

# The Nuts and Bolts of Deep RL Research

John Schulman

OpenAI

August 26, 2017

# Outline

Approaching New Problems

Ongoing Development and Tuning

General Tuning Strategies for RL

Policy Gradient Strategies

Q-Learning Strategies

Miscellaneous Advice

# Approaching New Problems

# New Algorithm? Use Small Test Problems

- ▶ Run experiments quickly
- ▶ Do hyperparameter search
- ▶ Interpret and visualize learning process: state visitation, value function, etc.
- ▶ Construct toy problems where your idea will be strongest and weakest, where you have a sense of what it should do
- ▶ Counterpoint: don't overfit algorithm to contrived problem
- ▶ Useful to have medium-sized problems that you're intimately familiar with

# New Task? Make It Easier Until Signs of Life

- ▶ Provide good input features
- ▶ Shape reward function

# POMDP Design



- ▶ Visualize random policy: does it sometimes exhibit desired behavior?
- ▶ Human control
  - ▶ Atari: can you see game features in downsampled image?
- ▶ Plot time series for observations and rewards. Are they on a reasonable scale?
  - ▶ `hopper.py` in gym:  
$$\text{reward} = 1.0 - 1\text{e-}3 * \text{np.square}(a).\text{sum}() + \text{delta\_x} / \text{delta\_t}$$
- ▶ Histogram observations and rewards

# Run Your Baselines

- ▶ Don't expect them to work with default parameters
- ▶ Recommended:
  - ▶ Cross-entropy method<sup>1</sup>
  - ▶ Well-tuned policy gradient method<sup>2</sup>
  - ▶ Well-tuned Q-learning + SARSA method

---

<sup>1</sup>István Szita and András Lörincz (2006). "Learning Tetris using the noisy cross-entropy method". In: *Neural computation*.

<sup>2</sup><https://github.com/openai/baselines>, <https://github.com/rll/rllab>

# Run with More Samples Than Expected



- ▶ Early in tuning process, may need huge number of samples
  - ▶ Don't be deterred by published work
- ▶ Examples:
  - ▶ TRPO on Atari: 100K timesteps per batch for  $KL = 0.01$
  - ▶ DQN on Atari: update freq=10K, replay buffer size=1M



# Ongoing Development and Tuning

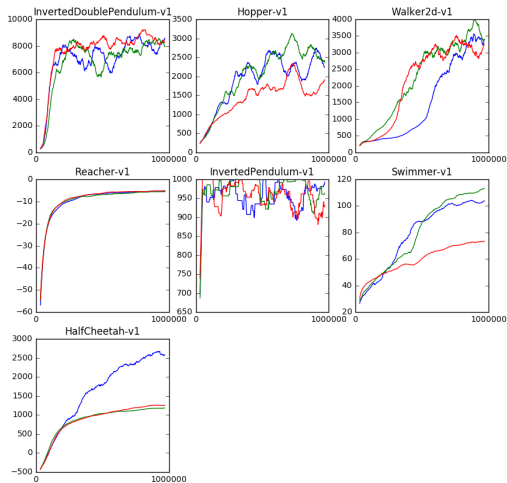
# It Works! But Don't Be Satisfied

- ▶ Explore sensitivity to each parameter
  - ▶ If too sensitive, it doesn't really work, you just got lucky
- ▶ Look for health indicators
  - ▶ VF fit quality
  - ▶ Policy entropy
  - ▶ Update size in output space and parameter space
  - ▶ Standard diagnostics for deep networks

# Continually Benchmark Your Code

- ▶ If reusing code, regressions occur
- ▶ Run a battery of benchmarks occasionally

# Always Use Multiple Random Seeds



# Always Be Ablating

- ▶ Different tricks may substitute
  - ▶ Especially whitening
- ▶ “Regularize” to favor simplicity in algorithm design space
  - ▶ As usual, simplicity  $\rightarrow$  generalization

# Automate Your Experiments

- ▶ Don't spend all day watching your code print out numbers
- ▶ Consider using a cloud computing platform (Microsoft Azure, Amazon EC2, Google Compute Engine)

# General Tuning Strategies for RL

# Whitening / Standardizing Data

- ▶ If observations have unknown range, standardize
  - ▶ Compute running estimate of mean and standard deviation
  - ▶  $x' = \text{clip}((x - \mu)/\sigma, -10, 10)$
- ▶ Rescale the rewards, but don't shift mean, as that affects agent's will to live
- ▶ Standardize prediction targets (e.g., value functions) the same way



# Generally Important Parameters

- ▶ Discount
  - ▶  $\text{Return}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$
  - ▶ Effective time horizon:  $1 + \gamma + \gamma^2 + \dots = 1/(1 - \gamma)$ 
    - ▶ I.e.,  $\gamma = 0.99 \Rightarrow$  ignore rewards delayed by more than 100 timesteps
  - ▶ Low  $\gamma$  works well for well-shaped reward
  - ▶ In TD( $\lambda$ ) methods, can get away with high  $\gamma$  when  $\lambda < 1$
- ▶ Action frequency
  - ▶ Solvable with human control (if possible)
  - ▶ View random exploration

# General RL Diagnostics


- ▶ Look at min/max/stddev of episode returns, along with mean
- ▶ Look at episode lengths: sometimes provides additional information
  - ▶ Solving problem faster, losing game slower

# Policy Gradient Strategies

# Entropy as Diagnostic

- ▶ Premature drop in policy entropy  $\Rightarrow$  no learning
- ▶ Alleviate by using entropy bonus or KL penalty

# KL as Diagnostic

- ▶ Compute KL  $[\pi_{\text{old}}(\cdot | s), \pi(\cdot | s)]$  
- ▶ KL spike  $\Rightarrow$  drastic loss of performance
- ▶ No learning progress might mean steps are too large
  - ▶ batchsize=100K converges to different result than batchsize=20K.

# Baseline Explained Variance

► explained variance =  $\frac{1 - \text{Var}[\text{empirical return} - \text{predicted value}]}{\text{Var}[\text{empirical return}]}$

# Policy Initialization

- ▶ More important than in supervised learning: determines initial state visitation
- ▶ Zero or tiny final layer, to maximize entropy



# Q-Learning Strategies

- ▶ Optimize memory usage carefully: you'll need it for replay buffer
- ▶ Learning rate schedules
- ▶ Exploration schedules
- ▶ Be patient. DQN converges slowly
  - ▶ On Atari, often 10-40M frames to get policy much better than random



# Miscellaneous Advice

- ▶ Read older textbooks and theses, not just conference papers
- ▶ Don't get stuck on problems—can't solve everything at once
  - ▶ Exploration problems like cart-pole swing-up
  - ▶ DQN on Atari vs CartPole
- ▶ Techniques from supervised learning don't necessarily work in RL: batch norm, dropout, big networks

That's all. Questions?