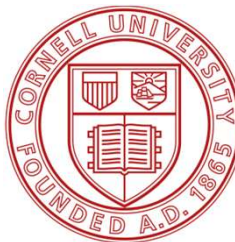
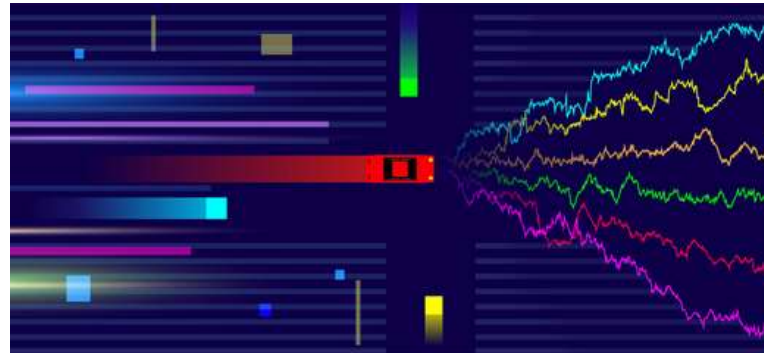
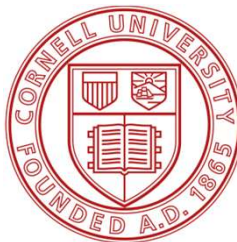


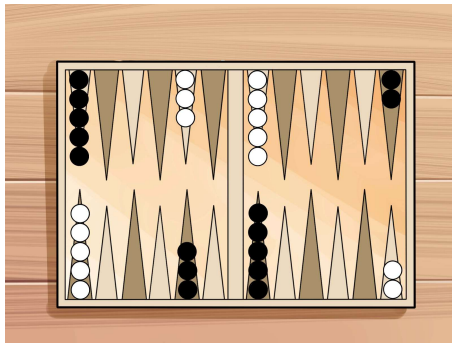
Introduction to Deep RL

Sean Sinclair
Cornell University

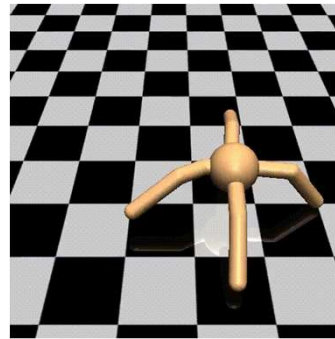




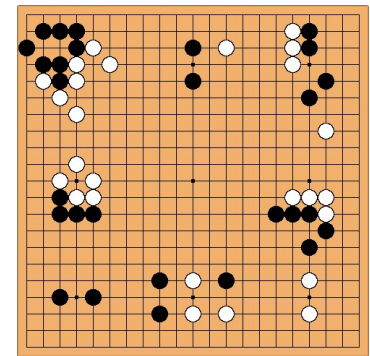
Success of RL



Backgammon



Mujoco Simulator



AlphaGo Zero

Focused on game playing + robotics

[Silver2017,Tesauro1995]

So far:

Previously saw algorithms designed with value and policy iteration for tabular (discrete) MDPs.

Model-Based

- Maintain estimates of reward and transition
- Plug estimates into Bellman equations for estimated V^*, Q^*
- Play greedy w.r.t. Q^*
- Time complexity / storage scales S^2A

Model Free

- Only maintain estimates of V^*, Q^* using fixed point
- Play greedy w.r.t. Q^*
- Better time complexity / storage

So far:

Previously saw algorithms designed with value and policy iteration for tabular (discrete) MDPs.

However, even if problem is tabular:

MemoryError: Unable to allocate 31.9 GiB for an array with shape (3094, 720, 1280, 3) and data type float32

$H = 3$

$S = 3094 \times 720$

$A = 1280$

So far:

Previously saw algorithms designed with value and policy iteration for tabular (discrete) MDPs.

However, even if problem is tabular:

MemoryError: Unable to allocate 31.9 GiB for an array with shape (3094, 720, 1280, 3) and data type float32

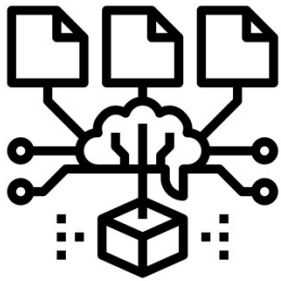
function approximation + state
aggregation help alleviate these
issues by appealing to ML
generalization

$H = 3$
 $S = 3094 \times 720$
 $A = 1280$

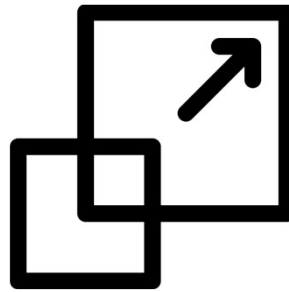
So far:

This workshop focuses on **RL for Operations**

We care about:



OR Models



Computation + Scale



Impact

Generalization

Want to use **RL in large-scale OR problems**

- Most problems have enormous state/action spaces
- Require representations (of models, state-action values, values, policies) that can generalize
- Represent quantities with a parameterized function instead of a table



Generalization

- Well developed theory for linear value function estimators for known features
- Works well *if* features are right
- Local representations (Kernel + discretization approaches) have nice properties (convergence results) but don't scale well
- Alternative: use **deep neural networks**:
 - uses distributed representation instead of local
 - “universal function approximator”



Generalization

Alternative: use **deep neural networks**:

- uses distributed representation instead of local
- “universal function approximator”



We will not focus on network representation choices, etc, until days 3+4. Consider arbitrary “parametric” representation (with gradients) for now

High level approach

Use a **deep neural network** to represent:

- Value function
- Policy
- Model of rewards and transitions
- “universal function approximator”

Optimize loss function via stochastic gradient descent (SGD)

High level approach

Use a **deep neural network** to represent:

- Value function
- Policy
- Model of rewards and transitions
- “universal function approximator”

Optimize loss function via stochastic gradient descent (SGD)

DQN

- Use neural network representation of value function
- Fits estimates using least squares

REINFORCE / VPG

- Uses neural network representation of policy
- Fits estimates using policy gradient loss

High level approach

Use a **deep neural network** to represent:

- Value function
- Policy
- Model of rewards and transitions
- “universal function approximator”

Optimize loss function via stochastic gradient descent (SGD)

DQN

- Use neural network representation of value function
- Fits estimates using least squares

REINFORCE / VPG

- Uses neural network representation of policy
- Fits estimates using policy gradient loss

DQN

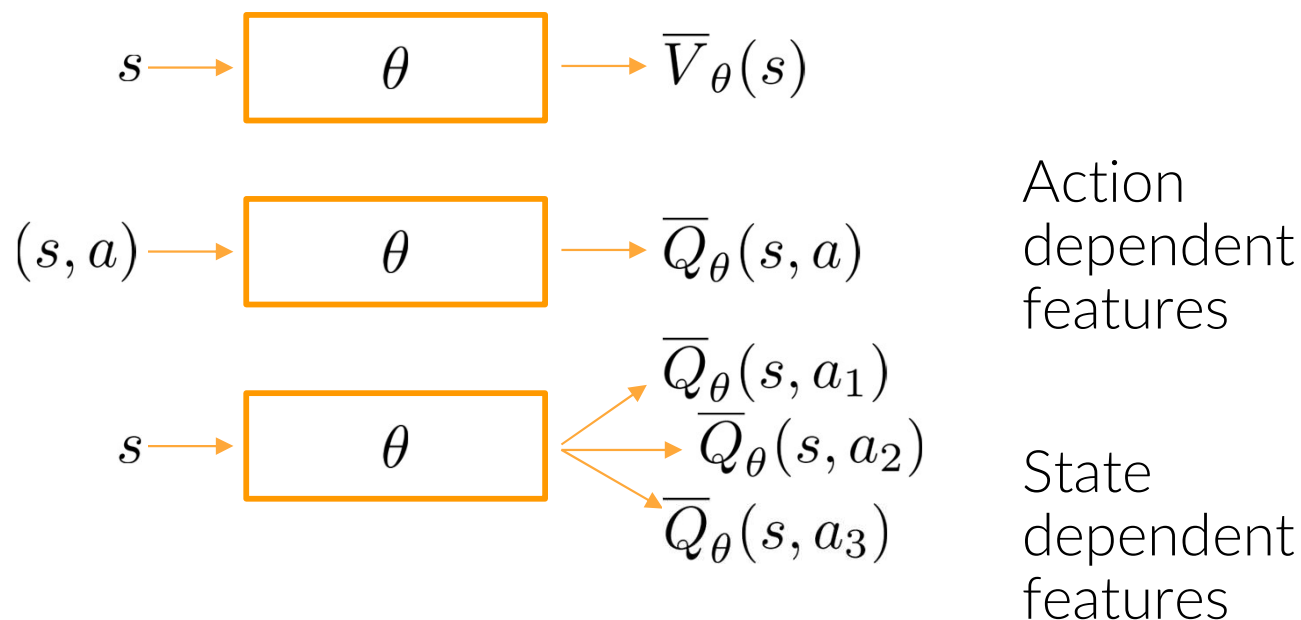
Use a **deep neural network** to represent:

- Value function
- Policy
- Model of rewards and transitions
- “universal function approximator”

Optimize loss function via stochastic gradient descent (SGD)

DQN

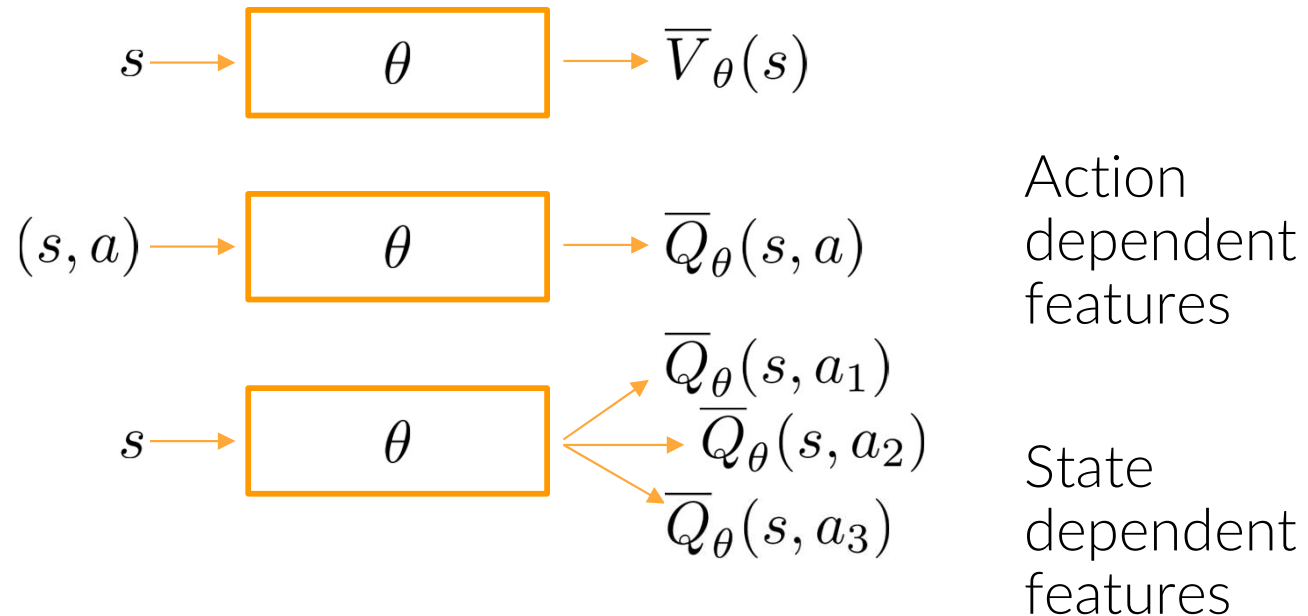
Use a **deep neural network** to represent value function:



Question: Which representation is better?

DQN

Use a **deep neural network** to represent value function:



Question: Which representation is better?

$$\pi_\theta(s) = \max_a \bar{Q}_\theta(s, a)$$

[Mnih2013]

Recall:

The Bellman Optimality Equations note that:

$$\begin{aligned} V^*(s) &= \max_a [r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)} [V^*(S')]] \\ Q^*(s, a) &= r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)} [V^*(S')] \end{aligned}$$

Or using that $V^*(s) = \max_a Q^*(s, a)$

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)} [\max_{a'} Q^*(S', a')]$$

Subtract off and plug in parametric estimates

$$0 = \bar{Q}_\theta(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)} [\max_{a'} \bar{Q}_\theta(S', a')])$$

[Mnih2013]

Recall:

Subtract off and plug in parametric estimates

$$0 = \overline{Q}_\theta(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot | s, a)} [\max_{a'} Q^*(S', a')])$$

Which objective function? Minimize **MSE**

$$L(\theta) = \mathbb{E}_{\pi_\theta} [(\overline{Q}_\theta(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot | s, a)} [\max_{a'} \overline{Q}_\theta(S', a')]))^2]$$


“Fixed Target”

Note: If $\mathbf{L}(\boldsymbol{\theta}) = \mathbf{0}$ then satisfy fixed point, so optimal

[Mnih2013]

Why MSE?

Which objective function? Minimize MSE

$$L(\theta) = \mathbb{E}_{\pi_{\theta}}[(\bar{Q}_{\theta}(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)}[\max_{a'} \bar{Q}_{\theta}(S', a')]))^2]$$

Assume tabular representation, “fixed target”, single sample from T

$$L(\theta) = \mathbb{E}_{\pi_{\theta}}[(\theta(s, a) - (r(s, a) + \gamma \max_{a'} \bar{Q}_{\theta}(s', a')))]^2]$$

Using gradient descent then....

$$\begin{aligned}\theta(s, a) &= \theta(s, a) - \alpha \nabla_{s, a} J(\theta) \\ &= \theta(s, a) - \alpha \left(r(s, a) + \gamma \max_{a'} \bar{Q}(s', a') - \theta(s, a) \right) (-1) \\ &= (1 - \alpha) \theta(s, a) + \alpha (r(s, a) + \gamma \max_{a'} \bar{Q}(s', a'))\end{aligned}$$


Recovers standard Q-Learning rules from earlier



[Mnih2013]

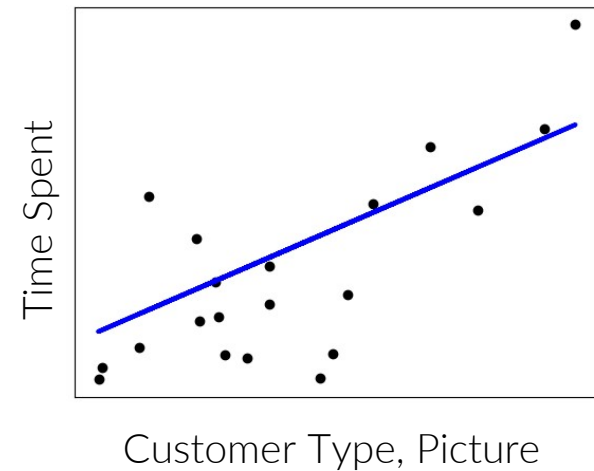
Nonstationary Target

Typically in supervised learning the “ground truth” is fixed

(, , 14 mins)

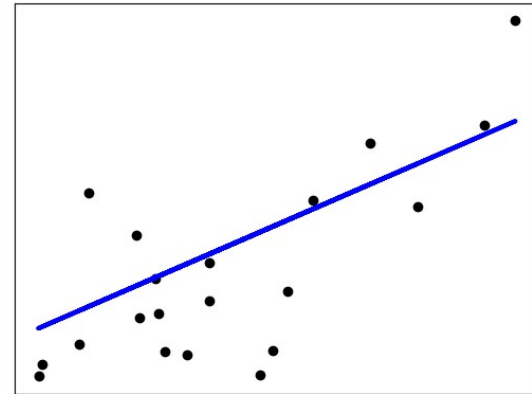
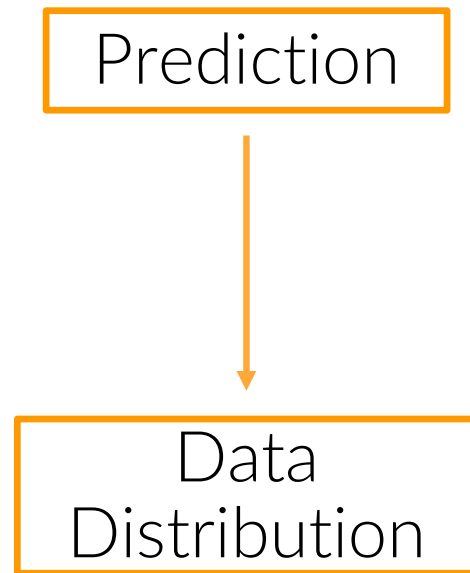
(, , 8 mins)

(, , 24 mins)



Nonstationary Target

Typically in supervised learning the “ground truth” is fixed



$$L(\theta) = \mathbb{E}_{\xi}[(f(\xi) - \overline{Q}_{\theta}(\xi))^2]$$

↑
No impact by network
parameters

Nonstationary Target

$$L^1(\theta) = \mathbb{E}_{\pi_\theta}[(\bar{Q}_\theta(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)}[\max_{a'} \bar{Q}_{\theta^1}(S', a')]))^2]$$

SGD Step for θ^2

$$L^2(\theta) = \mathbb{E}_{\pi_\theta}[(\bar{Q}_\theta(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)}[\max_{a'} \bar{Q}_{\theta^2}(S', a')]))^2]$$

SGD Step for θ^3

$$L^3(\theta) = \mathbb{E}_{\pi_\theta}[(\bar{Q}_\theta(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)}[\max_{a'} \bar{Q}_{\theta^3}(S', a')]))^2]$$

SGD Step for θ^4

Nonstationary Target



Like a dog chasing its own tail...

DQN

Converges to true Q^* using tabular representation

Diverges in neural networks due to:

- spurious correlation in samples
- “nonstationary target”

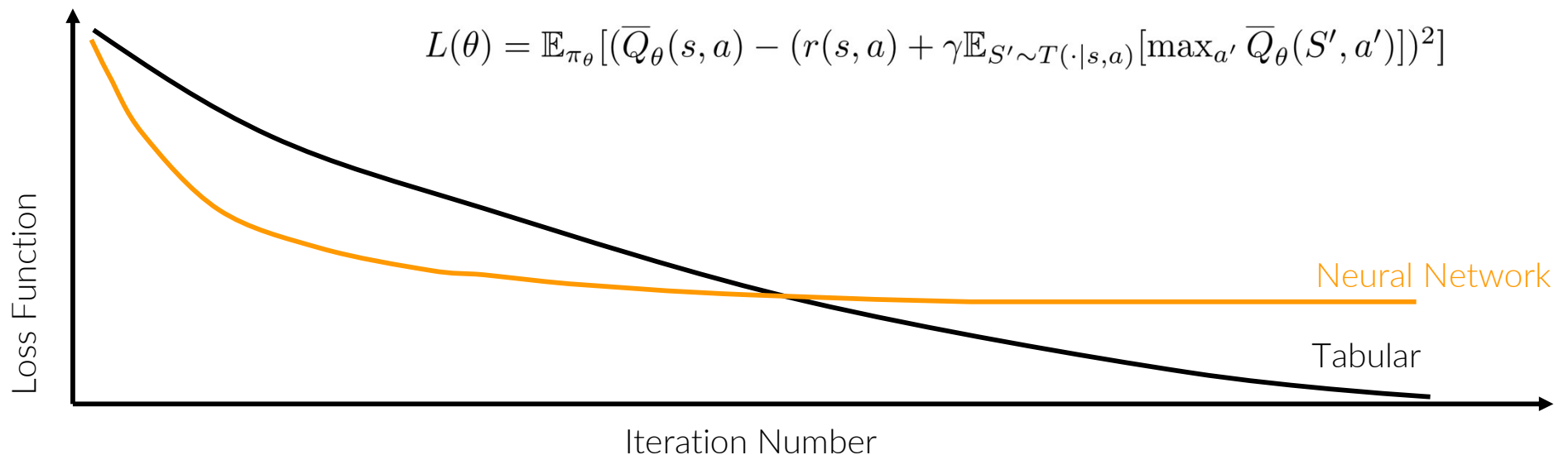
$$L(\theta) = \mathbb{E}_{\pi_{\theta}}[(\bar{Q}_{\theta}(s, a) - (r(s, a) + \gamma \mathbb{E}_{S' \sim T(\cdot|s, a)}[\max_{a'} \bar{Q}_{\theta}(S', a')]))^2]$$

DQN

Converges to true Q^* using tabular representation

Diverges in neural networks due to:

- spurious correlation in samples
- nonstationary target



Max vs Expectation

Other issue in practice:

$$L(\theta) = \frac{1}{N} \sum_i (\bar{Q}_\theta(s_i, a_i) - (r_i + \gamma \max_{a'} \bar{Q}_\theta(s_{i+1}, a')))^2$$

$\frac{1}{N} \sum_i \max_{a'} \bar{Q}_\theta(s_{i+1}, a')$ is biased for $\max_{a'} \mathbb{E}_{S' \sim T(\cdot|s, a')} [\bar{Q}_\theta(S', a')]$

Same value used both to select and evaluate an action, more likely to select overestimated values resulting in overoptimistic estimates

[Hessel2017]

DQN

Different versions of DQN fix this in two ways via:

Experience Replay

- Collects datasets which are recycled when calculating loss
- Eliminates spurious correlation in samples

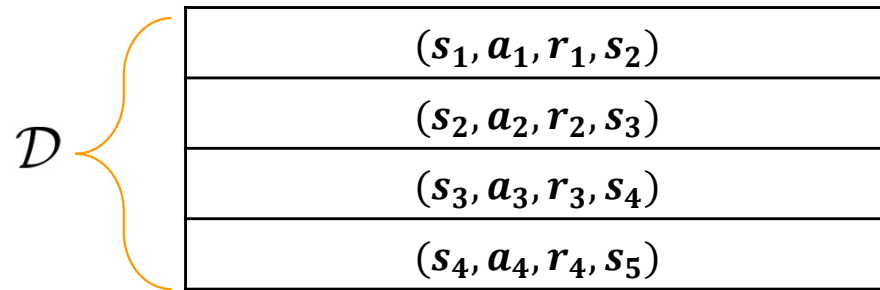
Fixed Q-Targets

- Fixes the target in the MSE loss
- Many choices of potential fixed targets

[Hessel2017]

Experience Replay

Store dataset (called **replay buffer**) from prior experience



The diagram shows a large curly brace on the left labeled \mathcal{D} pointing to a table with four rows. Each row contains a tuple of five elements: (s_1, a_1, r_1, s_2) , (s_2, a_2, r_2, s_3) , (s_3, a_3, r_3, s_4) , and (s_4, a_4, r_4, s_5) .

(s_1, a_1, r_1, s_2)
(s_2, a_2, r_2, s_3)
(s_3, a_3, r_3, s_4)
(s_4, a_4, r_4, s_5)

Sample batch on \mathcal{D} and update loss

$$L(\theta) = \frac{1}{N} \sum_i (\bar{Q}_\theta(s_i, a_i) - (r_i + \gamma \max_{a'} \bar{Q}_\theta(s_{i+1}, a'))^2$$

[Hessel2017]

Experience Replay

Sample batch on \mathcal{D} and update loss

$$L(\theta) = \frac{1}{N} \sum_i (\bar{Q}_\theta(s_i, a_i) - (r_i + \gamma \max_{a'} \bar{Q}_\theta(s_{i+1}, a'))^2$$

- Uniformly at random
- Weighted by loss
- Weighted by policy performance
-

Fixed Q Targets

Fix weights θ^- for target:

$$L(\theta) = \frac{1}{N} \sum_i (\bar{Q}_\theta(s_i, a_i) - (r_i + \gamma \max_{a'} \bar{Q}_{\theta^-}(s_{i+1}, a'))^2$$

Compute Q-learning targets with respect to “old”
parameters

[Hessel2017]

DQN

1. Use current policy (potentially with ϵ exploration) to collect data
2. Store transitions (s_t, a_t, r_t, s_{t+1}) in replay data \mathcal{D}
3. Sample batch of transitions from \mathcal{D}
4. Update loss with respect to **old** parameters using SGD

$$L(\theta) = \frac{1}{N} \sum_i (\bar{Q}_\theta(s_i, a_i) - (r_i + \gamma \max_{a'} \bar{Q}_\theta(s_{i+1}, a'))^2$$

[Hessel2017]

Double DQN

Same value used both to select and evaluate an action, more likely to select overestimated values resulting in overoptimistic estimates

Fix: Double DQN.

Learn two value functions with different parameters by assigning each data randomly to update one of the two estimates

[Hasselt2015]

Double DQN

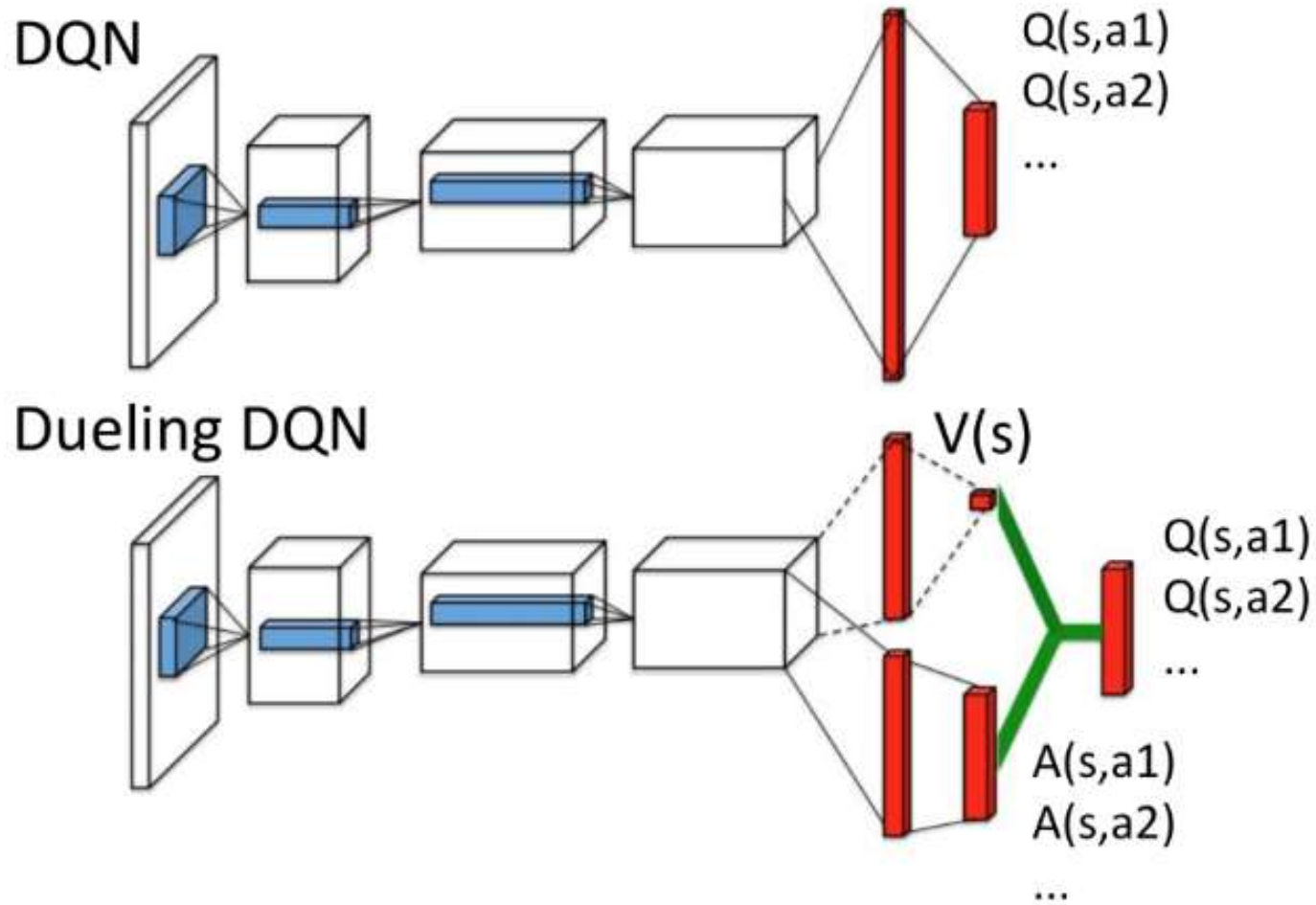
Two sets of weights θ_1, θ_2

1. Use current policy π_{θ_1} (potentially with ϵ exploration) to collect data
2. Store transitions (s_t, a_t, r_t, s_{t+1}) in replay data $\mathcal{D}_1, \mathcal{D}_2$
3. Sample batch of transitions from \mathcal{D}_i
4. Update loss with respect to **opposite** parameters using SGD

$$J(\theta) = \frac{1}{N} \sum_i (\bar{Q}_{\theta_1}(s_i, a_i) - (r_i + \gamma \max_{a'} \bar{Q}_{\theta_2}(s_{i+1}, a'))^2$$

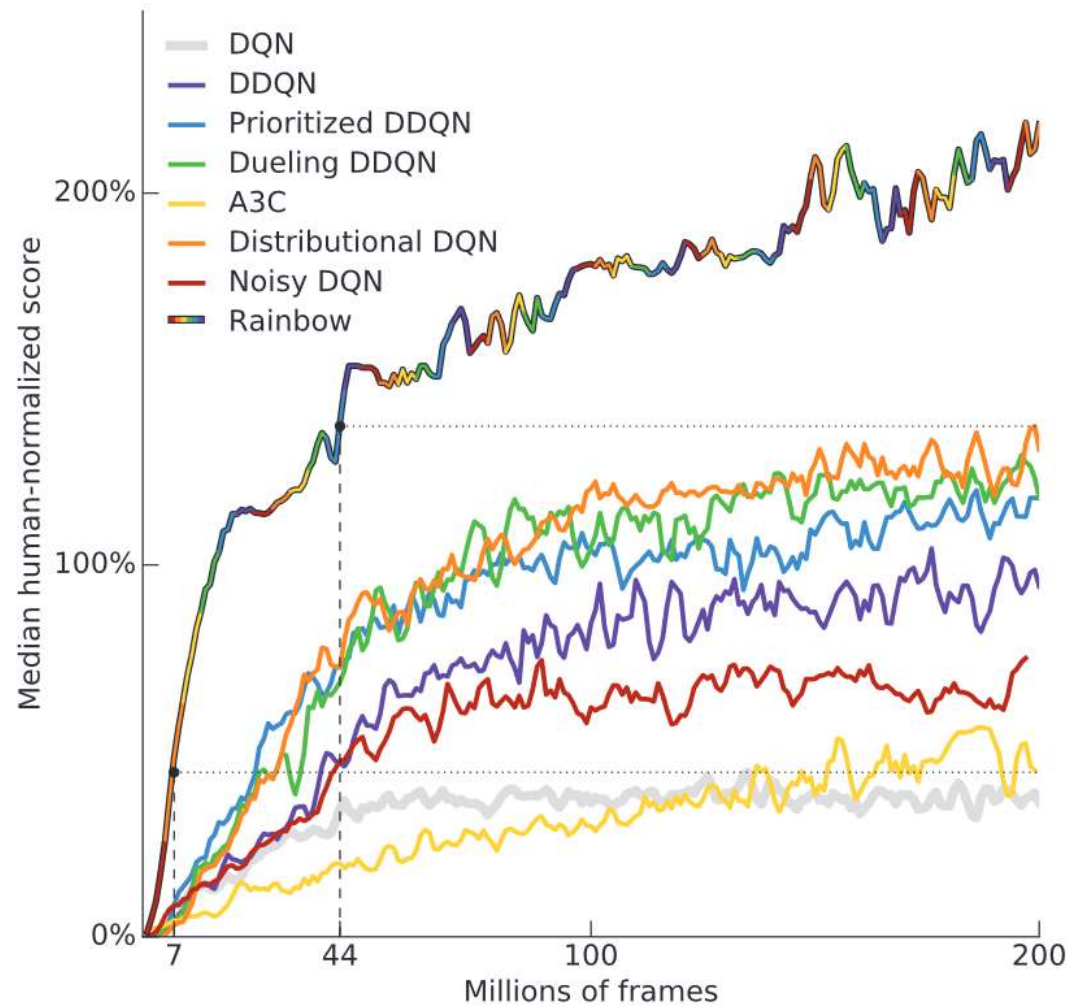
[Hasselt2015]

Dueling DQN



[Wang2015]

DQN+++++



[Hessel2017]

High level approach

Use a **deep neural network** to represent:

- Value function
- Policy
- Model of rewards and transitions
- “universal function approximator”

Optimize loss function via stochastic gradient descent (SGD)

DQN

- Use neural network representation of value function
- Fits estimates using least squares

REINFORCE / VPG

- Uses neural network representation of policy
- Fits estimates using policy gradient loss

High level approach

Focus on **REINFORCE** – simplest policy based algorithm. Works for on-policy and off-policy data, and in arbitrary domains.

Many flavors and modifications – similar to DQN

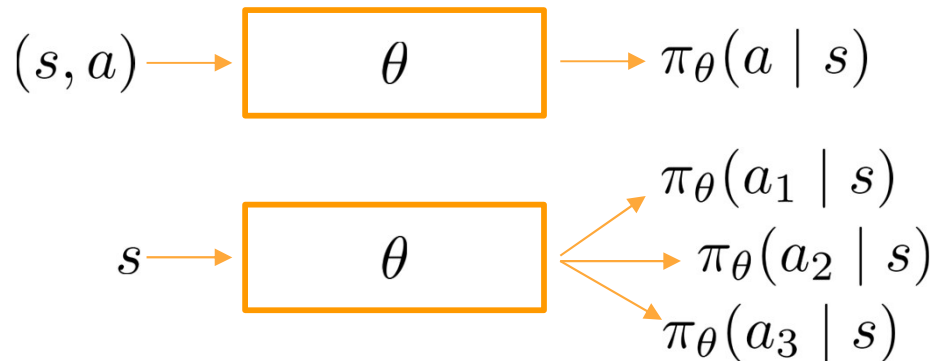
REINFORCE / VPG

- Uses neural network representation of policy
- Fits estimates using policy gradient loss

High level approach

Focus on **REINFORCE** – simplest policy based algorithm. Works for on-policy and off-policy data, and in arbitrary domains.

Many flavors and modifications – similar to DQN



REINFORCE / VPG

- Uses neural network representation of policy
- Fits estimates using policy gradient loss

High level approach

Focus on **REINFORCE** – simplest policy based algorithm. Works for on-policy and off-policy data, and in arbitrary domains.

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_{\theta}}(s_0)]$

High level approach

Focus on **REINFORCE** – simplest policy based algorithm. Works for on-policy and off-policy data, and in arbitrary domains.

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_{\theta}}(s_0)] = J(\theta)$

Refer to $J(\theta)$ as objective, can be modified for finite horizon + average cost

High level approach

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_{\theta}}(s_0)] = J(\theta)$

Any optimization algorithm could be applied:

- Zero-Order (Gradient-Free)
- First-Order (Gradient-Based)
- Second Order (Hessian-Based)

Gradient Free

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_\theta}(s_0)] = J(\theta)$

Evolutionary Algorithms or Grid Search

For $t = 0, 1, 2, \dots$

Sample: $\epsilon_1, \dots, \epsilon_N \sim N(0, I)$

Compute Returns: $F_i = J(\theta_t + \sigma \epsilon_i)$

Update: $\theta_{t+1} = \theta_t + \alpha \frac{1}{N\sigma} \sum_{i=1}^N F_i \epsilon_i$

Gradient Free

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_\theta}(s_0)] = J(\theta)$

Evolutionary Algorithms or Grid Search

For $t = 0, 1, 2, \dots$

Sample: $\epsilon_1, \dots, \epsilon_N \sim N(0, I)$

Compute Returns: $F_i = J(\theta_t + \sigma \epsilon_i)$

Update: $\theta_{t+1} = \theta_t + \alpha \frac{1}{N\sigma} \sum_{i=1}^N F_i \epsilon_i$

“Scalable” via strong parallelization, difficult for neural networks due to dimension

Policy Gradient

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_{\theta}}(s_0)] = J(\theta)$

What even is $\nabla J(\theta)$?

Policy Gradient

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_\theta}(s_0)] = J(\theta)$

What even is $\nabla J(\theta)$?

Policy Gradient Theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a | s) Q^{\pi_\theta}(s, a)]$$

Can evaluate using Monte-Carlo roll outs under current policy

[Williams1992]

REINFORCE

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_{\theta}}(s_0)] = J(\theta)$

For each episode

Sample: $(s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T) \sim \pi_{\theta}$

Compute Returns: $G_t = \sum_{\tau \geq t} \gamma^{\tau-t} r_{\tau}$

Update: $\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} \log \pi_{\theta}(s_{\tau} | a_{\tau}) G_{\tau} \quad \tau = 1, \dots, T$

Policy Gradient

Goal: Maximize $\sup_{\theta \in \Theta} \mathbb{E}[V^{\pi_\theta}(s_0)] = J(\theta)$

What even is $\nabla J(\theta)$?

Policy Gradient Theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a | s) Q^{\pi_\theta}(s, a)]$$

Two issues:


- Variance in gradient due to log probabilities
- “Double Expectation”

[Williams1992]

Double Expectation

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi_{\theta}}(s, a)]$$

Expectation over
trajectories



Expectation over
policy



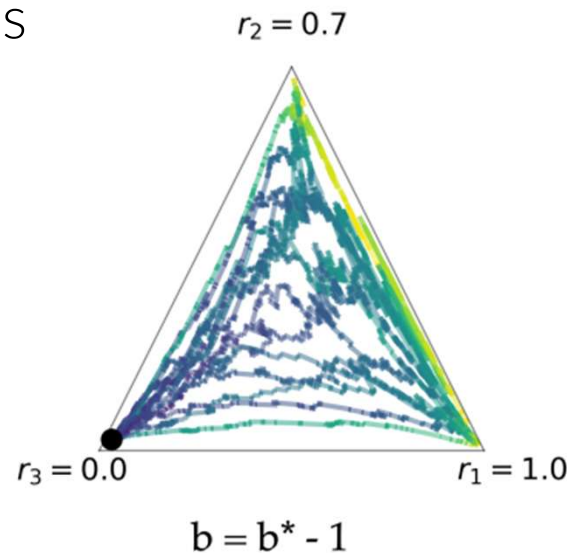
Can plug in a variety of terms into $Q^{\pi_{\theta}}$, similar to DQN

Variance Issues

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi_{\theta}}(s, a)]$$

Biggest Issue: Variance in policy gradient objective due to:

- Credit assignment
- Log probabilities (opt policy is deterministic...)
- Different distribution of trajectory when applying updates using batches



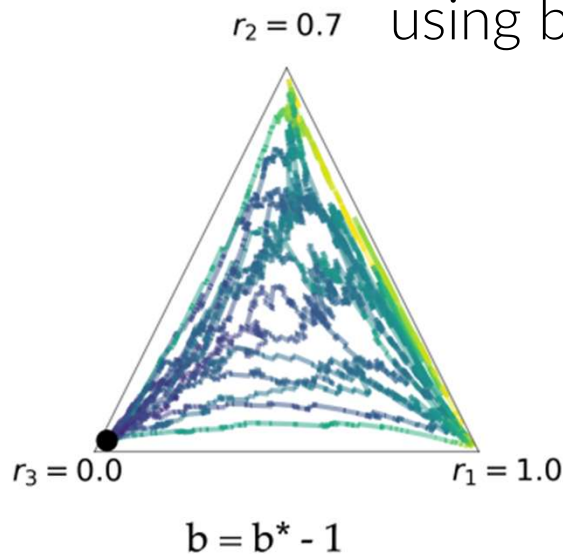
[Machado2020]

Variance Issues

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi_{\theta}}(s, a)]$$

Biggest Issue: Variance in policy gradient objective due to:

- Credit assignment
- Log probabilities (opt policy is deterministic...)
- Different distribution of trajectory when applying updates using batches



Fix One: Clipping probabilities to $[\epsilon, 1 - \epsilon]$

Fix Two: State dependent baseline, i.e.

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) (Q^{\pi_{\theta}}(s, a) - b(s))]$$

Form of control variate

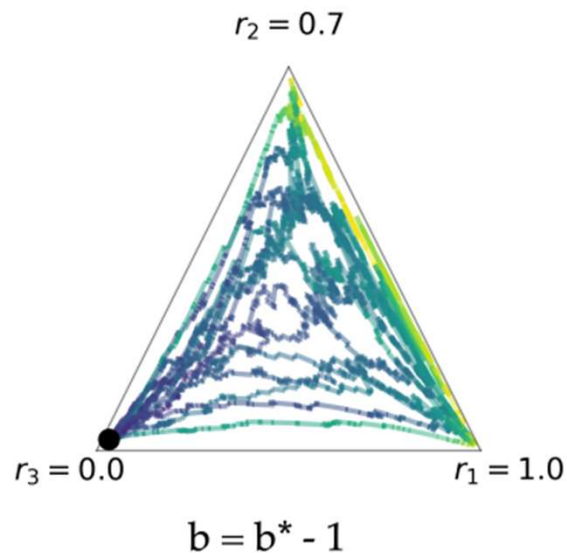
Variance Issues

Fix One: Clipping probabilities to $[\epsilon, 1 - \epsilon]$

Fix Two: State dependent baseline, i.e.

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) (Q^{\pi_{\theta}}(s, a) - b(s))]$$

Form of control variate



Typically use: $\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) (Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s))]$

$A^{\pi_{\theta}}(s, a)$

Advantage Function

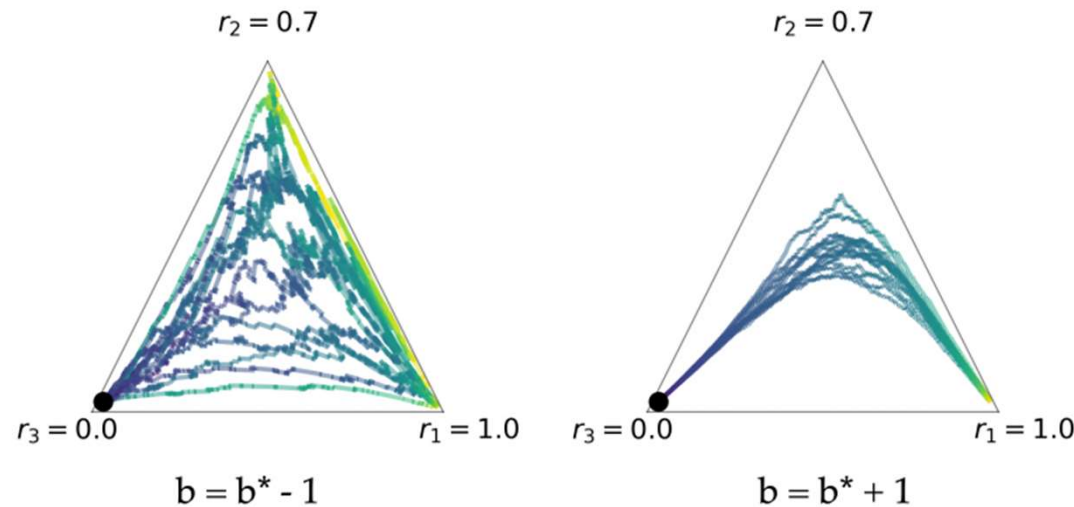
[Machado2020]

Variance Issues

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi_{\theta}}(s, a)]$$

Biggest Issue: Variance in policy gradient objective due to:

- Credit assignment
- Log probabilities (opt policy is deterministic...)
- Different distribution of trajectory when applying updates using batches



[Machado2020]

PG+++

Per usual, many flavours and modifications with improved empirical performance.

TRPO: add on KL divergence constraint on policy updates

PPO: add on clipping of the ratio with KL regularization

[Schulman2015, Schulman2017]

High level approach

Use a **deep neural network** to represent:

- Value function
- Policy
- Model of rewards and transitions
- “universal function approximator”

Optimize loss function via stochastic gradient descent (SGD)

DQN

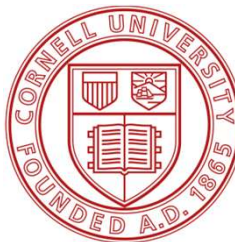
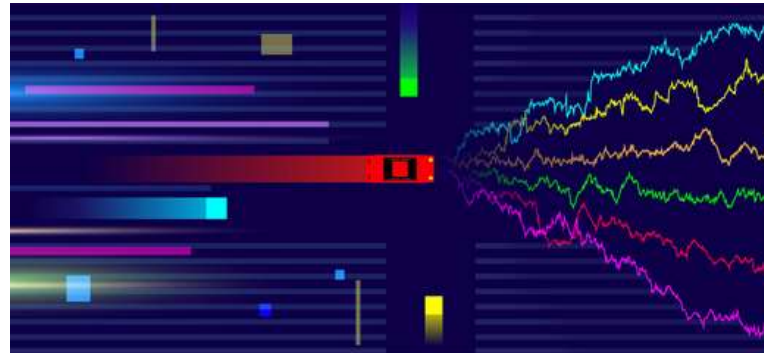
- Use neural network representation of value function
- Fits estimates using least squares

REINFORCE / VPG

- Uses neural network representation of policy
- Fits estimates using policy gradient loss

Introduction to Deep RL

Sean Sinclair
Cornell University



References

- [Machado2020] Marlos C. Machado. “[The True Impact of Baselines in Policy Gradient Methods](#).” *Blog Post*, 2020.
- [Schulman2015] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel “[Trust Region Policy Optimization](#)”. *ICML*, 2020.
- [Hessel2017] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver “[Rainbow: Combining Improvements in Deep Reinforcement Learning](#)”. *AAAI*, 2018.
- [Schulman2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. “[Proximal Policy Optimization Algorithms](#)”. *ArXiv*, 2017.
- [Mnih2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. “[Playing Atari with Deep Reinforcement Learning](#)”. *ArXiv*, 2013.

References

- [Silver2017] David Silver et al. "[Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.](#)" *ArXiv*, 2017.
- [Tesauro1995] Gerald Tesauro. "[Temporal Difference Learning and TD-Gammon](#)". *Communications of the ACM*, 1995.
- [Hasselt2015] Hado Van Hasselt, Arthur Guez, David Silver. "[Deep Reinforcement Learning with Double Q-Learning](#)". *AAAI*, 2016.
- [Wang2015] Ziju Wang et al. "[Dueling Network Architectures for Deep Reinforcement Learning.](#)" *ArXiv*, 2015.
- [Williams1992] Ronald J Williams. "[Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.](#)" *Machine Learning*, 1992.
- [Vikhar2016] Pradnya A Vikhar. "[Evolutionary algorithms: A Critical Review and its Future Prospects.](#)" *NeurIPS*, 2016.

References

[Salimans2017] Tim Salimans et al. "[Evolution Strategies as a Scalable Alternative to Reinforcement Learning.](#)" *NeurIPS*, 2017.