Introduction to Reinforcement Learning - pt. 2 : Policy Gradients

Deep Learning Course

Kevin Webster, Pierre Harvey Richemond

Course outline

1. Introduction to deep learning
2. Neural networks optimisation
3. Convolutional neural networks
4. Introduction to reinforcement learning - part 1
5. Introduction to reinforcement learning - part 2
6. Sequence models
7. Generative adversarial networks (GANs)
8. Variational autoencoders (VAEs)
9. Normalising flows
10. Theories of deep learning

Reminder : Deep Q-learning

In the previous lecture, we were approximating the (optimal) action-value function using neural networks and their weights θ, so that

$$Q^*_\theta(s,a) \approx Q^*(s,a)$$
$$V^*_\theta(s,a) \simeq V^*(s,a) = \max_{a'} \quad Q^*(s,a')$$

Knowledge of the Qs gave us both an acting policy - what to do in each state - and a state-value function - how good that state is. We were acting greedily (or ϵ-greedily) w.r.t. the policy defined implicitly by the Q-values, and proceeded to improve it iteratively by doing stochastic gradient descent on the mean squared Bellman residual, derived from (s,a,r,s') transitions from our experience buffer.

Sutton's deadly triad

The risk of numerical divergence exists whenever we combine those three elements:

- **Function approximation** A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources.
- **Bootstrapping** Update targets that include existing estimates, rather than relying exclusively on actual rewards and complete returns (as in MC methods).
- **Off-policy training** Training on a distribution of transitions other than that produced by the target policy (as in Q-learning).

Can we take an alternative view to Q-learning in that context ?

Policy methods - another RL paradigm

- In this lecture, we will proceed differently, and directly parameterize the policy

$$\begin{aligned}\pi_\theta(s,a) &= \mathbb{P}_\theta[a|s]\\ &= \mathbb{P}[a_t = a|s_t = s, \theta_t = \theta]\end{aligned}$$

- So instead of parameterizing the value function and doing greedy policy improvement we parameterize the policy (usually by a neural network with softmax output layer), and do gradient descent into a direction that improves it.
- To this end, we will need to define a *policy score function*.
- We will focus again on doing *model-free* reinforcement learning.

Policy methods - motivation

Advantages:

- Better convergence properties & simpler analysis than Q-learning
- Effective in high-dimensional or continuous action spaces (robotics...)
- The policy as defined above can be stochastic (rather than deterministic $a = \pi(s)$).
- Sometimes a deterministic policy is clearly suboptimal (rock-paper-scissors)
- The randomness inherent to a stochastic policy always leads to more exploration.

Policy methods - cons

Disdvantages:

- Evaluating a policy is typically inefficient and high-variance !
- Typically, convergence is to a local, rather than global, maximum.

Policy representation

Our typical example will be function approximation with a neural network, where parameter θ is now typically its weights, and a discrete action space.

In this case the usually accepted way to represent the policy is via a *softmax in action preferences* output layer :

$$\pi(a|s,\theta) := \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,a,\theta)}}$$

with $h(s,a,\theta)$ the logit numerical preferences for each state-action pair. Our output is now a probability distribution over actions.

Policy objectives

- In contrast to value-based methods, policy-based model-free methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameters θ by performing approximate gradient ascent on $J(\theta)$, the objective function.
- The usual objective we will optimize is the sum of episodic discounted rewards

$$J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{i=1}^{H} \gamma^i \cdot r_i\Big] = V^{\pi_\theta}(s_0)$$

- The typical SGA update is therefore simply

$$\theta_{t+1} \leftarrow \theta_t + \alpha \cdot \tilde{\nabla}_\theta J(\theta_t)$$

i.e. we need compute a stochastic policy gradient.

Interpretation

- All methods that follow this general schema we call policy gradient methods, whether or not they also learn an approximate value function. Methods that learn approximations to both policy and value functions are often called actor–critic methods, where ‘actor’ is a reference to the learned policy, and ‘critic’ refers to the learned value function, usually a state-value function [Sutton and Barto, 2018].
- We now focus on deriving an analytical expression for the policy gradient.

Policy gradient theorem - Derivation

The likelihood ratio trick (or REINFORCE trick), is ubiquitous:

$$\begin{aligned}\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a) \cdot \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ &= \pi_\theta(s,a) \cdot \nabla_\theta \log \pi_\theta(s,a)\end{aligned}$$

We derive the policy gradient theorem in the one-step MDP case for simplicity. Differentiating

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[R\right] = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

yields gradients

$$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) R(s,a)$$

so that, like the score function from statistics,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \cdot R\right]$$

Policy gradient theorem - Statement

Theorem

In a one-step MDP (then in fact in any episodic MDP, see [Sutton and Barto, 2018]):

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \cdot R(s,a)\right] \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \cdot Q^{\pi_\theta}(s,a)\right]
\end{aligned}$$

so that

$$\begin{aligned}
\tilde{\nabla}_\theta J(\theta_t) &= \langle \nabla_\theta \log \pi_\theta(s,a) | R(s,a) \rangle_{\textit{batch, on-policy}} \\
&= \langle \nabla_\theta \log \pi_\theta(s,a) | Q^{\pi_\theta}(s,a) \rangle_{\textit{batch, on-policy}}
\end{aligned}$$

We admit the Q^{π_θ} form in the interest of time here. There are several more forms of the PG theorem, differing by the right term in the inner product. Each of them gives rise to a different algorithm.

The REINFORCE algorithm

- The policy gradient theorem tells us we can do gradient ascent on our policy parameters, by replacing the on-policy expectation with sample episodic trajectories, scaling steps by the terminal reward, and updating parameters in the direction of the gradient of the log-policy (known through auto-differentiation).
- This is exactly what the REINFORCE algorithm [Williams and Peng, 1991] does. Standard REINFORCE updates the policy parameters θ in the direction

$$\nabla_\theta \log \pi(a_t|s_t;\theta) \cdot R_t$$

in an episodic fashion. No replay buffer or explicit exploration are needed.

The REINFORCE algorithm : Monte Carlo PG

1. Initialize the network with random weights
2. Play N full episodes, saving their (s, a, r, s') transitions
3. For every step t of every episode k, calculate the discounted total reward for subsequent steps, $Q_{k,t} = \sum_i \gamma^i r_i$
4. Calculate the loss function for all transitions

$$\mathcal{L} = -\sum_{k,t} Q_{k,t} \log \pi(s_{k,t}, a_{k,t})$$

5. Perform SGD weights update
6. Repeat from step 2, until convergence

Issues with this approach

1. Requires full episodes
2. Policy gradients are potentially high variance.

Variance reduction with baselines

- We subtract a baseline function $B(s)$ of s only (as a control variate) from the policy gradient:

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \cdot B(s)\right] &= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) B(s) \\ &= \sum_s d(s) B(s) \cdot \nabla_\theta \Big(\underbrace{\sum_a \pi_\theta(s,a)}_{=1} \Big) = 0\end{aligned}$$

- A good baseline to use is therefore the state-value function $B(s) = V^{\pi_\theta}(s)$.
- The policy gradient becomes, defining the advantage function A,

$$\begin{aligned}A^{\pi_\theta}(s,a) &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \cdot A^{\pi_\theta}(s,a)\right]\end{aligned}$$

- This provides a third form of the policy gradient.

Variance reduction with baselines - Summary

- When an approximate value function is used as the baseline, the quantity $R_t - b_t$ used to scale the policy gradient can be seen as an estimate of the *advantage* of action a_t in state s_t - instead of measuring the absolute goodness of an action we want to know how much better than "average" it is to take an action given a state -, because R_t is an estimate of $Q^\pi(a_t, s_t)$ and b_t is an estimate of $V^\pi(s_t)$.
- A learned estimate of the value function is commonly used as the baseline $b_t(s_t) \approx V^\pi(s_t)$, leading to a much lower variance estimate of the policy gradient.
- This approach can be viewed as an actor-critic architecture, where the policy π is the actor and the baseline b_t is the critic. policy gradient becomes

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi(a_t|s_t;\theta) \cdot (R_t - b(s_t))$$

The actor-critic paradigm

- Actor-Critic: Instead of waiting until the end of an episode as in REINFORCE, we use bootstrapping, and make an update at each step.
- To do that, we also train an oracle neural network critic $Q_{\theta_v}(s, a)$ that approximates the value function $Q^{\pi_\theta}(s, a)$.
- Now we have two function approximators: one for the policy π_θ (the actor), one for the critic. This is basically TD, but for Policy Gradients.
- A good estimate of the advantage function in the Actor-Critic algorithm is the TD-error. This gives rise to A2C : advantage actor-critic.

A2C : n-step returns

We want to get, pathwise

$$Q_{\theta_v}(s,a) \approx Q^{\pi_\theta}(s,a) \approx \sum_{i=1}^{N} r_1 + \gamma \cdot r_2 + \cdots + \gamma^{i-1} \cdot r_i$$

This is a regression problem : we can get it by mean-squared-error training, and then doing gradient descent step on the sampled TD-error.

A2C : n-step returns

This update folds itself recursively into n-step returns for the value function:

$$V \leftarrow r + \gamma \cdot V$$

In practice (convolutional architecture for instance), weights for the actor and the critic are often shared - feature re-use typically provides a convergence boost. Also note that when backpropagating n-step returns, we can perform n gradient updates (one per partial trajectory).

Advantage Actor-Critic - Entropy regularization

- In practice, to ensure exploration and prevent premature convergence[1] (punish overconfidence), we require that the policy never becomes deterministic.
- To this end we generally use the policy entropy (Shannon) as a regularizer for the objective function :

$$\beta \cdot H(\pi_\theta, s) = -\beta \cdot \sum_i \pi_\theta(a_i|s) \log \pi_\theta(a_i|s)$$

- This gives rise to the total policy gradient

$$\nabla_\theta \log \pi(a_t|s_t;\theta)(R_t - V(s_t;\theta_v)) + \beta \cdot \nabla_\theta H(\pi(s_t;\theta))$$

.

[1] as well as make sure that the policy gradient is actually well defined.

Summary of PG loss functions

$$\nabla_\theta J(\theta) := \nabla_\theta \log \pi(a_t|s_t;\theta) \cdot \underbrace{R_t}_{\text{REINFORCE}}$$

$$\nabla_\theta J(\theta) := \nabla_\theta \log \pi(a_t|s_t;\theta) \cdot \underbrace{(R_t - B(s_t))}_{\text{REINFORCE with baseline}}$$

$$\nabla_\theta J(\theta) := \nabla_\theta \log \pi(a_t|s_t;\theta) \cdot \underbrace{(R_t - V(s_t;\theta_v))}_{\text{Learned, advantage critic V}}$$

$$\nabla_\theta J(\theta) := \underbrace{\nabla_\theta \log \pi(a_t|s_t;\theta)}_{\text{actor}} \cdot \underbrace{(R_t - V(s_t;\theta_v))}_{\text{critic}} + \underbrace{\beta \cdot \nabla_\theta H(\pi(s_t;\theta))}_{\text{entropy regularizer}}$$

A3C - Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C), [Mnih et al., 2016]: Instead of using an experience replay buffer as in DQN, use multiple agents on different execution threads to explore the state-space, and make decorrelated updates to the actor and the critic.

Asynchronous Advantage Actor-Critic - Algorithm

Algorithm 1 A3C - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
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**
        $R \leftarrow r_i + \gamma R$
        $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A3C - Results

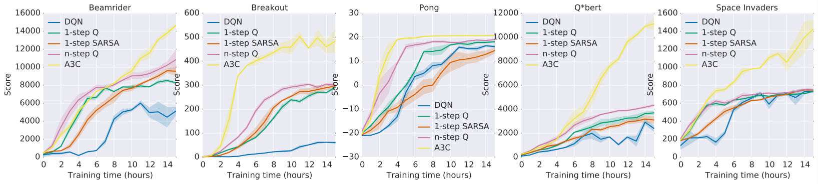

Figure 1: Learning speed comparison for DQN and the new asynchronous algorithms on five Atari 2600 games. DQN was trained on a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores. Plots are averaged over 5 runs. Taken from [Mnih et al., 2016].

Application to robotics : Dexterous manipulation

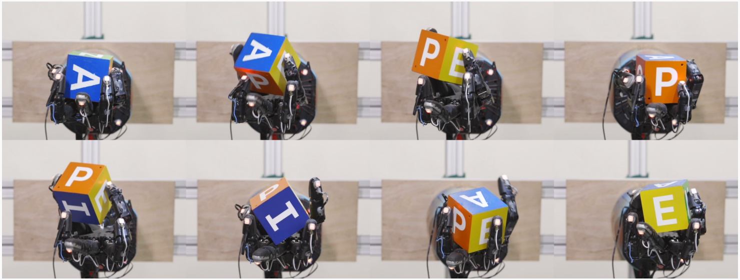

Figure 2: A five-fingered humanoid hand trained with reinforcement learning manipulating a block from an initial configuration to a goal configuration using vision for sensing. See [OpenAI et al., 2018] and this blog post.

Recent Advanced Policy Gradient and Actor-Critic Methods

We will be looking at four different, advanced policy gradient methods :

- TRPO ('Trust Region Policy Optimization') [Schulman et al., 2015]
- PPO ('Proximal Policy Optimization') [Schulman et al., 2017b]
- ACKTR (Kronecker-factored approximation) [Wu et al., 2017]
- PCL ('Path Consistency Learning') [Nachum et al., 2017]

These methods are all compatible with neural network function approximation. We will give high-level descriptions of the way they function.

Trust Region Policy Optimization - 1 (2015)

- Goal = stabilizing (parameterized) policy updates.
- The high-level idea [Schulman et al., 2015] is to take steps in directions that improve the policy, while simultaneously not straying too far from the old policy.
- Making too large a change from the previous policy, especially in high-dimensional, nonlinear environments, can lead to a dramatic decrease in performance. For example, a little forward lean helps running speed, but too much forward lean leads to a crash.
- TRPO takes a principled approach to controlling the rate of policy change - the algorithm places a constraint on the average Kullback-Leibler divergence between the new and old policy after each update.

Trust Region Policy Optimization - 2

- The change in reward η by updating policy π to policy $\hat{\pi}$ is

$$\eta(\hat{\pi}) = \eta(\pi) + \sum_s \rho_{\hat{\pi}}(s) \sum_a \hat{\pi}(a|s) A_\pi(s, a) \tag{1}$$

- The dependency on $\hat{\pi}$ through $\rho_{\hat{\pi}}(s)$ is complex, so we approximate/linearize via

$$L_\pi(\hat{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \hat{\pi}(a|s) A_\pi(s, a) \tag{2}$$

- In this case, one proves that by taking linear mixture steps, we can guarantee monotonic policy improvement :
$\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha \arg\max L_{\pi_{old}}(\pi)(a|s)$

Trust Region Policy Optimization - 3

Theorem (Schulman, 2015):

$$\eta(\hat{\pi}) \geq L_\pi(\hat{\pi}) - \frac{4\gamma}{(1-\gamma)^2} \cdot \max_{s,a} |A_\pi(s,a)| \cdot D_{KL}^{\max}(\pi, \hat{\pi})$$

(proven using policy coupling methods, and then comparing total variation distance and KL). Based on this pessimistic surrogate, we can iterately solve for policies (π_i) :

$$\pi_{i+1} = \arg\max_{\pi} \quad [L_{\pi_i}(\pi) - C \cdot D_{KL}^{\max}(\pi_i, \pi)]$$

This is a penalized optimization update.

Trust Region Policy Optimization - 4

- TRPO is a robust approximation to the update on the previous slide, using a constraint on the KL divergence rather than a penalty, in order to robustly allow large updates.
- We now use a policy parameter θ so that the algorithm becomes:
- Maximize $L_{\theta_{old}}(\theta)$ over θ, subject to constraint $D_{KL}^{\max}(\theta_{old}, \theta) \leq \delta$.
- Using Monte-Carlo expectations and cheating a little, this is equivalent to

$$\max_{\theta} \quad \mathbb{E}_{s \sim \rho_{\theta_{old}}, a \sim \theta_{old}} [\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{\theta_{old}}(s, a)]$$

- In practice the algorithm performs the optimization not by SGD, but by taking a quadratic approximation to the KL-divergence, and calculating conjugate gradients efficiently (see Hessian-vector products).

Proximal Policy Optimization - 1 (2017)

- TRPO is useful especially in continuous control tasks, but isn't easily compatible with algorithms that share parameters between a policy and value function or auxiliary losses.
- The algorithm is bespoke as it uses second-order information
- We would like to modify the policy gradients loss function in order to do a trust region update compatible with stochastic gradient descent. = proximal policy optimization [Schulman et al., 2017b]

Proximal Policy Optimization - 2

- This happens with the new objective function

$$L^{CLIP}(\theta) = \mathbb{E}_{empirical}\left[\min(r_t(\theta), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)) \cdot A_t\right]$$

- r_t is the ratio of the probability under the new and old policies, respectively
- ϵ is a hyperparameter, around 0.1 or 0.2.
- The algorithm is simplified : no more KL penalties and adaptive updates. We can train with SGD as usual.
- Why does this work ?

Proximal Policy Optimization - 3

- For comparison the policy gradient objective is with the same notations

$$L^{PG}(\theta) = \mathbb{E}\left[\log \pi_\theta(a|s) \cdot A_t\right]$$

- The TRPO objective is, neglecting the KL term,

$$L^{TRPO}(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \cdot A_t\right] = \mathbb{E}\left[r_t(\theta) \cdot A_t\right]$$

- In PPO's L^{CLIP}, the clipping term modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$.
- The min term in this scheme is conservative: we only ignore the change in probability ratio when it would make the objective improve, and we include it when it makes the objective worse.

ACKTR - 1 (2017)

- Idea 1 [Wu et al., 2017] = policy gradient updates following the natural gradient, gives us the direction in parameter space that achieves the largest instantaneous improvement in the objective per unit of change in the output distribution of the network - as measured using the KL-divergence.
- That is, the underlying is the Fisher metric:

$$\mathbb{E}_\pi \left[\nabla_\theta \log \pi(a|s)(\nabla_\theta \log \pi(a|s))^T\right]$$

- TRPO makes use of Hessian vector-products, in order to avoid inverting the Fisher information matrix explicitly.

ACKTR - 2

- K-FAC (Kronecker-Factored Approximation) is a quasi-Newton algorithm ($\theta_{t+1} \leftarrow \theta_t - \epsilon H_f^{-1} \nabla_\theta f$) replacing SGD optimizers for the training of neural networks.
- Idea 2 = use a structured approximation to the inverse Fisher matrix H_f^{-1}. This makes more progress per step, hence trades sample efficiency for computational cost.
- We can combine both natural policy gradients improvement and K-FAC.
- Helps solving the sample efficiency problem (faster per step convergence than PG+ADAM).

PCL - 1 (2017)

- Idea [Nachum et al., 2017] = make the entropic regularization in actor-critic discounted and recursive just like the rewards themselves.
- Inserting $\pi \log \pi$ recursively, this gives the Bellman equation for the loss as

$$L^{ENT}(s, \pi) = \sum_a \pi(a|s) \left[r(s, a) - \tau \log \pi(a|s) + \gamma L^{ENT}(s', \pi) \right]$$

- Doing this, all the math works out, with one-hot hard maxes replaced by Boltzmann soft-maxes (log-sum-exp operator, because it is the Fenchel-Legendre dual of the entropy regularization functional).

$$V^*(s) = L^{ENT}(s, \pi^*) = \tau \log \sum_a \exp \left[\frac{r(s, a) + \gamma V^*(s')}{\tau} \right]$$

$$\pi^*(a|s) = \frac{\exp \left[\frac{r(s,a) + \gamma V^*(s')}{\tau} \right]}{\exp(\frac{V^*(s)}{\tau})}$$

PCL - 2

- Hence for each pair (s, s') and action a we get at optimality:

$$V^*(s) - \gamma V^*(s') = r(s, a) - \tau \log \pi^*(a|s)$$

- Therefore on any full d-length sub-trajectory of states $(s_{i,i+d})$, we can define the loss $C(s_{i,i+d}, \theta, \phi)$ as

$$-V_\phi(s_i) + \gamma^d V_\phi(s_{i+d}) + \sum_{j=0}^{d-1} \gamma^j \left[r(s_{i+j}, a_{i+j}) - \tau \log \pi_\theta(a_{i+j}|s_{i+j}) \right]$$

- The joint (θ, ϕ) loss

$$L_{\theta,\phi}^{PCL} = \frac{1}{2} \sum_{s_{i,i+d}} C(s_{i,i+d}, \theta, \phi)^2$$

  is the MSE objective whose gradients are minimized by PCL.
- Note that under that formulation policy gradients and (soft) Q-learning are equivalent!
  [Neu et al., 2017, Schulman et al., 2017a]

DRL is hard... DRL that matters

- Encouraging reproducibility in reinforcement learning [Henderson et al., 2017]
- 'RL algorithms have many moving parts that are hard to debug, and they require substantial effort in tuning in order to get good results.'
- False Discoveries everywhere ? Dependency on the random seed
- Multiple runs are necessary to give an idea of the variability. Reporting the best returns is not enough - at the very least, report top and bottom quantiles
- Don't get discouraged !

Don't be an alchemist

Suggestions for experimental design

Factors influencing training, by order of importance:

- Fix your random seed for reproducibility !
- *Reward scaling* (and more generally reward engineering) help a lot
- *Number of steps* in the calculation of returns
- *Discount factor*, if you have one, also an issue - cf. Ape-X DQfD again
- RAINBOW (see previous lecture) and max(PPO/ACKTR) two widely accepted state-of-the-art baselines.

Clip gradients to avoid NaNs, visualize them with TensorBoard, and finally be patient waiting for convergence...

Github code repositories

TensorFlow

DeepMind-TRFL

Keras

TORCS-DDPG

Pytorch

RL-Adventure-2

OpenAI

Gym Roboschool Baselines

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017).
Deep Reinforcement Learning that Matters.
ArXiv e-prints.

Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016).
Asynchronous Methods for Deep Reinforcement Learning.
ArXiv e-prints.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2017).
Bridging the Gap Between Value and Policy Based Reinforcement Learning.
ArXiv e-prints.

Neu, G., Jonsson, A., and Gómez, V. (2017).
A unified view of entropy-regularized Markov decision processes.
ArXiv e-prints.

OpenAI, :, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2018).
Learning Dexterous In-Hand Manipulation.
ArXiv e-prints.

Schulman, J. (2016).
Optimizing expectations : from deep reinforcement learning to stochastic computation graphs.
Ph.D. thesis.

- Schulman, J., Chen, X., and Abbeel, P. (2017a).
  Equivalence Between Policy Gradients and Soft Q-Learning.
  ArXiv e-prints.

- Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015).
  Trust Region Policy Optimization.
  ArXiv e-prints.

- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b).
  Proximal Policy Optimization Algorithms.
  ArXiv e-prints.

- Sutton, R. and Barto, A. (2018).
  Reinforcement Learning : An Introduction.
  Online book - second edition.

Williams, R. and Peng, A. (1991).
Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.
Machine Learning, Kluwer.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., and Ba, J. (2017).
Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.
ArXiv e-prints.