.9%	78.3%

Table 1 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Number of actors	256
Actor parameter update interval	400 environment steps
Sequence length m	80 (+ prefix of $l = 40$ in burn-in experiments)
Replay buffer size	4×10^6 observations (10^5 part-overlapping sequences)
Priority exponent	0.9
Importance sampling exponent	0.6
Discount γ	0.997
Minibatch size	64 (32 for R2D2+ on DMLab)
Optimizer	Adam (Kingma & Ba, 2014)
Optimizer settings	learning rate = 10^{-4} , $\epsilon = 10^{-3}$
Target network update interval	2500 updates
Value function rescaling	$h(x) = \text{sign}(x)(\sqrt{ x + 1} - 1) + \epsilon x$, $\epsilon = 10^{-3}$

Table 2: Hyper-parameters values used in R2D2. All missing parameters follow the ones in Ape-X (Horgan et al., 2018).

Table 2 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Atari-57 - Human-normalized Median

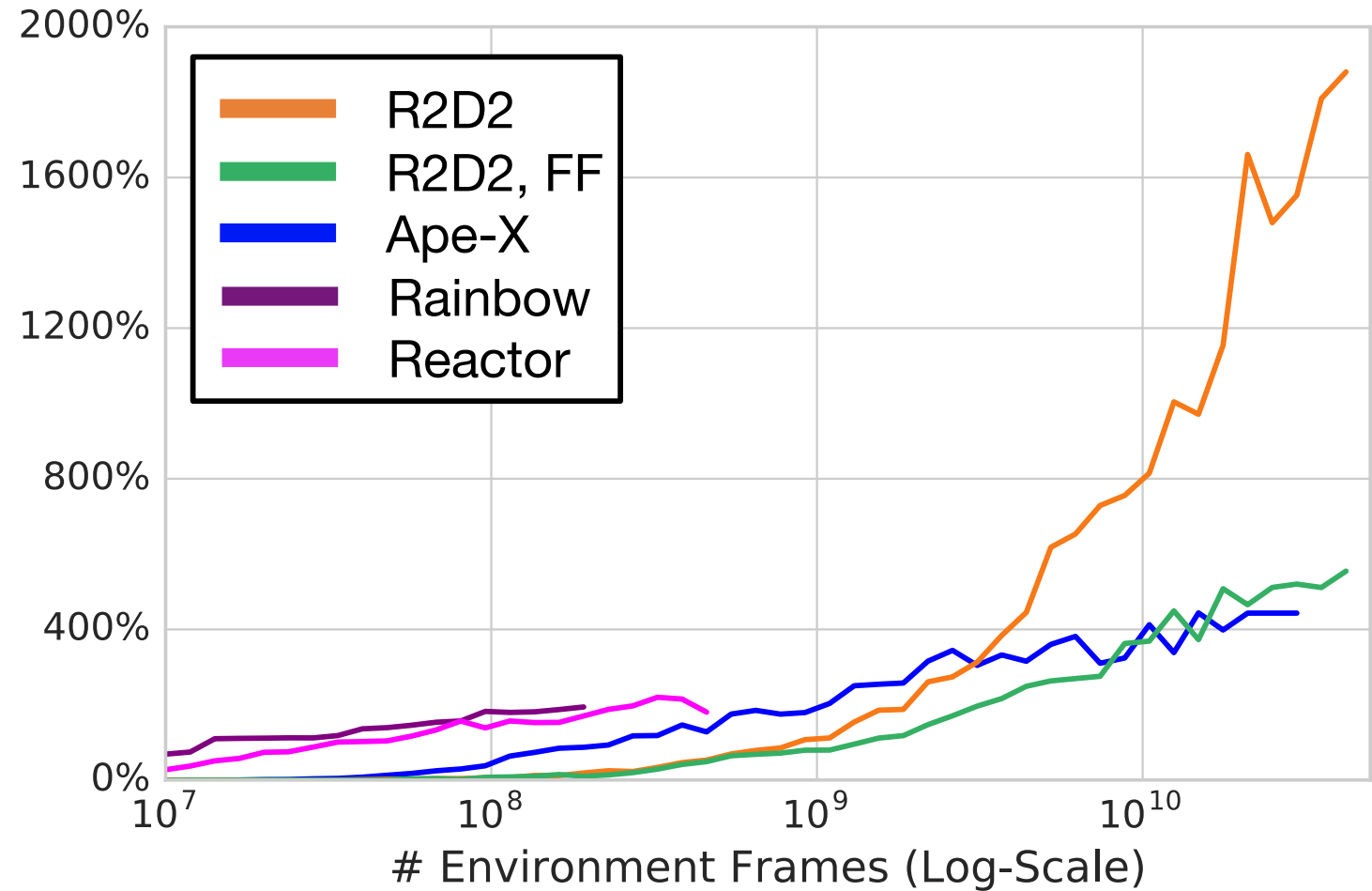


Figure 9 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Recurrent Replay Distributed DQN (R2D2)

Ablations comparing the reward clipping instead of value rescaling (**Clipped**), smaller discount factor $\gamma = 0.99$ (**Discount**) and **Feed-Forward** variant of R2D2. Furthermore, life-loss **reset** evaluates resetting an episode on life loss, with **roll** preventing value bootstrapping (but not LSTM unrolling).

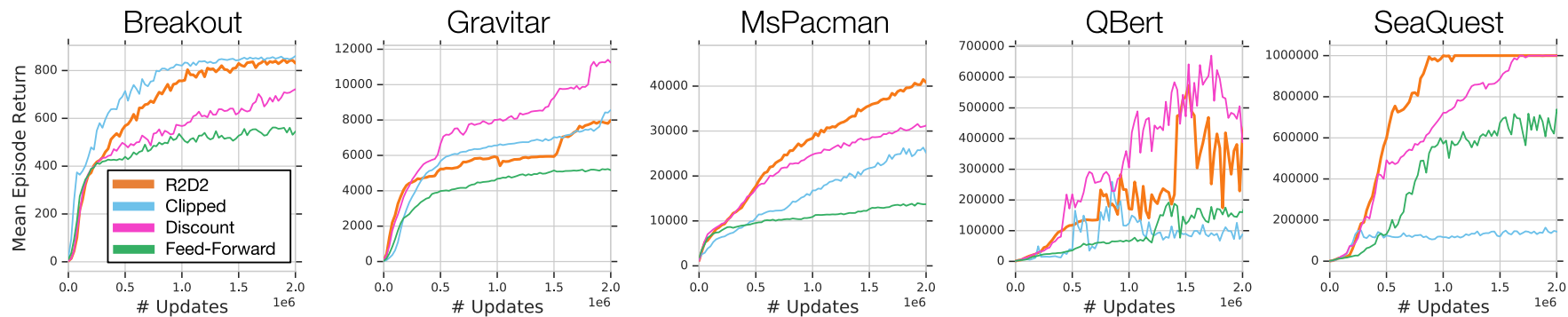


Figure 4 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

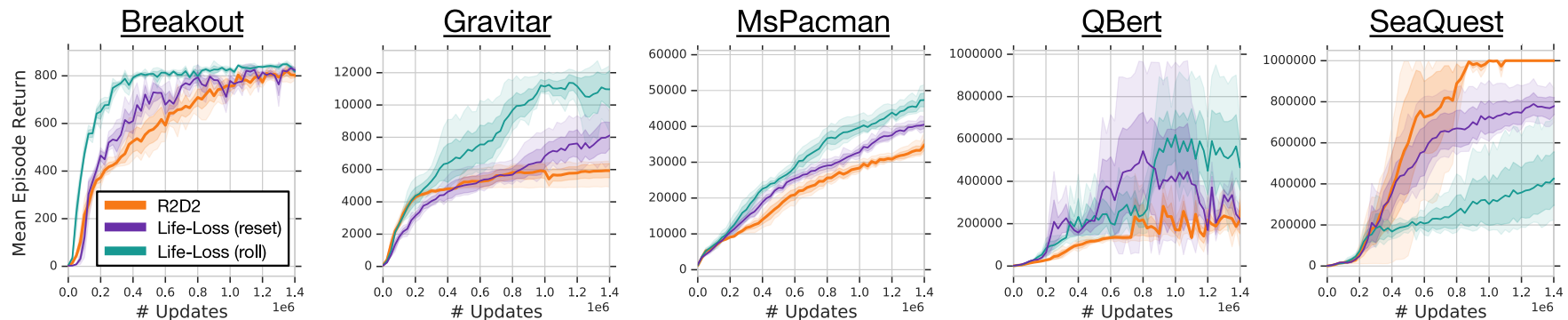


Figure 7 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Utilization of LSTM Memory During Inference

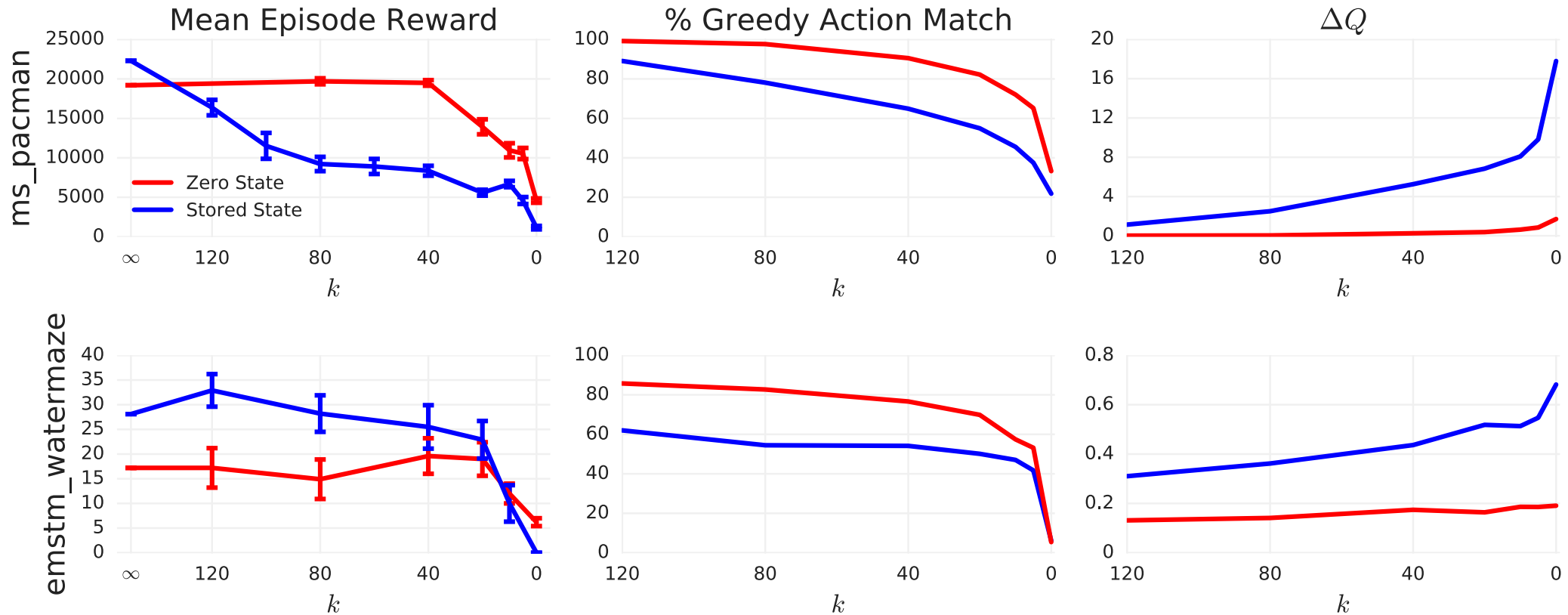


Figure 5 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

The Agent57 is an agent (from Mar 2020) capable of outperforming the standard human benchmark on all 57 games.

Its most important components are:

- Retrace; from *Safe and Efficient Off-Policy Reinforcement Learning* by Munos et al., <https://arxiv.org/abs/1606.02647>,
- Never give up strategy; from *Never Give Up: Learning Directed Exploration Strategies* by Badia et al., <https://arxiv.org/abs/2002.06038>,
- Agent57 itself; from *Agent57: Outperforming the Atari Human Benchmark* by Badia et al., <https://arxiv.org/abs/2003.13350>.

$$\mathcal{R}q(s, a) \stackrel{\text{def}}{=} q(s, a) + \mathbb{E}_b \left[\sum_{t \geq 0} \gamma^t \left(\prod_{j=1}^t c_t \right) \left(R_{t+1} + \gamma \mathbb{E}_{A_{t+1} \sim \pi} q(S_{t+1}, A_{t+1}) - q(S_t, A_t) \right) \right],$$

where there are several possibilities for defining the traces c_t :

- **importance sampling**, $c_t = \rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$,
 - the usual off-policy correction, but with possibly very high variance,
 - note that $c_t = 1$ in the on-policy case;
- **Tree-backup TB(λ)**, $c_t = \lambda \pi(A_t|S_t)$,
 - the Tree-backup algorithm extended with traces,
 - however, c_t can be much smaller than 1 in the on-policy case;
- **Retrace(λ)**, $c_t = \lambda \min \left(1, \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \right)$,
 - off-policy correction with limited variance, with $c_t = 1$ in the on-policy case.

The authors prove that \mathcal{R} has a unique fixed point q_π for any $0 \leq c_t \leq \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$.

Never Give Up

The NGU (Never Give Up) agent performs *curiosity-driver exploration*, and augment the extrinsic (MDP) rewards with an intrinsic reward. The augmented reward at time t is then $r_t^\beta \stackrel{\text{def}}{=} r_t^e + \beta r_t^i$, with β a scalar weight of the intrinsic reward.

The intrinsic reward fulfills three goals:

1. quickly discourage visits of the same state in the same episode;
2. slowly discourage visits of the states visited many times in all episodes;
3. ignore the parts of the state not influenced by the agent's actions.

The intrinsic rewards is a combination of the episodic novelty r_t^{episodic} and life-long novelty α_t :

$$r_t^i \stackrel{\text{def}}{=} r_t^{\text{episodic}} \cdot \text{clip} \left(1 \leq \alpha_t \leq L = 5 \right).$$

Never Give Up

The episodic novelty works by storing the embedded states $f(S_t)$ visited during the episode in episodic memory M .

The r_t^{episodic} is then estimated as

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{\text{visit count of } f(S_t)}}$$

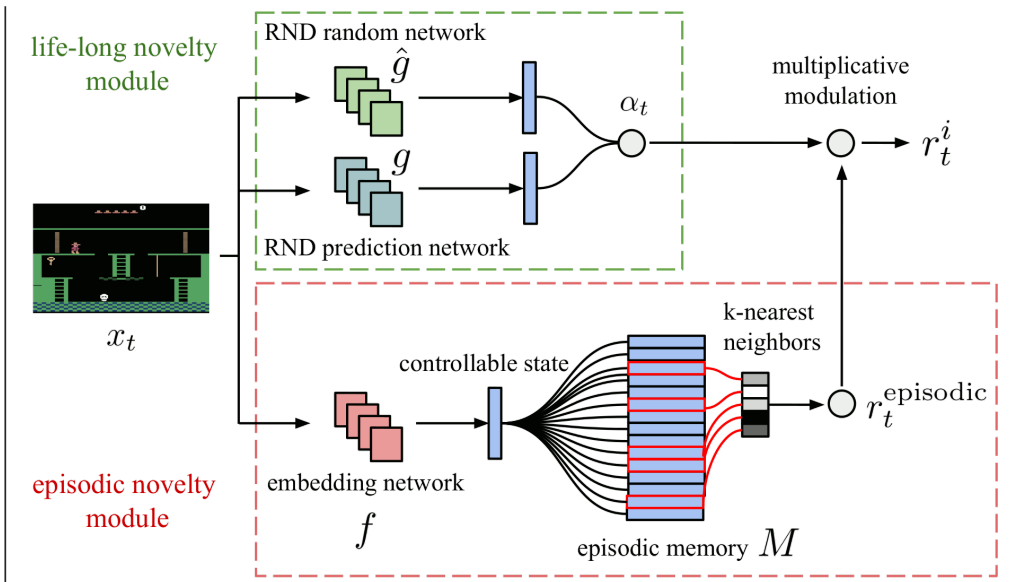
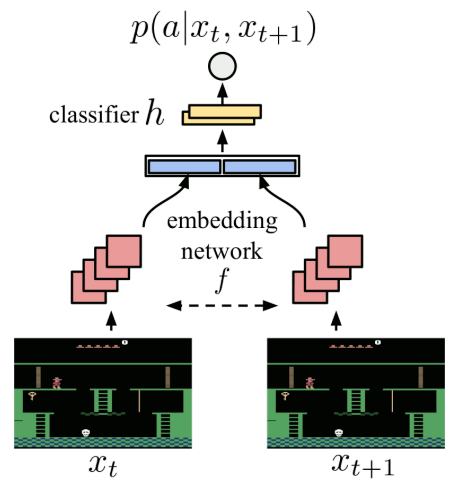


Figure 1 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

The visit count is estimated using similarities of k -nearest neighbors of $f(S_t)$ measured via an inverse kernel $K(x, z) = \frac{\epsilon}{\frac{d(x,z)^2}{d_m^2} + \epsilon}$ for d_m a running mean of the k -nearest neighbor distance:

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{\sum_{f_i \in N_k} K(f(S_t), f_i) + c}}, \text{ with pseudo-count } c=0.001.$$

Never Give Up

The state embeddings are trained to ignore the parts not influenced by the actions of the agent.

To this end, Siamese network f is trained to predict $p(A_t | S_t, S_{t+1})$, i.e., the action A_t taken by the agent in state S_t when the resulting state is S_{t+1} .

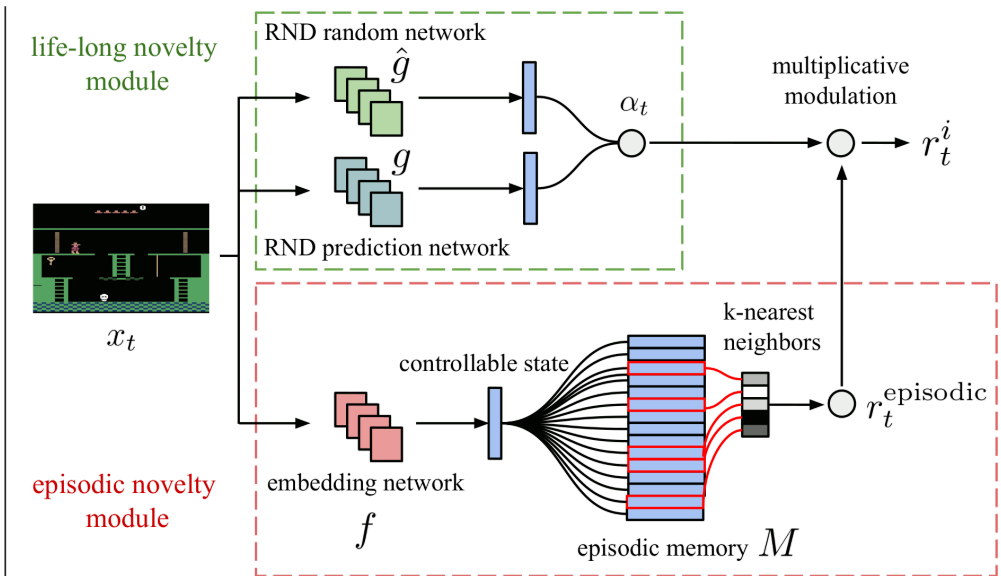
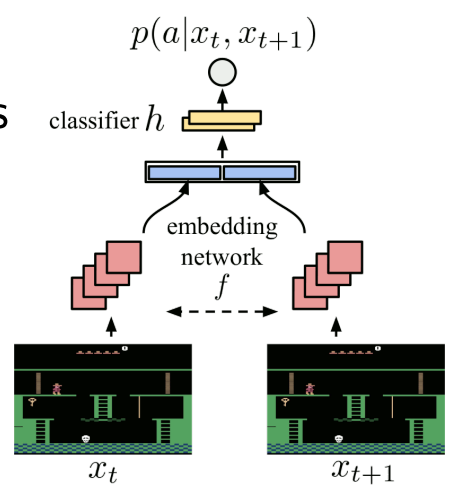
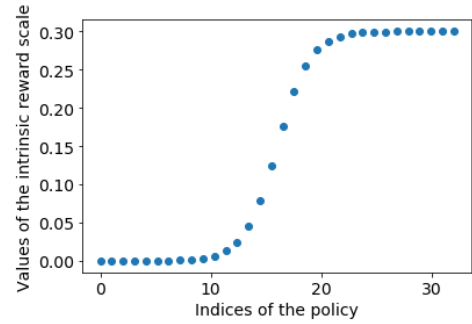


Figure 1 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

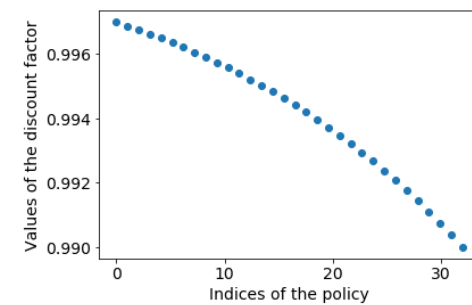
The life-long novelty $\alpha_t = 1 + \frac{\|\hat{g} - g\|^2 - \mu_{\text{err}}}{\sigma_{\text{err}}}$ is trained using random network distillation (RND), where a predictor network \hat{g} tries to predict the output of an untrained convolutional network g by minimizing the mean squared error; the μ_{err} and σ_{err} are the running mean and standard deviation of the error $\|\hat{g} - g\|^2$.

The NGU agent uses transformed Retrace loss with the augmented reward

$$r_t^i \stackrel{\text{def}}{=} r_t^{\text{episodic}} \cdot \text{clip} \left(1 \leq \alpha_t \leq L = 5 \right).$$



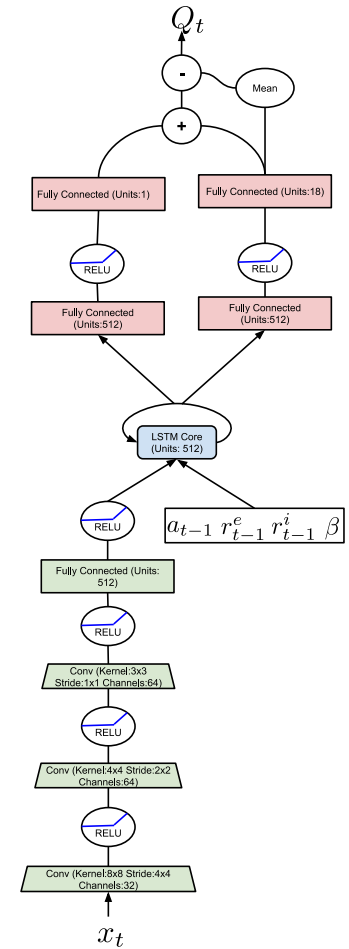
(a) Values taken by the $\{\beta_i\}_{i=0}^{N-1}$



(b) Values taken by the $\{\gamma_i\}_{i=0}^{N-1}$

To support multiple policies concentrating either on the extrinsic or the intrinsic reward, the NGU agent trains a parametrized action-value function $q(s, a, \beta_i)$ which corresponds to reward $r_t^{\beta_i}$ for $\beta_0 = 0$ and $\gamma_0 = 0.997$, ..., $\beta_{N-1} = \beta$ and $\gamma_{N-1} = 0.99$.

For evaluation, $q(s, a, 0)$ is employed.



(b) Detailed R2D2 Agent

Figure 17 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

Figure 7b of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

Algorithm	Gravitar	MR	Pitfall!	PrivateEye	Solaris	Venture
Human	3.4k	4.8k	6.5k	69.6k	12.3k	1.2k
Best baseline	15.7k	11.6k	0.0	11k	5.5k	2.0k
RND	3.9k	10.1k	-3	8.7k	3.3k	1.9k
R2D2+RND	15.6k±0.6k	10.4k±1.2k	-0.5±0.3	19.5k±3.5k	4.3k±0.6k	2.7k±0.0k
R2D2(Retrace)	13.3k±0.6k	2.3k±0.4k	-3.5±1.2	32.5k±4.7k	6.0k±1.1k	2.0k±0.0k
NGU(N=1)-RND	12.4k±0.8k	3.0k±0.0k	15.2k±9.4k	40.6k±0.0k	5.7k±1.8k	46.4±37.9
NGU(N=1)	11.0k±0.7k	8.7k±1.2k	9.4k±2.2k	60.6k±16.3k	5.9k±1.6k	876.3±114.5
NGU(N=32)	14.1k±0.5k	10.4k±1.6k	8.4k±4.5k	100.0k±0.4k	4.9k±0.3k	1.7k±0.1k

Table 1: Results against exploration algorithm baselines. Best baseline takes the best result among R2D2 (Kapturowski et al., 2019), DQN + PixelCNN (Ostrovski et al., 2017), DQN + CTS (Bellemare et al., 2016), RND (Burda et al., 2018b), and PPO + CoEx (Choi et al., 2018) for each game.

Table 1 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

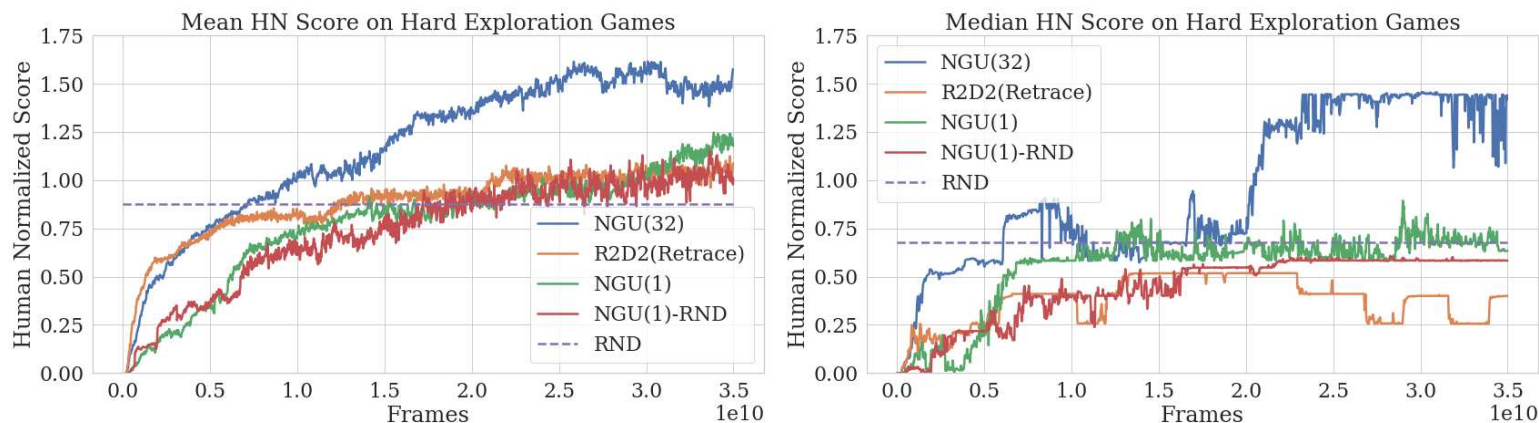


Figure 4: Human Normalized Scores on the 6 hard exploration games.

Figure 4 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

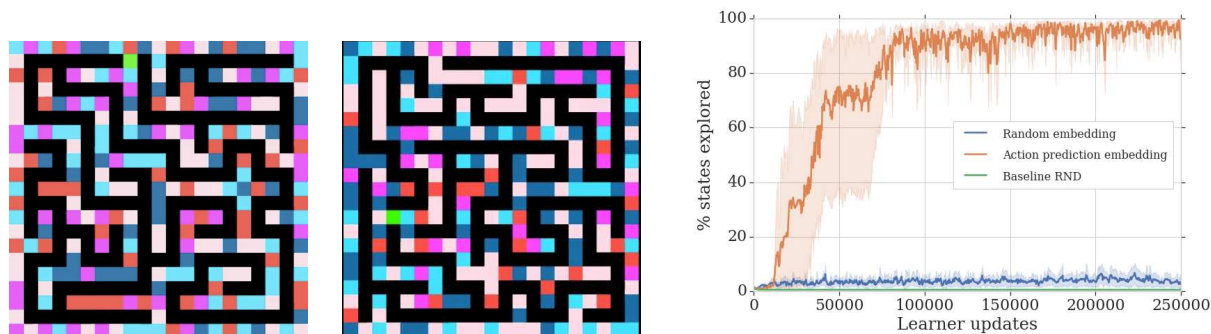


Figure 2: (Left and Center) Sample screens of Random Disco Maze. The agent is in green, and pathways in black. The colors of the wall change at every time step. (Right) Learning curves for Random projections vs. learned controllable states and a baseline RND implementation.

Figure 2 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

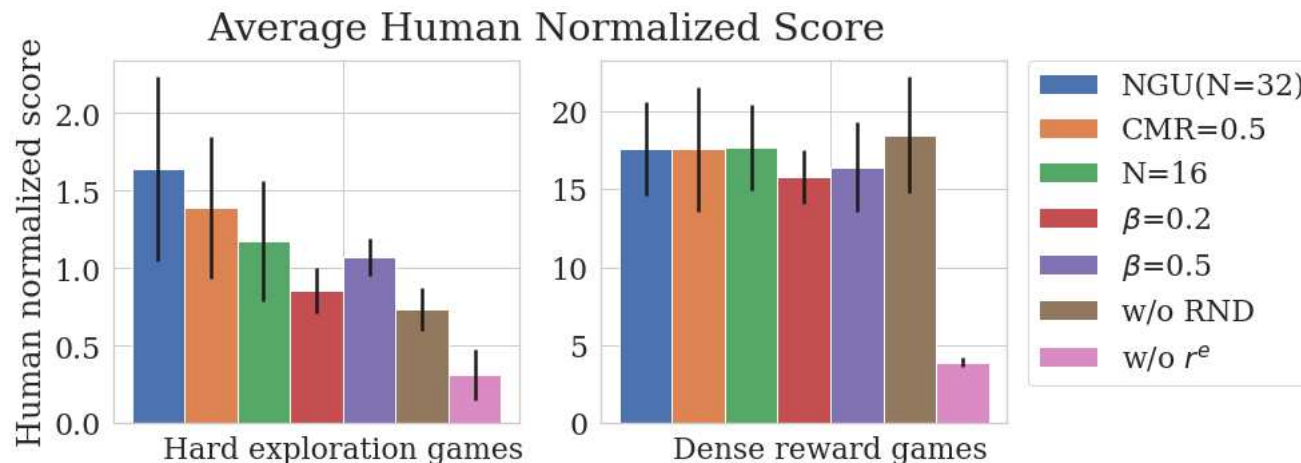


Figure 3 of "Never Give Up: Learning Directed Exploration Strategies" by A. P. Badia et al.

The Agent57 is an agent (from Mar 2020) capable of outperforming the standard human benchmark on all 57 games.

Its most important components are:

- Retrace; from *Safe and Efficient Off-Policy Reinforcement Learning* by Munos et al., <https://arxiv.org/abs/1606.02647>,
- Never give up strategy; from *Never Give Up: Learning Directed Exploration Strategies* by Badia et al., <https://arxiv.org/abs/2002.06038>,
- Agent57 itself; from *Agent57: Outperforming the Atari Human Benchmark* by Badia et al., <https://arxiv.org/abs/2003.13350>.

Then Agent57 improves NGU with:

- splitting the action-value as $q(s, a, j; \theta) \stackrel{\text{def}}{=} q(s, a, j; \theta^e) + \beta_j q(s, a, j; \theta^i)$, where
 - $q(s, a, j; \theta^e)$ is trained with r_e as targets, and
 - $q(s, a, j; \theta^i)$ is trained with r_i as targets.
- instead of considering all (β_j, γ_j) equal, we train a meta-controller using a non-stationary multi-arm bandit algorithm, where arms correspond to the choice of j for a whole episode (so an actor first samples a j using multi-arm bandit problem and then updates it according to the observed return), and the reward signal is the undiscounted extrinsic episode return; each actor uses a different level of ε_t -greedy behavior;
- γ_{N-1} is increased from 0.997 to 0.9999.

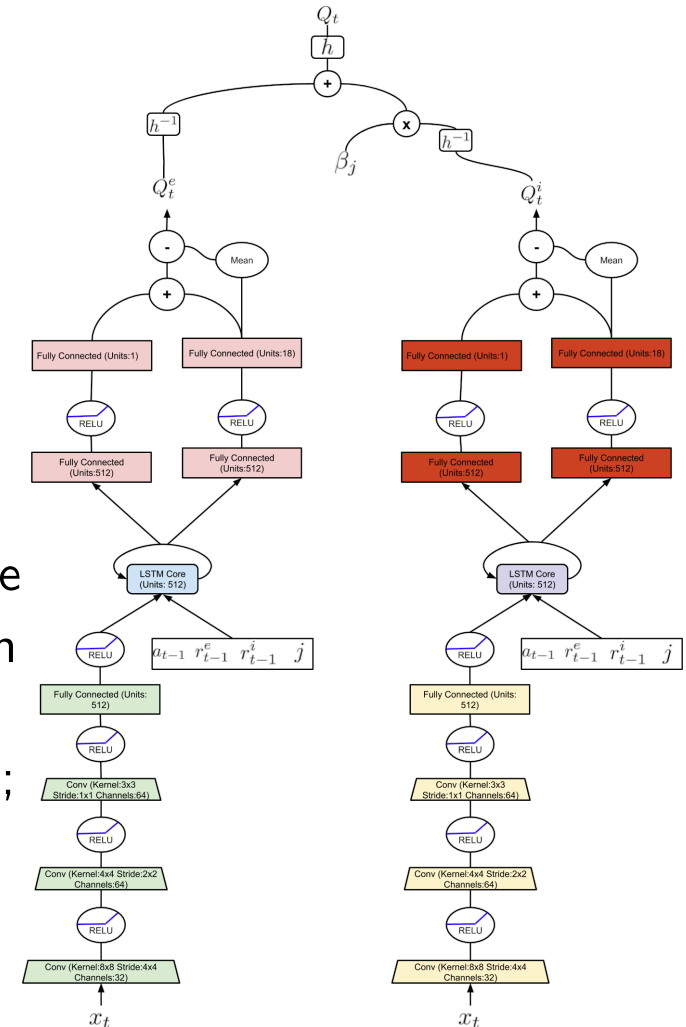


Figure 10 of "Agent57: Outperforming the Atari Human Benchmark" by A. P. Badia et al.

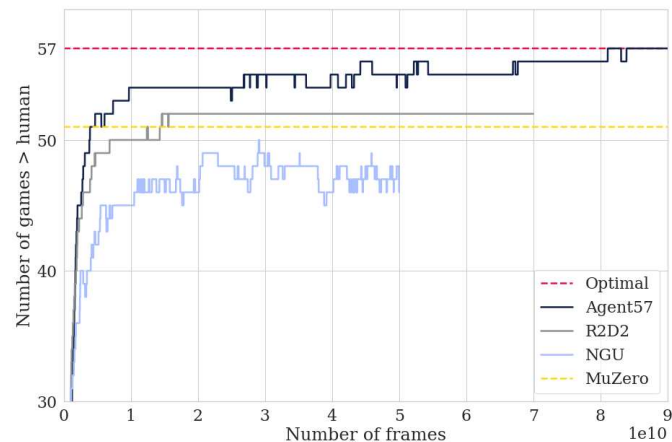


Figure 1. Number of games where algorithms are better than the human benchmark throughout training for Agent57 and state-of-the-art baselines on the 57 Atari games.

Figure 1 of "Agent57: Outperforming the Atari Human Benchmark" by A. P. Badia et al.

Table 1. Number of games above human, mean capped, mean and median human normalized scores for the 57 Atari games.

Statistics	Agent57	R2D2 (bandit)	NGU	R2D2 (Retrace)	R2D2	MuZero
Capped mean	100.00	96.93	95.07	94.20	94.33	89.92
Number of games > human	57	54	51	52	52	51
Mean	4766.25	5461.66	3421.80	3518.36	4622.09	5661.84
Median	1933.49	2357.92	1359.78	1457.63	1935.86	2381.51
40th Percentile	1091.07	1298.80	610.44	817.77	1176.05	1172.90
30th Percentile	614.65	648.17	267.10	420.67	529.23	503.05
20th Percentile	324.78	303.61	226.43	267.25	215.31	171.39
10th Percentile	184.35	116.82	107.78	116.03	115.33	75.74
5th Percentile	116.67	93.25	64.10	48.32	50.27	0.03

Table 1 of "Agent57: Outperforming the Atari Human Benchmark" by A. P. Badia et al.

Agent57 – Ablations

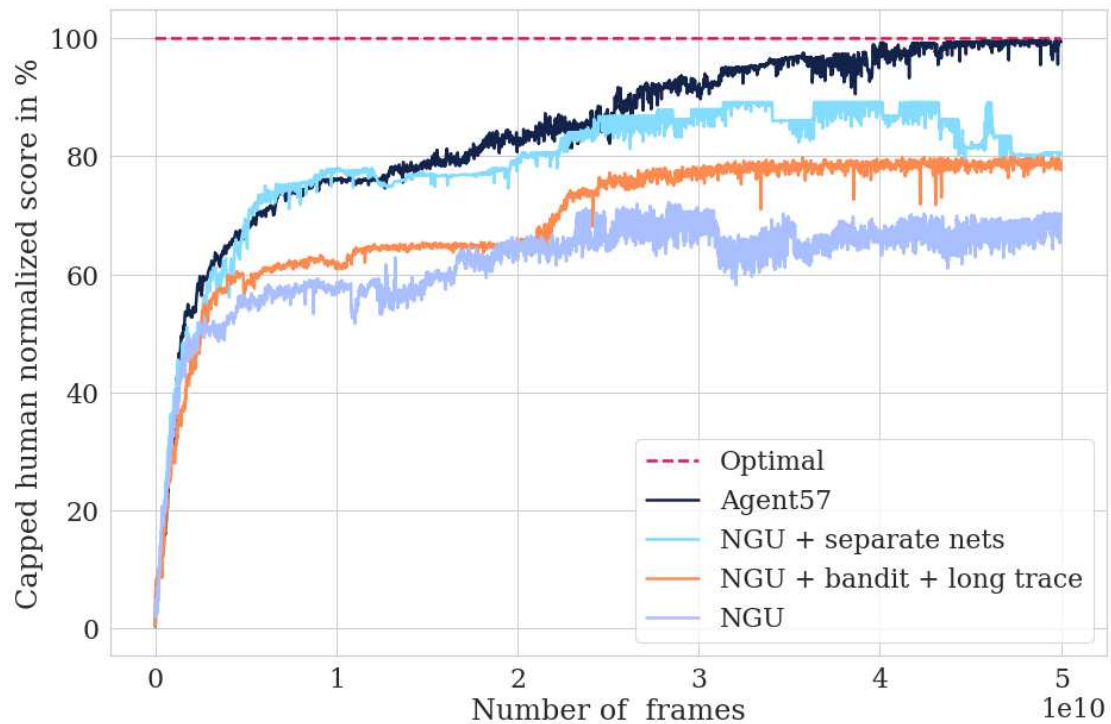


Figure 4. Performance progression on the 10-game *challenging set* obtained from incorporating each one of the improvements.

Figure 4 of "Agent57: Outperforming the Atari Human Benchmark" by A. P. Badia et al.

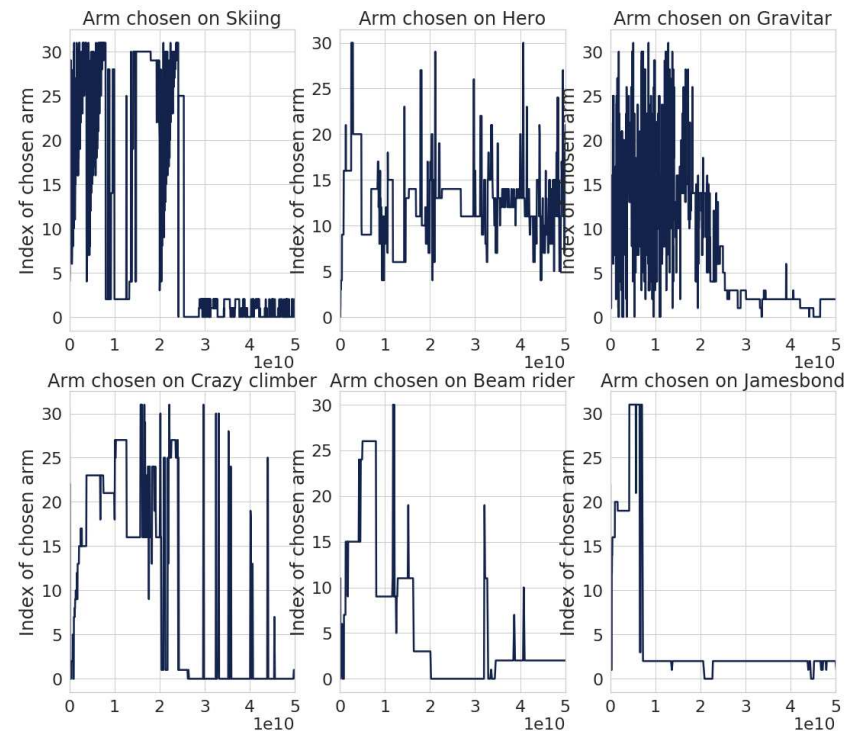


Figure 8. Best arm chosen by the evaluator of Agent57 over training for different games.

Figure 8 of "Agent57: Outperforming the Atari Human Benchmark" by A. P. Badia et al.