

DS-GA 3001 007 | Lecture 5

Reinforcement Learning

Jeremy Curuksu, PhD

NYU Center for Data Science

jeremy.cur@nyu.edu

March 4, 2023

DS-GA 3001 RL Curriculum

Reinforcement Learning:

- ▶ Introduction to Reinforcement Learning
- ▶ Multi-armed Bandit
- ▶ Dynamic Programming on Markov Decision Process
- ▶ Model-free Reinforcement Learning
- ▶ **Value Function Approximation (Deep RL)**
- ▶ Policy Function Approximation (Actor-Critic)
- ▶ Planning from a Model of the Environment
- ▶ Examples of Industrial Applications
- ▶ Advanced Topics and Development Platforms

Reinforcement Learning

Last week: Model Free RL

- ▶ Monte Carlo and Temporal Difference
- ▶ Sample based Prediction and Control
- ▶ Off policy Learning

Today: Function Approximation and Deep RL

- ▶ Categories of Functions
- ▶ Approximation of State Update and Value Functions
- ▶ Deep Reinforcement Learning

Categories of Functions in Reinforcement Learning

RL Functions and Approximations

- ▶ All key components of an RL agent are functions
 - ▶ State update functions map observations to states
 - ▶ Value functions map states to values
 - ▶ Policies map states to actions
 - ▶ Models map states and actions to next states and rewards
- ▶ Functions can be parameterized and approximated by linear or non-linear gradient methods e.g., Deep Learning
- ▶ If approximated, this is called RL with function approximation
- ▶ Challenge: Supervised learning assumptions are often violated
- ▶ Deep reinforcement learning has recently lead to major breakthroughs in AI and is an active field of research

Why Approximate?

Tabular methods do not scale to large state spaces

- ▶ So far we defined value functions as **lookup tables**, where every state s has an entry $v(s)$, or set of $q(s, a)$
- ▶ Many RL problems contain a very large number of states:
 - ▶ Chess: 10^{120} states
 - ▶ Go: 10^{170} states
 - ▶ Helicopter: Continuous state space
 - ▶ Robot: Infinite state space (physical universe)
- ▶ Problem with large MDPs:
 - ▶ There are too many states/actions to store in memory
 - ▶ Learning the value of each state individually is too slow
 - ▶ Individual states are often not fully observable
 - ▶ Exact nature of a state may never happen again

Types of Function Approximation

Any function approximator can be used:

- ▶ Tabular function (look-up tables)
- ▶ State aggregation (feature engineering)
- ▶ Linear function (regression, nearest-neighbors, etc)
- ▶ Non-linear function (neural network, decision tree, etc)

But RL has specific properties:

- ▶ Experience is not i.i.d. — successive time-steps are correlated
- ▶ Agent's policy affects the data it observes
- ▶ Reward and values may continuously evolve (non-stationary)
- ▶ Feedback is delayed, not instantaneous

Approximation of State Update Functions

Generalization through state space

Function to update state and engineer state features

$$s_{t+1} = \phi(s_t, o_{t+1})$$

- ▶ Represent state by a feature vector. For example:
 - ▶ GPS location and sonar readings (distance to objects) of a robot
 - ▶ Recent trends in the stock market
 - ▶ Global configuration of the board in Chess
- ▶ If the environment state is not fully observable, there is at least an implicit mapping $o_t \mapsto s_t$

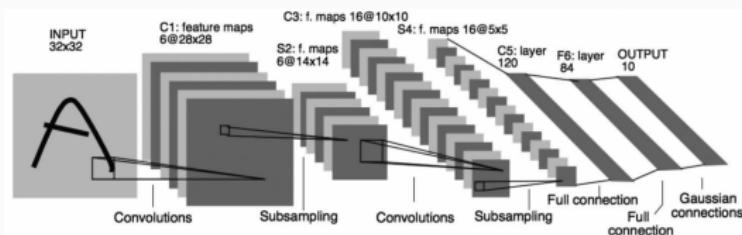


Generalization through state space

Function to update state and engineer state features

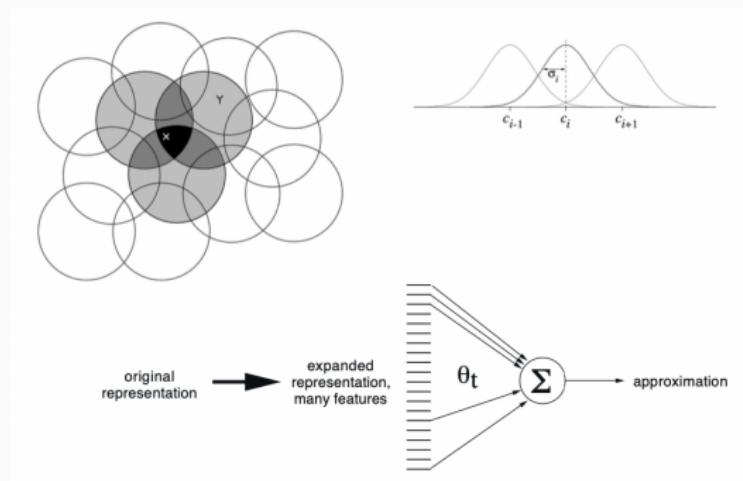
$$s_{t+1} = \phi(s_t, o_{t+1})$$

- ▶ Represent state by a feature vector. For example:
 - ▶ GPS location and sonar readings (distance to objects) of a robot
 - ▶ Recent trends in the stock market
 - ▶ Piece configurations in Chess
- ▶ If the environment state is not fully observable, there is at least an implicit mapping $o_t \mapsto s_t$



Example of State Feature Engineering

Aggregate multiple states by coarse coding:



- ▶ Generalise from seen states to unseen states
- ▶ Resulting features not Markovian \Rightarrow **Partially Observable MDP**

Approximation of Value Prediction Functions

Linear Value Function Approximation

Approximate Values using a Parametric Function

$$v_w(s) \approx v_\pi(s)$$

- ▶ $v_w(s)$ can be a linear combination of state features:

$$v_w(s) = w^T \phi(s) = \sum_{i=1}^n w_i \phi_i(s)$$

- ▶ Parameters w_i can be updated incrementally by optimizing an objective measure of performance:

$$w \sim \arg \min_w \mathbb{E}_{\pi}[(v_\pi(s) - v_w(s))^2]$$

- ▶ Gradient descent can converge to global optimum if v_π known

Linear Value Function Approximation

The special case of tabular value function

- ▶ Define $v_w(s)$ as a linear combination of state features:

$$v_w(s) = w^T \phi(s) = \sum_{i=1}^n w_i \phi_i(s) = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}^T \begin{bmatrix} \phi_1(s) \\ \vdots \\ \phi_n(s) \end{bmatrix}$$

where n is the number of features

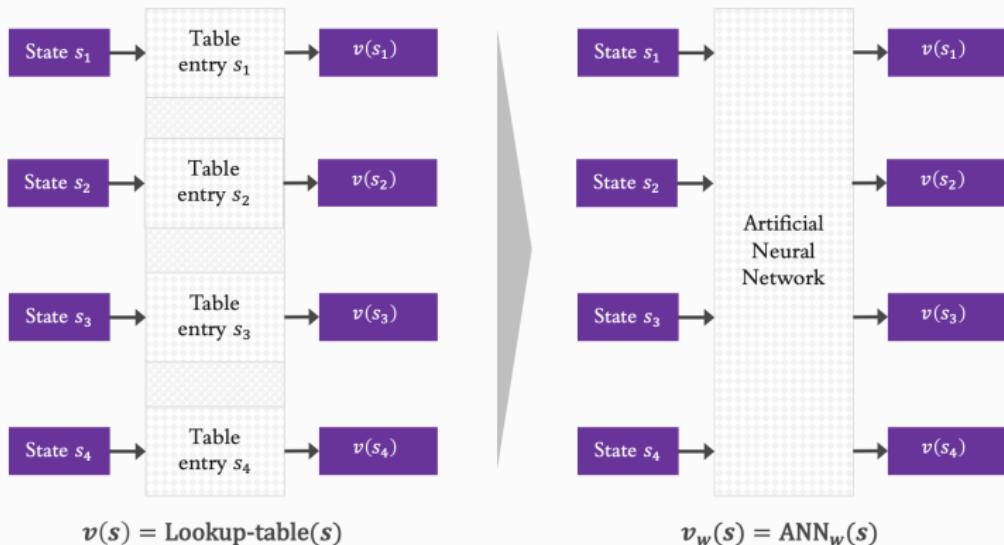
- ▶ Tabular value function (lookup table) is the special case where $n = \text{number of states}$ and where w_i is the value of each state s_i :

$$v_w(s) = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}^T \begin{bmatrix} 1(s = s_1) \\ \vdots \\ 1(s = s_n) \end{bmatrix}$$

Non-linear Value Function Approximation

Approximate Values using a Deep Neural Network

- Deep RL: Replace the look-up table by an ANN



Function Optimization by Gradient Descent

Approximate Values by Stochastic Gradient Descent

- **Goal:** Find w that minimizes difference between $v_w(s)$ and $v_\pi(s)$:

$$J(w) = \mathbb{E}_{\pi}[(v_\pi(s) - v_w(s))^2]$$

- To find a minimum of $J(w)$, define the gradient of $J(w)$ and move w_i in the direction of negative gradient at every step:

$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix}$$

$$w_{t+1} = w_t - \frac{1}{2}\alpha \nabla_w (v_\pi(s_t) - v_w(s_t))^2$$

$$w_{t+1} = w_t - \alpha (v_\pi(s_t) - v_w(s_t)) \nabla_w v_w(s_t)$$

Incremental Prediction RL Algorithms

Replace the true value of $v_\pi(s)$ by sampled target

- ▶ In practice we do not know the true value $v_\pi(s)$, we replace it by a target sampled using Monte-Carlo or Temporal Difference:
- ▶ MC value function approximation target is the return G_t

$$w_{t+1} = w_t - \alpha (G_t - v_w(s_t)) \nabla_w v_w(s_t)$$

- ▶ TD value function approximation target is $r_{t+1} + \gamma v_w(s_{t+1})$

$$w_{t+1} = w_t - \alpha (r_{t+1} + \gamma v_w(s_{t+1}) - v_w(s_t)) \nabla_w v_w(s_t)$$

Monte-Carlo Value Function Approximation

Monte-Carlo Policy Evaluation

- ▶ The return G_t is an unbiased, noisy sample of $v_\pi(s)$
- ▶ Apply supervised learning to sampled data $\{(s_0, G_0), \dots, (s_t, G_t)\}$

$$w_{t+1} = w_t - \alpha (G_t - v_w(s_t)) \nabla_w v_w(s_t)$$

- ▶ Monte-Carlo evaluation converges to a local optimum at least (proof out of scope)
- ▶ With a linear function approximator, it simplifies to:

$$w_{t+1} = w_t - \alpha (G_t - v_w(s_t)) \phi(s_t)$$

TD(o) Value Function Approximation

Temporal Difference Policy Evaluation

- ▶ The TD target $r_{t+1} + \gamma v_w(s_{t+1})$ is a biased sample of $v_\pi(s)$
- ▶ Supervised learning can still be applied to sampled data which takes the form $\{(s_0, r_1 + \gamma v_w(s_1)), \dots, (s_t, r_{t+1} + \gamma v_w(s_{t+1}))\}$

$$w_{t+1} = w_t - \alpha (r_{t+1} + \gamma v_w(s_{t+1}) - v_w(s_t)) \nabla_w v_w(s_t)$$

- ▶ Temporal Difference evaluation converges to a local optimum which may not be the same as the asymptotic MC solution
- ▶ MC asymptotic solution unbiased, but TD may converge faster

Human-Level Control with Deep Reinforcement Learning

Control with Value Function Approximation

Approximate action values using a parametric function

$$q_w(s, a) \approx q_\pi(s, a)$$

- ▶ Find w that minimizes difference between $q_w(s, a)$ and $q_\pi(s, a)$ or $q_*(s, a)$ by stochastic gradient descent:

$$w_{t+1} = w_t - \alpha (q_*(s_t, a_t) - q_w(s_t, a_t)) \nabla_w q_w(s_t, a_t)$$

- ▶ Example: Sample $q_*(s, a)$ as TD target $r_{t+1} + \gamma \max_{a'} q_w(s_{t+1}, a')$ and apply supervised learning while acting on ϵ -greedy policy:

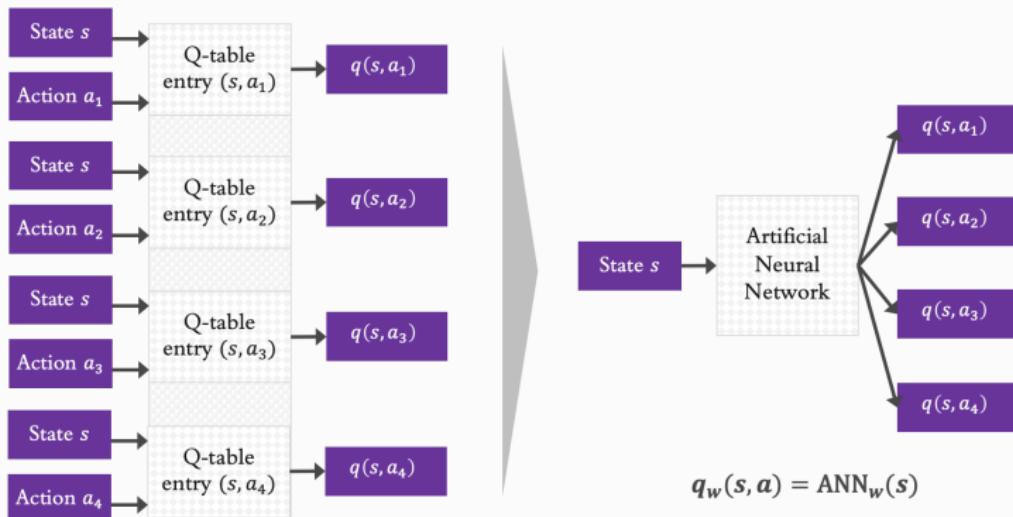
$$w_{t+1} = w_t - \alpha \left(r_{t+1} + \gamma \max_{a'} q_w(s_{t+1}, a') - q_w(s_t, a_t) \right) \nabla_w q_w(s_t, a_t)$$

If $q_w(s, a)$ is a neural network, this is called **Neural Q-Learning**

Control with Neural Q-Learning

Approximate action-values using a Deep Neural Network

- Deep RL: Replace look-up table by an ANN



$$q(s, a) = \text{Q-table}(s, a)$$

Practice: Semi-Gradient TD Control

Semi-Gradient TD Control predicts $q(s, a)$ using the same parametric function at every step with no model of the environment

Initialize w , $\pi(s)$ and s arbitrarily

Loop forever:

Select and take a from s following $\pi(s)$, observe r_{t+1} and s'

$$w = w - \alpha \left(r_{t+1} + \gamma \max_{a'} q_w(s', a') - q_w(s, a) \right) \nabla_w q_w(s, a)$$

Update π at s given $q_w(s, a)$ by ϵ -greedy soft update

$$s = s'$$

Convergence of Semi-Gradient TD Control

Semi-Gradient TD Control combines bootstrapping, off-policy learning, and value approximation

- ▶ Training data used to parameterize the predictive function is gathered online and thus often non-stationary/not i.i.d.
- ▶ The environment itself may continuously evolve, or be too big for exhaustive sampling, and thus may never reach equilibrium
- ▶ **Tracking is often preferred to convergence: continually adapt the policy instead of trying to converge to a fixed policy**
- ▶ **Theory of control with function approximation still unclear**

Challenges with Semi-Gradient TD Control

Deadly triad: Convergence not guaranteed if we combine these 3 methods together, even with infinite sampling

Bootstrapping

Learn from estimates in neighbor states

Pros: Faster learning and data efficient

Cons: Bias in estimate of target return



Function Approximation

Learn from function generalizing across state space

Pros: Scale to large problems

Cons: Converges iff sampling unbiased estimate of G_t or stationary distribution of states under target policy

Off-Policy

Learn about target policy by following behavior policy

Pros: Learn target behavior from alter streams of experiences

Cons: Bias due to mismatch between expected vs. sampled distribution of states

Improving Quality of RL approximations

Reduce correlation between predicted vs. target values
to "makes the training data more i.i.d."

- ▶ **Experience Replay:** Keep set of past experiences in memory, find best fitting value function in randomized training batches
- ▶ **Add a Target Network:** Use two separate parametric functions, one updated at every step, and one updated periodically which predicts the TD target (frozen within each *temporal period*)
- ▶ **Emphatic TD method:** Weight down states observed only in the behavior policy and weight up states observed in target policy (beyond scope)

Practice: Deep Q-Network (DQN)

DQN predicts $q(s, a)$ with two Neural Networks to learn at every step from stable buffers of experiences, with no model of environment

Initialize \mathbf{w} , $\mathbf{w}^{\text{target}}$, $\pi(s)$ and s arbitrarily

Loop forever:

Select and take a from s following $\pi(s)$, observe r_{t+1} and s'

$$\mathbf{w} = \mathbf{w} - \alpha \left(r_{t+1} + \gamma \max_{a'} q_w^{\text{target}}(s', a') - q_w(s, a) \right) \nabla_{\mathbf{w}} q_w(s, a)$$

Update π at s given $q_w(s, a)$ by ϵ -greedy soft update

Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in a Replay Buffer

$s = s'$

Periodically:

Sample mini-batches from the Replay Buffer

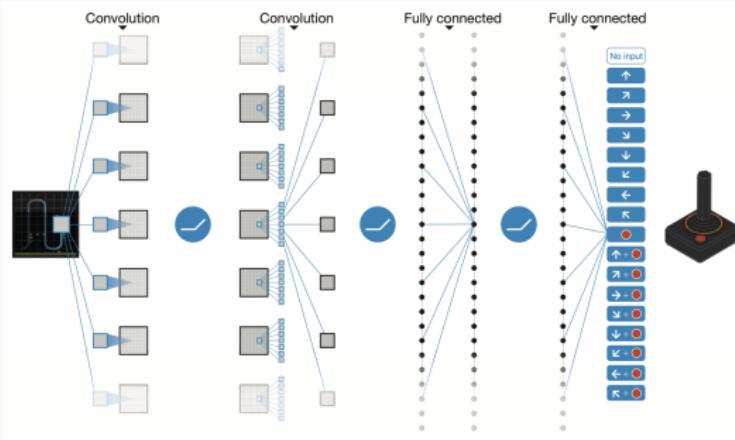
Loop through these mini-batches to further update \mathbf{w}

Update $\mathbf{w}^{\text{target}} \leftarrow \mathbf{w}$

Deep RL Research

Novelty of DQN (Mnih et al., Nature 2015)

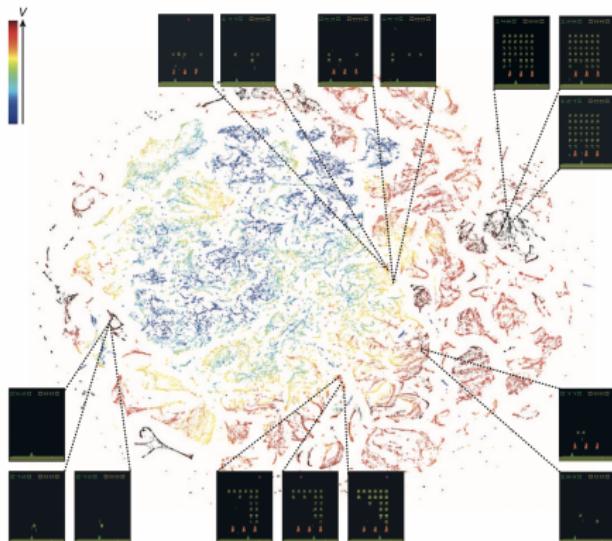
1. **Convolutional Neural Network** creating states from raw pixels
2. **Experience Replay buffer** storing set of past experiences $e_i = (s_i, a_i, r_i, s_{i+1})$ from which mini-batches are randomly sampled
3. **Target network** updated periodically (frozen between updates)



Deep RL Research

Perceptually similar states are mapped to nearby points in the high-dimensional CNN embedding space

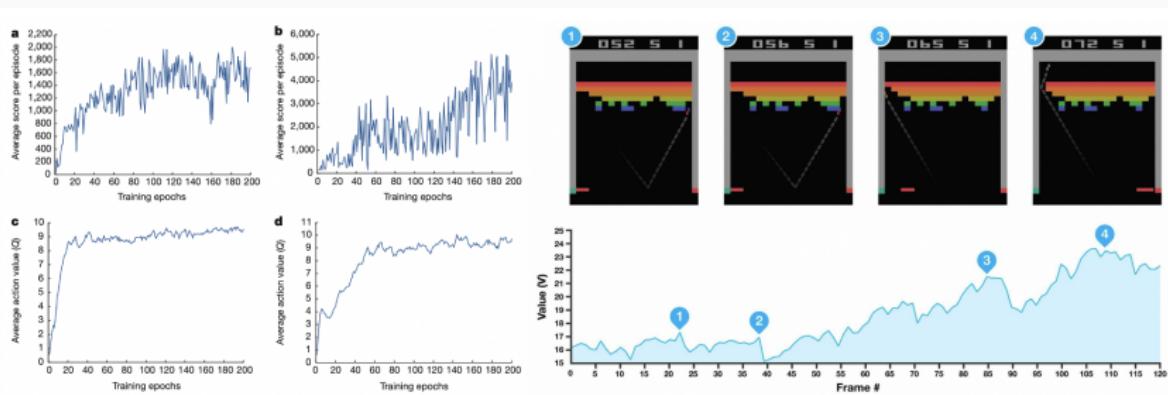
- ▶ 2D t-SNE representations in last CNN layer assigned to Space Invaders states



Deep RL Research

DQN can train a large ANN with stochastic gradient descent in a stable manner to identify winning actions

- ▶ Temporal evolution of average score-per-episode and predicted Q-values



Mnih et al., Nature 2015

Deep RL Research

Disabling DQN's replay memory and/or separate target Q-network has a detrimental effect on performance

- ▶ Effects of Experience Replay and Separate Target Q-Network on performance

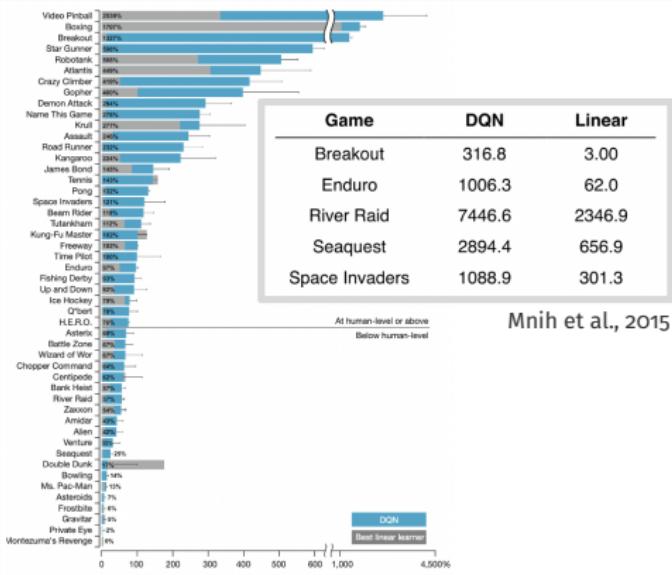
| Game | With replay, with target Q | With replay, without target Q | Without replay, with target Q | Without replay, without target Q |
|----------------|-------------------------------|----------------------------------|----------------------------------|-------------------------------------|
| Breakout | 316.8 | 240.7 | 10.2 | 3.2 |
| Enduro | 1006.3 | 831.4 | 141.9 | 29.1 |
| River Raid | 7446.6 | 4102.8 | 2867.7 | 1453.0 |
| Seaquest | 2894.4 | 822.6 | 1003.0 | 275.8 |
| Space Invaders | 1088.9 | 826.3 | 373.2 | 302.0 |

Mnih et al., Nature 2015

Deep RL Research

DQN outperforms all linear methods and achieve level comparable to that of professional human players

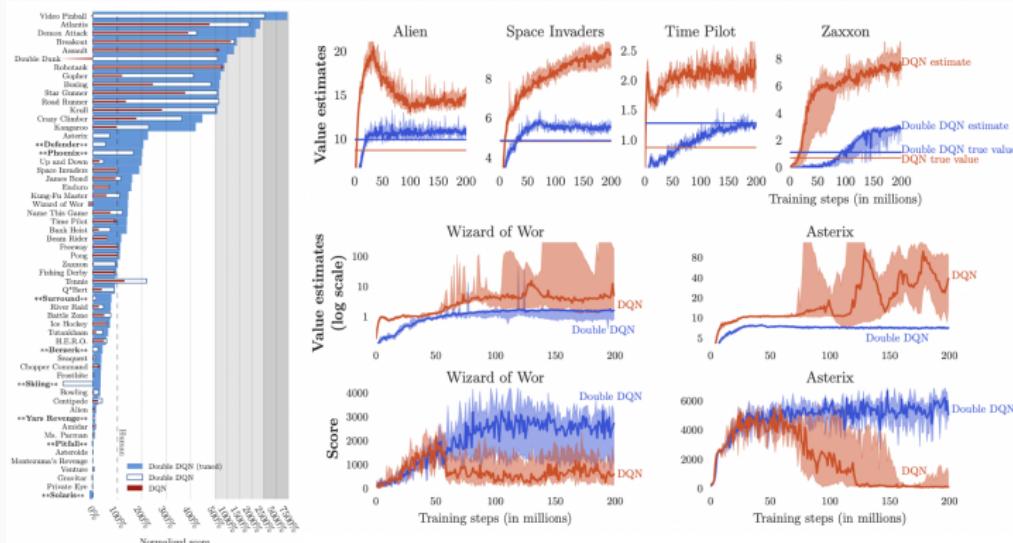
- ▶ Comparison of DQN performance vs. best RL from 2015 on 49 Atari video games



Deep RL Research

DDQN outperforms DQN by reducing the overoptimism due to value estimation errors

- ▶ Comparison of DDQN performance vs. DQN on 57 Atari video games



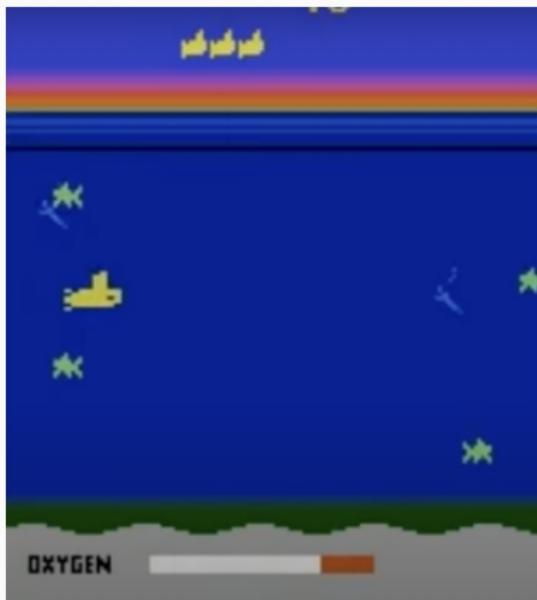
Deep RL Research

Deep RL is a fertile research area

- ▶ Performance has improved dramatically in past few years:
 - ▶ Deep Q-Network (2014)
 - ▶ Deepmind AlphaGo (2016)
 - ▶ Deepmind AlphaZero (2018)
 - ▶ OpenAI ChatGPT (2022)
- ▶ Some key open questions: (lectures 6-9)
 - ▶ Can we directly parameterize and optimize a policy "actor" function?
Does direct policy optimization benefit from also learning values?
 - ▶ Can we learn a model of the environment? How best to use a model?
 - ▶ How best to build the agent state? (including what it stores in memory)
 - ▶ How best to build objective measures to optimize function parameters?
 - ▶ How best to improve data sampling efficiency?

DQN learns to play Atari

Deep Q-network training on video game Seaquest



(Source: Sprague N., 2015)

Thank you!