Reinforcement Learning

Deep Reinforcement Learning II

Stefano Albrecht, Arrasy Rahman, Filippos Christianos, Lukas Schäfer

6 March 2020

THE UNIVERSITY of EDINBURGH informatics

Lecture Outline

- Problems with experience replay
- Asynchronous methods for Deep RL
- Deep actor-critic methods
- Generalised advantage estimation
- Deep deterministic policy gradient
- Choosing algorithms

Recap: DQN

Deep Q-Network:

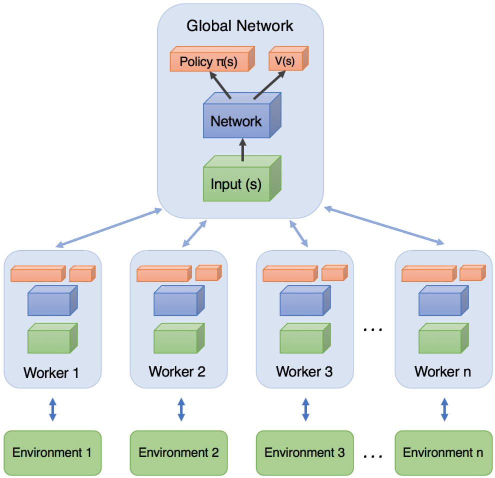
- Approximate state-action values using a neural network
- Stabilise training by:
  - — Sampling training batches from experience replay buffer
  - — Using separate target network to compute update targets

Problems of DQN

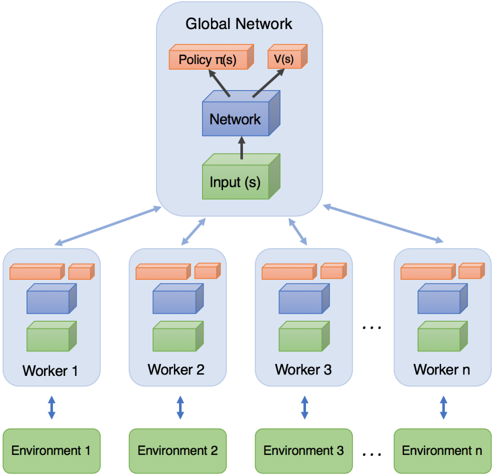
- Requires large storage for replay buffer
  (e.g. Atari game requires $\approx$56GB, cannot fit in a modern PC)
- Use of replay buffer requires off-policy method (*why?*)
- Not straightforward handling of multi-step returns (*why?*)

Asynchronous Training

Asynchronous Framework

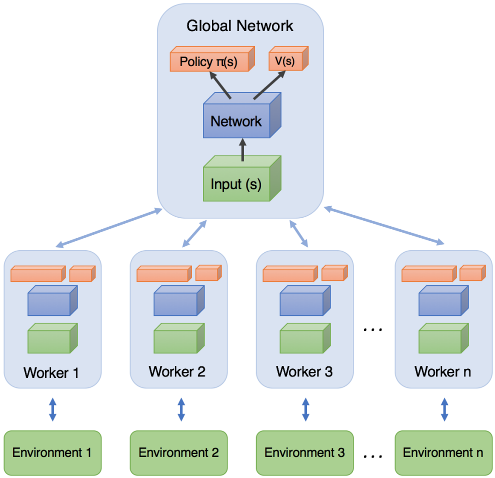


Asynchronous Framework



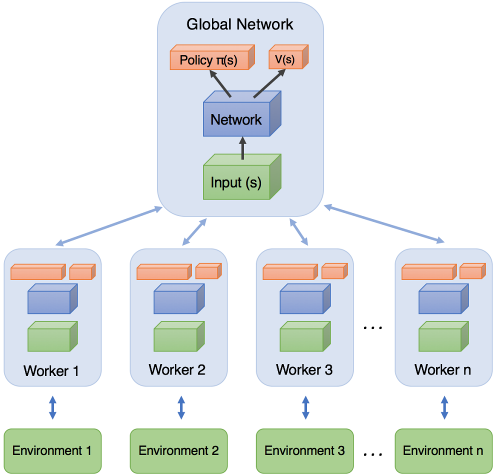
- Create n parallel "worker" threads with own environment copies and shared global network

Asynchronous Framework



- Create *n* parallel "worker" threads with own environment copies and shared global network
- Each worker interacts independently with its environment

Asynchronous Framework



- Create *n* parallel “worker” threads with own environment copies and shared global network
- Each worker interacts independently with its environment
- Asynchronous updates:
  Periodically, each worker updates the global network parameters based on its local experiences

Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples

Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples

  $\Rightarrow$ Means we don't need replay buffer!

Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples

  $\Rightarrow$ Means we don't need replay buffer!

- Better handling of sequential data: can use on-policy and multi-step returns

Benefits of Asynchronous Framework

- Asynchronous updating is another way of breaking correlation in samples
  $\Rightarrow$ Means we don't need replay buffer!
- Better handling of sequential data: can use on-policy and multi-step returns
- Runs on normal multi-threaded CPUs

Asynchronous 1-Step Q-Learning [Mnih *et al.*, 2016]

θ for value network

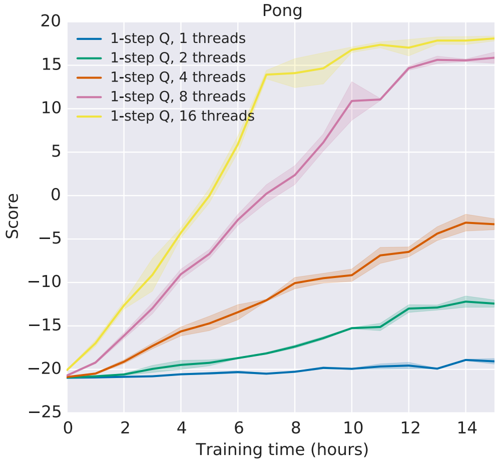
θ^- for target network

θ/θ^- are global shared between workers

repeat
  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
  Receive new state s' and reward r
  $$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$
  Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  if $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

More Workers, Faster Learning

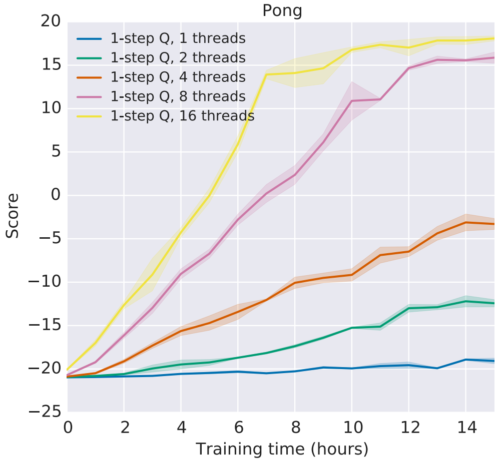
More workers (parallel threads) lead to faster learning



More Workers, Faster Learning

More workers (parallel threads) lead to faster learning

- Workers explore different parts of the environment
- Workers can use different exploration policies (e.g. ϵ-values)



Deep Actor-Critic

Recap: Actor-Critic Algorithm

Objective: Find parameters θ which maximise $J = V^{\pi_\theta}(s)$

Recap: Actor-Critic Algorithm

Objective: Find parameters θ which maximise $J = V^{\pi_\theta}(s)$

- Estimate gradient $\nabla_\theta J$ using the **policy gradient theorem**:

$$\nabla_\theta J = \mathbb{E}_{s\sim d(s),a\sim\pi}[R\,\nabla_\theta \log \pi_\theta(a|s)]$$

Recap: Actor-Critic Algorithm

Objective: Find parameters θ which maximise $J = V^{\pi_\theta}(s)$

- Estimate gradient $\nabla_\theta J$ using the **policy gradient theorem**:

$$\nabla_\theta J = \mathbb{E}_{s \sim d(s), a \sim \pi}[R \, \nabla_\theta \log \pi_\theta(a|s)]$$

- Approximate the return R using a critic $\hat{V}_w$ with parameters w

$$\nabla_\theta J = \mathbb{E}_{s \sim d(s), a \sim \pi}\left[(r + \hat{V}_w(s'))\nabla_\theta \log \pi_\theta(a|s)\right]$$

  Train the critic by minimising the TD-error

Recap: Actor-Critic Algorithm

Objective: Find parameters θ which maximise $J = V^{\pi_\theta}(s)$

- Estimate gradient $\nabla_\theta J$ using the **policy gradient theorem**:

$$\nabla_\theta J = \mathbb{E}_{s\sim d(s), a\sim\pi}[R\, \nabla_\theta \log \pi_\theta(a|s)]$$

- Approximate the return R using a critic $\hat{V}_w$ with parameters w

$$\nabla_\theta J = \mathbb{E}_{s\sim d(s), a\sim\pi}\left[(r + \hat{V}_w(s'))\nabla_\theta \log \pi_\theta(a|s)\right]$$

  Train the critic by minimising the TD-error

- Subtract a baseline function in order to reduce the variance of the estimation

$$\nabla_\theta J = \mathbb{E}_{s\sim d(s), a\sim\pi}\left[(r + \hat{V}_w(s') - \hat{V}_w(s))\nabla_\theta \log \pi_\theta(a|s)\right]$$

Asynchronous Advantage Actor-Critic (A3C) [Mnih *et al.*, 2016]

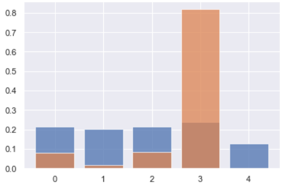
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

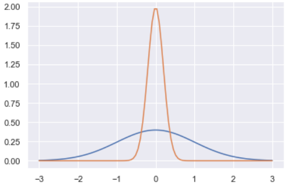
Entropy Regularisation

- Entropy of a stochastic policy

$$H[\pi(a|s)] = \mathbb{E}_{a\sim\pi(a|s)}[-\log \pi(a|s)] = -\sum_a \pi(a|s) \log \pi(a|s)$$

The entropy is maximised when the policy distribution is uniform



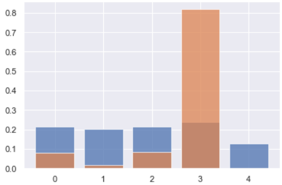


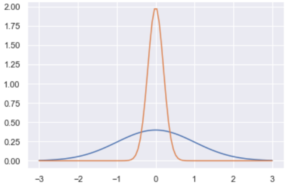
Entropy Regularisation

- Entropy of a stochastic policy

$$H[\pi(a|s)] = \mathbb{E}_{a\sim\pi(a|s)}[-\log \pi(a|s)] = -\sum_a \pi(a|s) \log \pi(a|s)$$

The entropy is maximised when the policy distribution is uniform





- Add an entropy regularisation in A3C

$$L_{actor} = -(R - V(s)) \log \pi(a|s) - \beta H[\pi(a|s)]$$

Encourage exploration by maximising entropy while minimising policy loss

Results of Asynchronous Methods [Mnih *et al.*, 2016]

Generalised Advantage Estimation

Generalised Advantage Estimation

We can compute advantage functions for multiple steps:

$$A_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t^V$$

Generalised Advantage Estimation

We can compute advantage functions for multiple steps:

$$A_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t^V$$

$$A_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) = \delta_t^V + \gamma \delta_{t_1}^V$$

Generalised Advantage Estimation

We can compute advantage functions for multiple steps:

$$A_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t^V$$

$$A_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) = \delta_t^V + \gamma \delta_{t_1}^V$$

$$A_t^{(k)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t) = \sum_{l=0}^{k} \gamma^l \delta_{t+l}^V$$

Generalised Advantage Estimation

We can compute advantage functions for multiple steps:

$$A_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t^V$$

$$A_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) = \delta_t^V + \gamma \delta_{t_1}^V$$

$$A_t^{(k)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t) = \sum_{l=0}^{k} \gamma^l \delta_{t+l}^V$$

Generalised advantage estimation (Schulman et al. 2016):

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \ldots) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

Generalised Advantage Estimation

Use the λ parameter to control the bias-variance trade-off

Generalised Advantage Estimation

Use the λ parameter to control the bias-variance trade-off

- For $\lambda = 0$:

$$A_t^{GAE(\gamma,0)} = \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

  This estimation has low variance but high bias

Generalised Advantage Estimation

Use the λ parameter to control the bias-variance trade-off

- For $\lambda = 0$:

$$A_t^{GAE(\gamma,0)} = \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

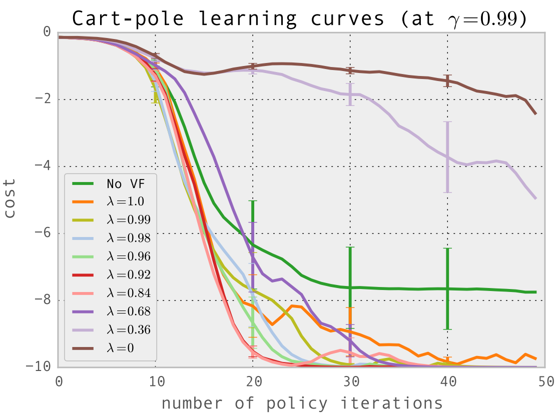
  This estimation has low variance but high bias

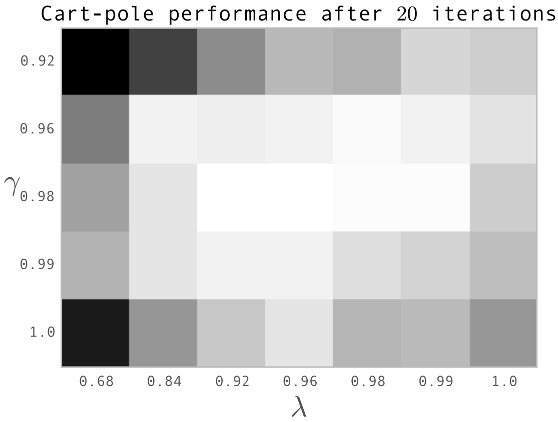
- For $\lambda = 1$:

$$A_t^{GAE(\gamma,1)} = \sum_{l=0}^{\infty} \delta_{t+l}^V = \sum_{l}^{\infty} \gamma^l r_{t+l} - V(s_t)$$

  This estimation has high variance but no bias

Effect of γ and λ [Schulman *et al.*, 2016]





Deep Deterministic Policy Gradient

- Can we use A3C?
  - *what is the disadvantage?*

Reinforcement Learning in Continuous Action Spaces

- Can we use A3C?
  - *what is the disadvantage?*
- Can we use DQN and discretize the action spaces?
  - *what is the disadvantage?*

Reinforcement Learning in Continuous Action Spaces

- Can we use A3C?
  - *what is the disadvantage?*
- Can we use DQN and discretize the action spaces?
  - *what is the disadvantage?*
- How to we compute $argmax_a Q(s, a)$ in continuous action spaces?

Deterministic Policy Gradient

- Extension of policy gradient to *deterministic* policies $\mu : S \to \mathbb{R}^{|A|}$

$$\nabla_\theta V(s_0) = \mathbb{E}_{s\sim d(s)}[\nabla_a Q(s, \mu(s))\nabla_\theta \mu(s)]$$

Deterministic Policy Gradient

- Extension of policy gradient to *deterministic* policies $\mu : S \to \mathbb{R}^{|A|}$

$$\nabla_\theta V(s_0) = \mathbb{E}_{s\sim d(s)}[\nabla_a Q(s, \mu(s)) \nabla_\theta \mu(s)]$$

- This method assumes *continuous actions* because it computes the gradient of Q-values with respect to them

Deterministic Policy Gradient

- Extension of policy gradient to *deterministic* policies $\mu : S \to \mathbb{R}^{|A|}$

$$\nabla_\theta V(s_0) = \mathbb{E}_{s \sim d(s)}[\nabla_a Q(s, \mu(s)) \nabla_\theta \mu(s)]$$

- This method assumes *continuous actions* because it computes the gradient of Q-values with respect to them
- Can be extended to discrete environments using mechanisms that produce differentiable samples from categorical distribution (e.g. *Gumbel-Softmax*)

Deterministic Policy Gradient

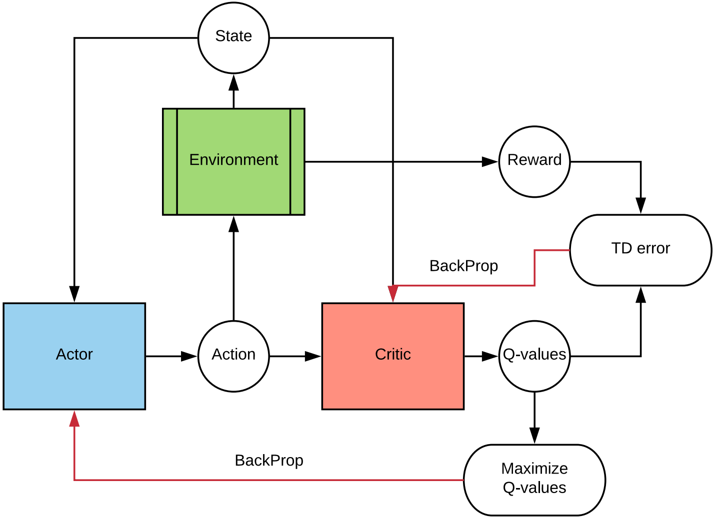
- Extension of policy gradient to *deterministic* policies $\mu : S \rightarrow \mathbb{R}^{|A|}$

$$\nabla_\theta V(s_0) = \mathbb{E}_{s \sim d(s)}[\nabla_a Q(s, \mu(s)) \nabla_\theta \mu(s)]$$

- This method assumes *continuous actions* because it computes the gradient of Q-values with respect to them
- Can be extended to discrete environments using mechanisms that produce differentiable samples from categorical distribution (e.g. *Gumbel-Softmax*)
- Train the critic by minimising the TD-error:

$$L = \frac{1}{2}\Big(r + \gamma Q_{target}(s', \mu_{target}(s')) - Q(s, a)\Big)^2$$

Deterministic Policy Gradient – Diagram



Deep Deterministic Policy Gradient (DDPG) [Lillicrap *et al.*, 2016]

for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

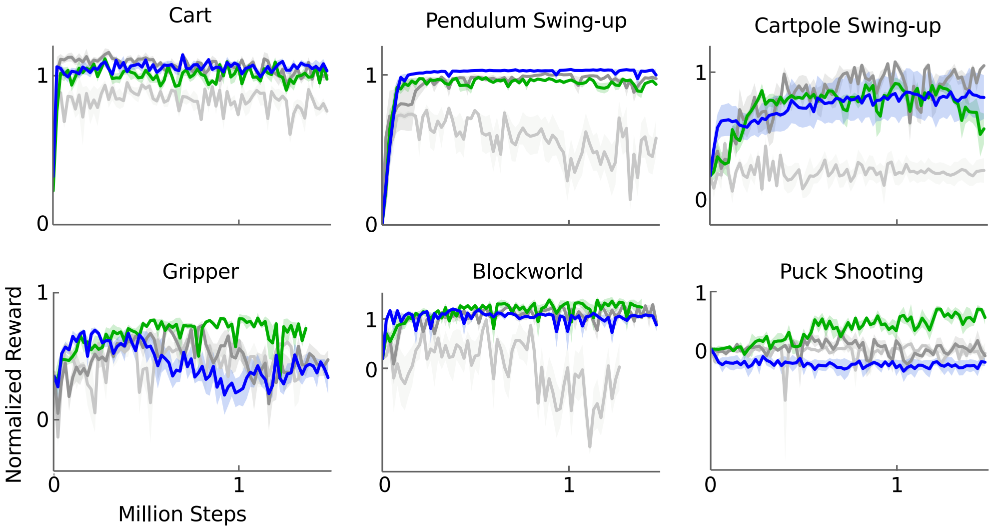
$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

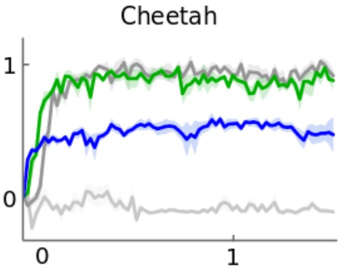
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

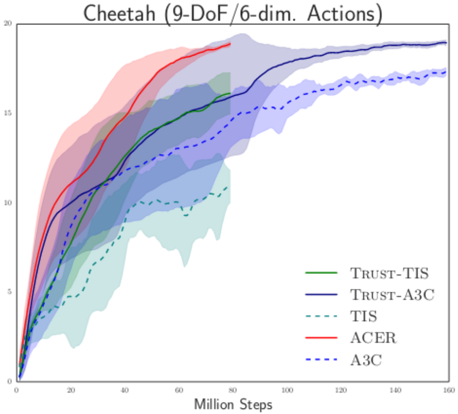
DDPG Performance in Continuous Tasks [Lillicrap *et al.*, 2016]



Original DPG (light grey); With target network (dark grey); With target networks and batch normalisation (green); With target networks from pixel-only (blue)

Sample Efficiency of DDPG [Wang et al., 2017]





DDPG converges in 1M steps, A3C requires 150M steps

Exploration

- For Q-learning use ϵ-greedy: anneal ϵ from 1.0 to a smaller value
  - In asynchronous Q-learning, workers can use different ϵ values
  - Each worker process will be annealed to a certain ϵ value

Exploration

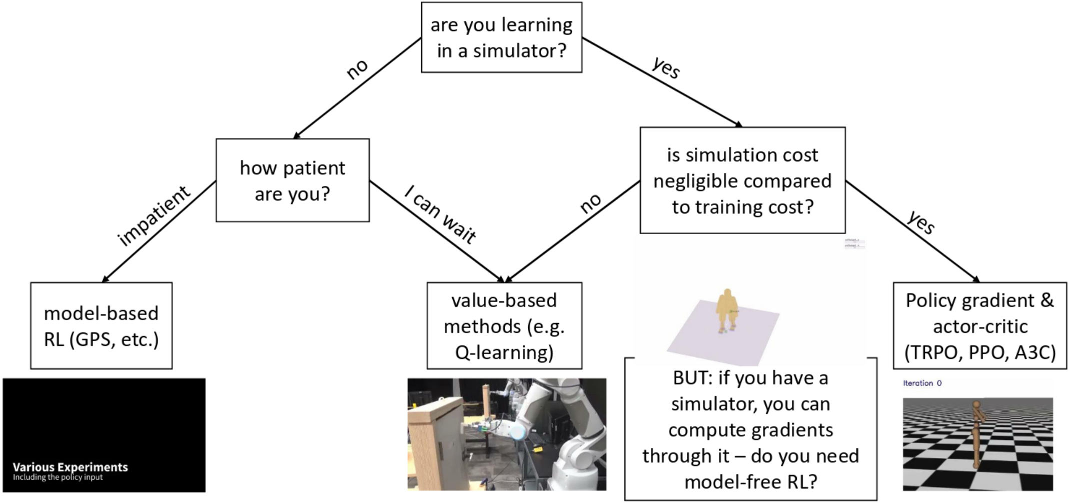
- For Q-learning use ϵ-greedy: anneal ϵ from 1.0 to a smaller value
  - In asynchronous Q-learning, workers can use different ϵ values
  - Each worker process will be annealed to a certain ϵ value
- In A3C the actions are sampled from a softmax distribution and exploration is encouraged through an entropy-based term in the actor's loss

Exploration

- For Q-learning use ϵ-greedy: anneal ϵ from 1.0 to a smaller value
    - In asynchronous Q-learning, workers can use different ϵ values
    - Each worker process will be annealed to a certain ϵ value
- In A3C the actions are sampled from a softmax distribution and exploration is encouraged through an entropy-based term in the actor's loss
- In DDPG add a random noise to the output of the actor (e.g. Gaussian noise, Ornstein–Uhlenbeck noise)

Choosing Algorithms

Which RL Algorithm To Choose?



http://rail.eecs.berkeley.edu/deeprlcourse-fa17/index.html

Debugging Deep RL Algorithms

- Start with simple environments that are quick to train on
- Log everything (frequently)!
  - In particular, keep track of:
    - Performance
    - Exploration hyperparameters
    - Loss function components
    - Gradients (Ensure they do not explode)
  - Save your logs in a format that can be used for further processing
  - Use tools that automatically displays your logs as Figures, e.g. Tensorboard

Debugging Deep RL Algorithms

- Policy Gradient
  - Policy should not get too close to deterministic policies early on
  - Track the magnitude of the policy gradient loss and entropy loss
- Q-Learning based methods
  - Track learning rate schedules
  - Track exploration schedule
  - Check magnitude of the gradients
- Visualize the policies during evaluation

Reading (Optional)

- Volodymyr, Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. ”Asynchronous methods for deep reinforcement learning.” In International Conference on Machine Learning, pp. 1928-1937, 2016
- John, Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. ”High-dimensional continuous control using generalized advantage estimation.” arXiv preprint arXiv:1506.02438 (2015)
- Timothy P., Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. ”Continuous control with deep reinforcement learning.” arXiv preprint arXiv:1509.02971 (2015)

Going Forward ...

- ~ 4 weeks left for the coursework
- Demonstrations next week
  - Come with questions prepared!
  - If you are unfamiliar with PyTorch, check out the documentation and tutorials on https://pytorch.org

Any Questions?