# Reinforcement Learning
# Lecture 6

Georg Martius

Distributed Intelligence / Autonomous Learning Group, Uni Tübingen, Germany

November 19, 2024

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

# Overview

- Last time: Multiarmed Bandits
  - RL without states
  - Optimal exploration in noisy reward environments
- Today: RL with function approximation
  - Value function approximation
- Next time: policy gradient

# Value Function Approximation

Slides adapted from David Silver, Material from Sutton & Barto's RL book

# Plan for today

1. Introduction
2. Incremental Methods
3. Batch Methods

# Large-Scale Reinforcement Learning

What is the problem with the tabular case?

- State spaces are **exponentially large** for many interesting problems
  - Backgammon: $10^{20}$ states
  - Go (Board Game): $10^{170}$
- Or the state space is **continuous**
  - Robot control

What to do?

- **Approximate the value function** by a function approximator.

# Value Function Approximation

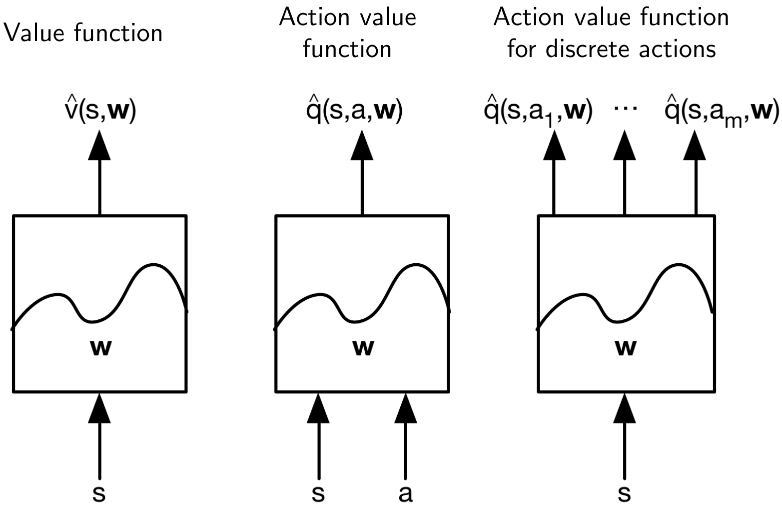
- The tabular value function
  - Every state $s$ has an entry $V(s)$
  - Or every state-action pair $s, a$ has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are **too many states** and/or actions to store in memory
  - It is **too slow to learn** the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - **Generalize** from seen states to unseen states
  - **Update** parameter $\mathbf{w}$ using MC or TD learning
  - Notation: approximations denoted with "hat"

# Types of Value Function Approximation



# Which Function Approximator?

There are many function approximators, for instance:

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbors
- Fourier / wavelet bases
- . . .

# Which Function Approximator?

Differentiable function approximators will be handy:

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbors
- Fourier / wavelet bases
- ...

the value changes with policy,
so it is a moving target (not stationary)

We require a training method that is suitable for non-stationary, non-iid (not identically and independently distributed) data

Suitable learning algorithm: (Stochastic) Gradient Descent

# Value Function Approximation
## using supervised learning

- Value function parametrized by parameters $\mathbf{w}$: $\hat{v}(s, \mathbf{w})$
- Goal: find parameter vector $\mathbf{w}$ minimizing difference between approximate value function $\hat{v}(s, \mathbf{w})$ and true value function $v_\pi(s)$

$$L(\mathbf{w}) = \mathbb{E}_\pi \left[ \left( v_\pi(S) - \hat{v}(S, \mathbf{w}) \right)^2 \right]$$

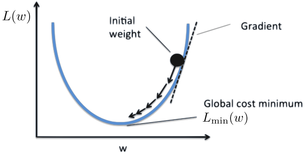
- Loss function:

$$L(\mathbf{w}) = \mathbb{E}_\pi \left[ \left( v_\pi(S) - \hat{v}(S, \mathbf{w}) \right)^2 \right]$$

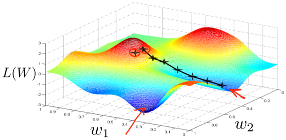
- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_\mathbf{w} L(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[ \left( v_\pi(S) - \hat{v}(S, \mathbf{w}) \right) \nabla_\mathbf{w} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

For convex functions:




The general case:




[Image: hackernoon.com]

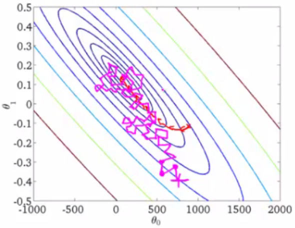
- General loss:

$$L(\mathbf{w}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[ \mathrm{d}\left( \underbrace{y}_{\text{target}}, \underbrace{\hat{y}(x)}_{\text{prediction}} \right)\right]$$

- Loss is expected empirical error: sum over examples (batch)
- SGD: update parameters on every example:

$$\Delta\mathbf{w} = -\alpha \xcancel{\sum_i^N} \nabla_{\mathbf{w}}\, \mathrm{d}\left(y^{(i)}, \hat{y}^{(i)}\right)$$

  expected update is equal to full gradient update
- Minibatches:
  average gradient over a small # of examples




instead of averaging gradient over all samples,
do it in small batches

# Feature Vectors

Represent state by a feature vector

$$\mathbf{x}(S) = \begin{pmatrix} x_1(S) \\ x_2(S) \\ \vdots \\ x_n(S) \end{pmatrix}$$

For example:

- Distance of robot from landmarks
- Trends in the stock market
- Piece and pawn configurations in chess

# Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$L(\mathbf{w}) = \mathbb{E}_\pi \left[ \left( v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w} \right)^2 \right]$$

- Stochastic gradient descent converges on global optimum
  Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$
$$\Delta \mathbf{w} = \alpha \left( v_\pi(S) - \hat{v}(S, \mathbf{w}) \right) \mathbf{x}(S)$$

- Update $=$ step-size $\times$ prediction error $\times$ feature value

# One-hot features = Tabular representation

Given a one-hot feature vector

$$\mathbf{x}(S) = \begin{pmatrix} [\![S = s_1]\!] \\ \vdots \\ [\![S = s_n]\!] \end{pmatrix}$$

if in state s1:
x(S) = (1, 0, ..., 0)

$\mathbf{x}(S)^\top \mathbf{w}$ yields the individual "table entries"

## Iterative Prediction Algorithms

In contrast to supervised learning, we do not have access to $v_\pi(s)$

- In RL: only rewards are given
- In practice, we substitute a target for $v_\pi(s)$
  - For MC, use the return $G_t$

$$\Delta \mathbf{w} = \alpha \left(G_t - \hat{v}\left(S_t, \mathbf{w}\right)\right) \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$$

  - For TD(0): use TD-target $R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right)$

$$\Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right) - \hat{v}\left(S_t, \mathbf{w}\right)\right) \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$$

  - For TD($\lambda$):use $\lambda$-return $G_t^\lambda$

$$\Delta \mathbf{w} = \alpha \left(G_t^\lambda - \hat{v}\left(S_t, \mathbf{w}\right)\right) \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$$

- as before the TD-target is biased (we use our own estimate)
- Note: For TD: target is not independent of $\mathbf{w}$.
  Semi-gradient methods: do not necessarily converge

# Iterative Prediction Algorithm
## for Linear Features

Consider the case of **linear features**: $\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^\top \mathbf{w}$:

- The derivative $\nabla_{\mathbf{w}} \hat{v}\,(s, \mathbf{w})$ is simply $\mathbf{x}(s)$
- MC:

$$\begin{aligned}\Delta \mathbf{w} &= \alpha\,(G_t - \hat{v}\,(S_t, \mathbf{w}))\,\nabla_{\mathbf{w}} \hat{v}\,(S_t, \mathbf{w}) \\ &= \alpha\,(G_t - \hat{v}\,(S_t, \mathbf{w}))\,\mathbf{x}(S_t)\end{aligned}$$

- TD(0):

$$\begin{aligned}\Delta \mathbf{w} &= \alpha\,\underbrace{(R_{t+1} + \gamma \hat{v}\,(S_{t+1}, \mathbf{w}) - \hat{v}\,(S_t, \mathbf{w}))}_{\delta}\,\nabla_{\mathbf{w}} \hat{v}\,(S_t, \mathbf{w}) \\ &= \alpha \delta_t \mathbf{x}(S_t)\end{aligned}$$

- TD($\lambda$): Forward view: $\Delta \mathbf{w} = \alpha\left(G_t^\lambda - \hat{v}\,(S_t, \mathbf{w})\right)\mathbf{x}(S_t)$
- TD($\lambda$): Backward view with Eligibility trace $E$:

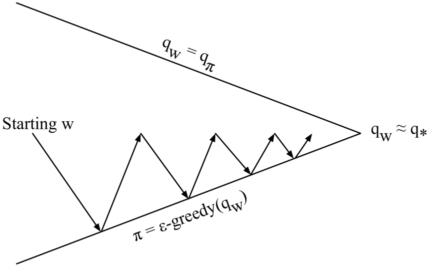
$$\begin{aligned}E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

# For control: Action-Value Function

We can do the same procedure for the action-value function.

$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$




- Policy evaluation: update $\mathbf{w}$
- Policy improvement: $\epsilon$-greedy policy improvement

# Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value function by linear combination of features:

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

- Stochastic gradient descent update

$$\begin{aligned} \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) &= \mathbf{x}(S, A) \\ \Delta \mathbf{w} &= \alpha\Big(\underbrace{q_\pi(S, A)}_{\text{target}} - \hat{q}(S, A, \mathbf{w})\Big)\mathbf{x}(S, A) \end{aligned}$$

- Replace target by: $G_t$ or TD-target ...

# On-policy control with function approximators

SARSA: using TD(0) target (semi-gradient method)

**SARSA with function approximators (episodic)**

Given: $\hat{q}(s, a, \mathbf{w}) : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

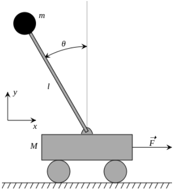
1. Initialize $\mathbf{w}$
2. Repeat (for each episode):
   - 2.1 $S \leftarrow$ from environment
   - 2.2 $A \sim \pi^{\hat{q}}(S)$ (e.g. $\epsilon$-greedy w.r.t. $\hat{q}$)
   - 2.3 Repeat (for each step of episode):
     - 2.3.1 Take action $A$, observe $R, S'$
     - 2.3.2 if $S'$ terminal
       $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})]\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$
       Go to next episode
     - 2.3.3 $A' \sim \pi^{\hat{q}}(S')$ (e.g. $\epsilon$-greedy w.r.t. $\hat{q}$)
     - 2.3.4 $\mathbf{w} \leftarrow \mathbf{w} + \alpha[\underbrace{R + \gamma\hat{q}(S', A', \mathbf{w})}_{\text{TD-target}} - \hat{q}(S, A, \mathbf{w})]\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$
     - 2.3.5 $S \leftarrow S'$; $A \leftarrow A'$

# Topical Break / Discussion

What features can you think of for:

- a cart-pendulum



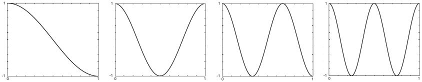

- large chain
  (random walk example with 1000 states)
- grid worlds

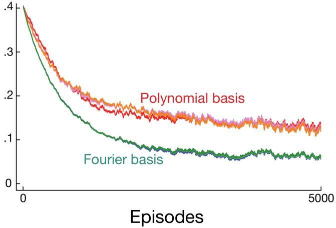
- Polynomials:
  - when states are numbers: positions/velocities of robot, #cars in parking, ...
  - for continous state-space with $s_1, s_2 \in \mathbb{R}$, consider concrete example:

    $$\mathbf{x}(s) = (s_1, s_2)^\top$$

    Why is that not so good?
  - if $s_1 = s_2 = 0$ then value is 0, no interaction between $s_1$ and $s_2$, very inflexible
  - $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2)^\top$ or $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1^2 s_2, s_1 s_2^2, s_1^2 s_2^2)^\top$
  - for degree $n$ and state-space $k$ the number of features grows exponentially in $k$
- Fourier Basis
  - Cosine basis: $\mathbf{x}_i(s) = \cos(i\pi s), s \in [0, 1]$

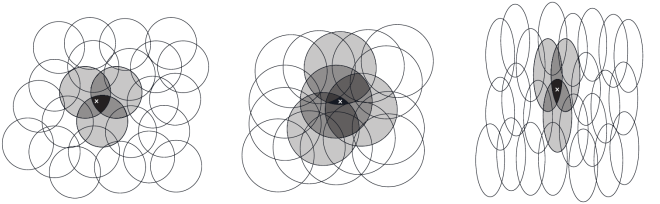


  - Value Error (RMSE) for 1000 random states example and gradient MC $\Rightarrow$



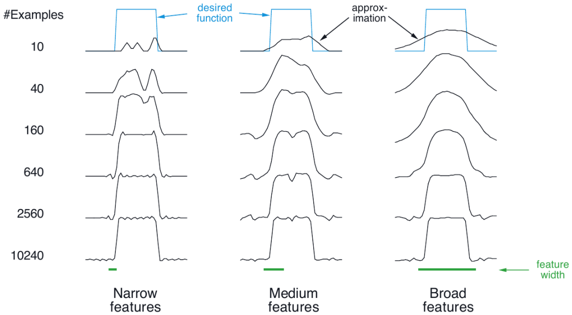

 

- Coarse Coding: inside circle: feature $= 1$



Narrow generalization | Broad generalization | Asymmetric generalization

- Initial generalization:

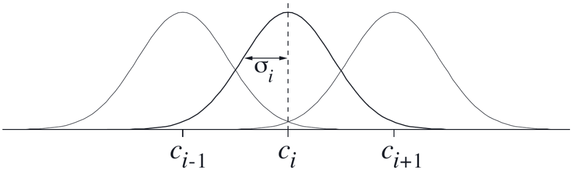


# Feature representations (Cont II)

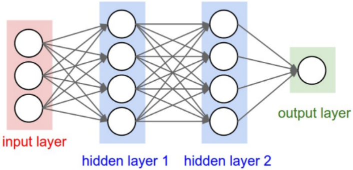
- Radial basis functions (RBF)

$$\mathbf{x}_i(s) = \exp(\|s - c_i\|^2 / 2\sigma_i^2)$$



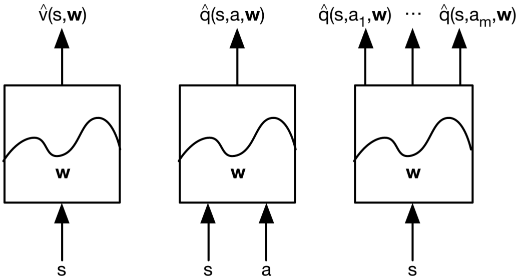

- How to choose $c_i$'sand $\sigma_i$'s?
- Scales badly with the dimensionality

## Neural Networks:



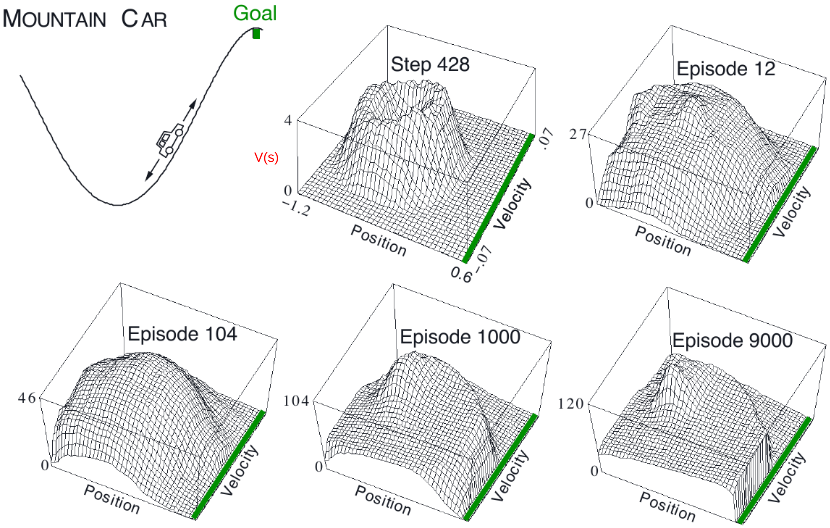

they are doing function approximation
and regression at the same time!
No need to do feature engineering

Direct approximation (implicit feature learning):



# Example: Linear SARSA with Coarse Coding
## Mountain Car Environment



# Off-policy and Convergence

- Exactly the same can be done for the off-policy case (Q-Learning)
- In the tabular case we have proven TD to converge
- With function approximation and off-policy this cannot be done

## Simple Example

- Two states: with features $w$ and $2w$:

$$w \longrightarrow 2w$$

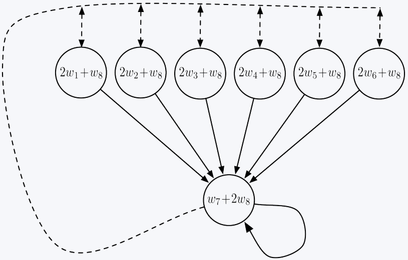
- Assume initial $w = 10$
- Transition in data: left to right
- $v(l) = 10$ to $v(r) = 20$: TD error $= 10$
- Update for $w$ with $\alpha = 0.1$: $w = 11$
- ➡ $v(r) = 22$
- Next update TD error $= 11$ (error is even larger!)

Incomplete MDP, so maybe not convincing

error is getting larger, but maybe not best of features

## Baird's Counterexample

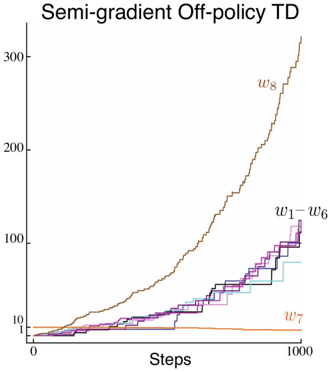



$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$




TD off-policy diverges

Features / representation needs to be adapted and should not be fixed.

# Convergence of Prediction Algorithms

| | Algorithm | Table Lookup | Linear | Non-linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✓ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✗ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |

- TD does not follow the gradient of any objective function, divergence possible
- Gradient TD follows true gradient of projected Bellman error [Maei et al, 2009]

# Convergence of Control Algorithms

| Algorithm | Table Lookup | Linear | Non-linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| SARSA | ✓ | (✓) | ✗ |
| Q-Learning | ✓ | ✗ | ✗ |
| Gradient Q-Learning | ✓ | ✓ | ✗ |

(✓) means that it fluctuate around near-optimal solution

For control with non-linear features, we cannot guarantee convergence.

# Incremental or Batch?

- So far: Incremental update, i.e. update parameters towards targets locally by SGD
- Alternative: find a parameters that fit best the experience / training data
  ➡ Batch methods

# Least Squares Prediction

- Value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- Experience $\mathcal{D}$: pairs of (state, value)

$$\mathcal{D} = \left\{ \left( s_1, v_1^\pi \right), \left( s_2, v_2^\pi \right), \ldots, \left( s_T, v_T^\pi \right) \right\}$$

- Which parameters $\mathbf{w}$ give the best fitting value function $\hat{v}(s, \mathbf{w})$?
- Least squares algorithms:

$$\begin{aligned} \mathrm{LS}(\mathbf{w}) &= \sum_{t=1}^{T} \left( v_t^\pi - \hat{v}\left( s_t, \mathbf{w} \right) \right)^2 \\ &= \mathbb{E}_{\mathcal{D}} \left[ \left( v^\pi - \hat{v}(s, \mathbf{w}) \right)^2 \right] \end{aligned}$$

- If $\hat{v}$ linear in features, then closed form solution

# SGD with Experience Replay

What about using SGD, but train on data many times?
Use experience $\mathcal{D}$:

$$\mathcal{D} = \left\{\left(s_1, v_1^\pi\right), \left(s_2, v_2^\pi\right), \ldots, \left(s_T, v_T^\pi\right)\right\}$$

Repeat:

1. Sample state value pair (or mini-batch)

$$\left(s, v^{\text{target}}\right) \sim \mathcal{D}$$

2. Apply stochastic gradient descent update:

$$\Delta \mathbf{w} = \alpha \left(v^{\text{target}} - \hat{v}(s, \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Converges to least square solution

$$\mathbf{w}^\pi = \arg\min_{\mathbf{w}} \mathrm{LS}(\mathbf{w})$$

# Problems

- We need a lot of data for high-capacity models
- In the on-policy case we have to throw away all the collected experience :-(
- Off-policy should be more efficient
  - Wait: don't we have convergence problems?
  - Neural networks are very flexible and can in practice often break the parameter-state entanglement leading to divergence
  - If behavior policy is closed to target policy (e.g. $\epsilon$-greedy version) less of a problem
  - We can freeze the value-target, more below

Deep Q-Networks

- Q-Learning with Deep network as function approximator
- + Experience Replay
- + Target Network

### Experience Replay

Data: $\mathcal{D} = \{(s_t, a_t, r_{t+1}, s_{t+1})\}$

Typically a fixed maximal size $|\mathcal{D}|$ (old data gets deleted)

### Target Network

Two sets of parameters: $\mathbf{w}$ and $\mathbf{w}^{\text{target}}$
Idea: keep $\mathbf{w}^{\text{target}}$ fixed for some time (old value of $\mathbf{w}$)
TD: error is computed using $\mathbf{w}^{\text{target}}$

- avoids instabilities
- for a fixed $\mathbf{w}^{\text{target}}$ it is a gradient method

DQN [Mnih et al 2015], Earlier work: Neural fitted Q [Riedmiller 2005]

## DQN

Given: Network $\hat{q}(s, a; \mathbf{w})$, replay buffer $\mathcal{D}$

1. Initialize $\mathbf{w}$, $\mathbf{w}^{\text{target}} \leftarrow \mathbf{w}$
2. $s \leftarrow$ from environment
3. $a \sim \pi^{\hat{q}}(s)$ (e.g. $\epsilon$-greedy w.r.t. $\hat{q}$)
4. Repeat (for each step of episode):
   - 4.1 Take action $a$, observe $r, s'$
   - 4.2 $n \leftarrow \begin{cases} 0 & s' \text{ is terminal} \\ 1 & \text{otherwise} \end{cases}$ (*not done* flag)
   - 4.3 Store $(s, a, r, s', n)$ in $\mathcal{D}$
   - 4.4 Sample random minibatch $D \subset \mathcal{D}$
   - 4.5 Optimize with respect to:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{s,a,r,s',n\sim\mathcal{D}}\left[\left(r + n\gamma \max_{a'} \hat{q}\left(s', a'; \mathbf{w}^{\text{target}}\right) - \hat{q}\left(s, a; \mathbf{w}\right)\right)^2\right]$$

     with favorit optimizer (ADAM, SGD)

5. after many update steps: $\mathbf{w}^{\text{target}} \leftarrow \mathbf{w}$

DQN [Mnih et al 2015], Earlier work: Neural fitted Q [Riedmiller 2005]

# Every iteration or grouped training?

- Due to random sampling, the currently collected data-point has very small influence
- Often easier to collect a bunch of data (e.g. one episode)
- make a number of training steps and continue

# DQN in Atari

What made Deep RL famous in 2015. . .

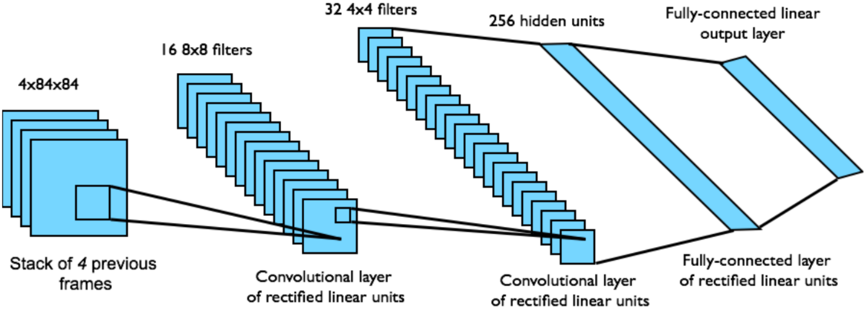
www.youtube.com/watch?v=rQIShnTz1kU

# DQN in Atari

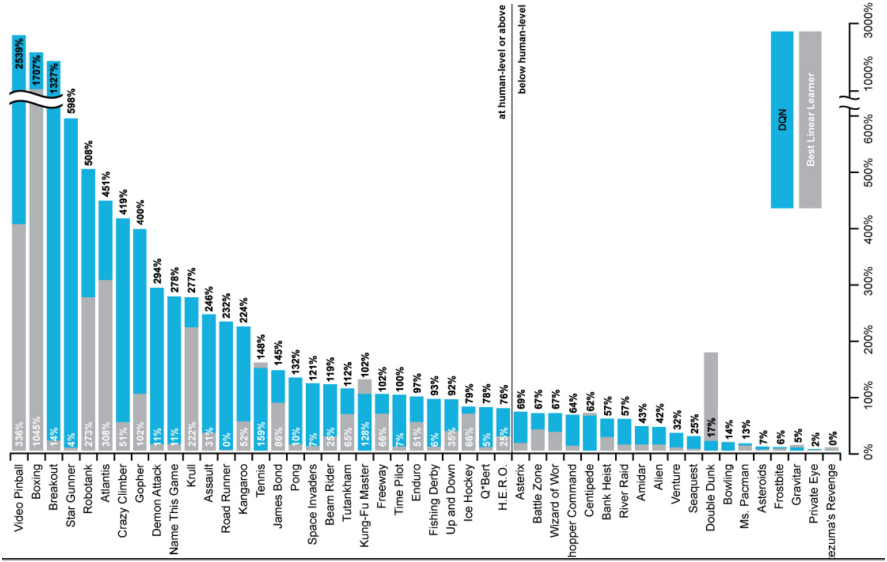
What made Deep RL famous in 2015...

- Observation space: raw pixels $(84 \times 84)$, 4 frames
- End-to-end learning of Q-values from pixels $s$
- Action space: 18 discrete joystick positions and buttons
- Network $q(s, \mathbf{w}) \mapsto \mathbb{R}^{18}$: direct prediction of Q-values for all actions
- Reward: change in score for that step



# DQN Results in Atari




Best linear learner: using the best results with handcrafted features

# Ablation

## How important is Replay and Target Network?

Scores on some Atari games

| | Replay Target-Q | Replay Q-learning | No replay Target-Q | No replay Q-learning |
|---|---|---|---|---|
| Breakout | 316.81 | 240.73 | 10.16 | 3.17 |
| Enduro | 1006.30 | 831.25 | 141.89 | 29.10 |
| Space Invaders | 1088.94 | 826.33 | 373.22 | 301.99 |
| Seaquest | 2894.40 | 822.55 | 1003.00 | 275.81 |
| River Raid | 7446.62 | 4102.81 | 2867.66 | 1453.02 |

# Linear Least Squares Prediction and Control

- Experience replay finds least squares solution
  but it may take many iterations
- Using linear value function approximation $\hat{v}(s, \mathbf{w}) = x(s)^\top \mathbf{w}$
  solve the least squares solution in closed form

# Linear Least Squares Prediction

Linear v.fn.: $\hat{v}(s, \mathbf{w}) = x(s)^\top \mathbf{w}$
Targets: $v_t^\pi$
Loss: $L(\mathbf{w}) = \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2$

Refresher: Linear Regression

- At minimum of $\mathrm{LS}(\mathbf{w})$: Derivative is zero.

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$$

$$\sum_{t=1}^{T} \mathbf{x}(s_t) \left( v_t^\pi - \mathbf{x}(s_t)^\top \mathbf{w} \right) = 0$$

$$\sum_{t=1}^{T} \mathbf{x}(s_t)\, v_t^\pi = \sum_{t=1}^{T} \mathbf{x}(s_t)\, \mathbf{x}(s_t)^\top \mathbf{w}$$

$$\mathbf{w} = \left( \sum_{t=1}^{T} \mathbf{x}(s_t)\, \mathbf{x}(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t)\, v_t^\pi$$

- For $N$ features, direct solution time is $O\left(N^3\right)$
- Incremental solution time is $O\left(N^2\right)$ using Shermann-Morrison

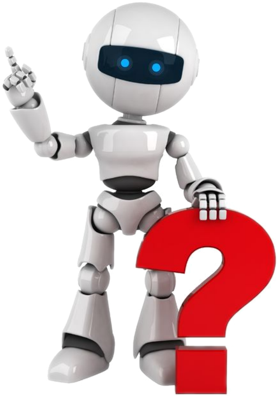
# Convergence of Least Squares Prediction Algorithms

- As usual: plugging in different targets: LS$\{$MC$|$TD$|$TD$(\lambda)\}$

| | Algorithm | Table Lookup | Linear | Non-linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | LSMC | ✓ | ✓ | – |
| | TD | ✓ | ✓ | ✗ |
| | LSTD | ✓ | ✓ | – |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | LSMC | ✓ | ✓ | – |
| | TD | ✓ | ✗ | ✗ |
| | LSTD | ✓ | ✓ | – |

- On-Policy Least Squares needs lots of samples from current policy
- Putting it inside control: Policy iteration: LSPI
- LSPI also converges in off-policy setting
  (naturally limited to the linear case)

# Questions?



[Image: globalrobots.com]