6- Approximate TD Methods

Melih Kandemir

University of Southern Denmark
Department of Mathematics and Computer Science (IMADA)
kandemir@imada.sdu.dk

Fall 2022

RL in large and/or continuous state spaces

What if

- the state space is too large to allocate memory for every state?
- or, the state description is very high-dimensional (e.g. a Go table, or the scene image)?

A solution approach:

- Project states to a feature space $\phi(i)$. Do feature extraction, use kernels, learn an **embedding** separately or **end-to-end**, i.e. as part of the RL algorithm.
- Represent cost-to-go as a parametric function, for instance a neural network, that is

$$J_\theta^\pi(i) = w_2^T \sigma(W_1^T \phi(i))$$

where $\theta = \{W_1, w_2\}$ and σ is an activation function (e.g. ReLU).

Value-based RL as a supervised learning problem

Given a sample (i_k, u_k, i_{k+1}), solve

$$\arg\min_{\theta} \left(\underbrace{g(i_k, u_k, i_{k+1}) + \alpha J_\theta^\mu(i_{k+1})}_{\text{target}} - \underbrace{J_\theta^\mu(i_k)}_{\text{prediction}} \right)^2$$

Since θ describe the cost-to-go of all possible states, unlike the tabular approach, updating parameters for a single state affects the values of many other states!
We **bootstrap** if J_θ^μ is used both in prediction and target calculation.

MC versus TD on a single episode

Reorganize the sampled episode $i_0, u_0, i_1, u_1, \ldots, i_{N-1}, u_{N-1}, i_N = 0$ as $\{(i_0, u_0, i_1), (i_1, u_1, i_2), \ldots, (i_{N-1}, u_{N-1}, 0)\}$, generate a labeled data set and do gradient-descent

- **MC:** $\mathcal{D} = \{(i_0, \sum_{k=0}^{N-1} g(i_k, u_k, i_{k+1}))\}$

$$\theta := \theta - \gamma \left[\sum_{k=0}^{N-1} g(i_k, u_k, i_{k+1}) - J_\theta^\mu(i_0) \right] \nabla_\theta J_\theta^\mu(i_0)$$

- **TD(0):**

$$\mathcal{D} = \left\{ \begin{array}{c} (i_0, g(i_0, u_0, i_1) + \alpha J_\theta^\mu(i_1)) \\ (i_1, g(i_1, u_1, i_2) + \alpha J_\theta^\mu(i_2)) \\ \vdots \\ (i_{N-1}, g(i_{N-1}, u_{N-1}, 0)) \end{array} \right\}$$

$$\theta := \theta - \gamma \Big[g(i_k, u_k, i_{k+1}) + \alpha J_\theta^\mu(i_{k+1}) - J_\theta^\mu(i_k) \Big] \nabla_\theta J_\theta^\mu(i_k),$$

$$k = 0, \ldots, N-1$$

MC versus TD

- TD can learn *before* the final outcome is observed
  - TD learns online from every state transition
  - MC has to wait the episode end to calculate the return
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences (i.e. works in continuing environments)
  - MC can only learn from complete sequences (i.e. works only in episodic environments)
- TD exploits Markov property, MC does not
  - TD works better if the environment is Markov
  - MC can better handle non-stationarity

Unified view of RL algorithms

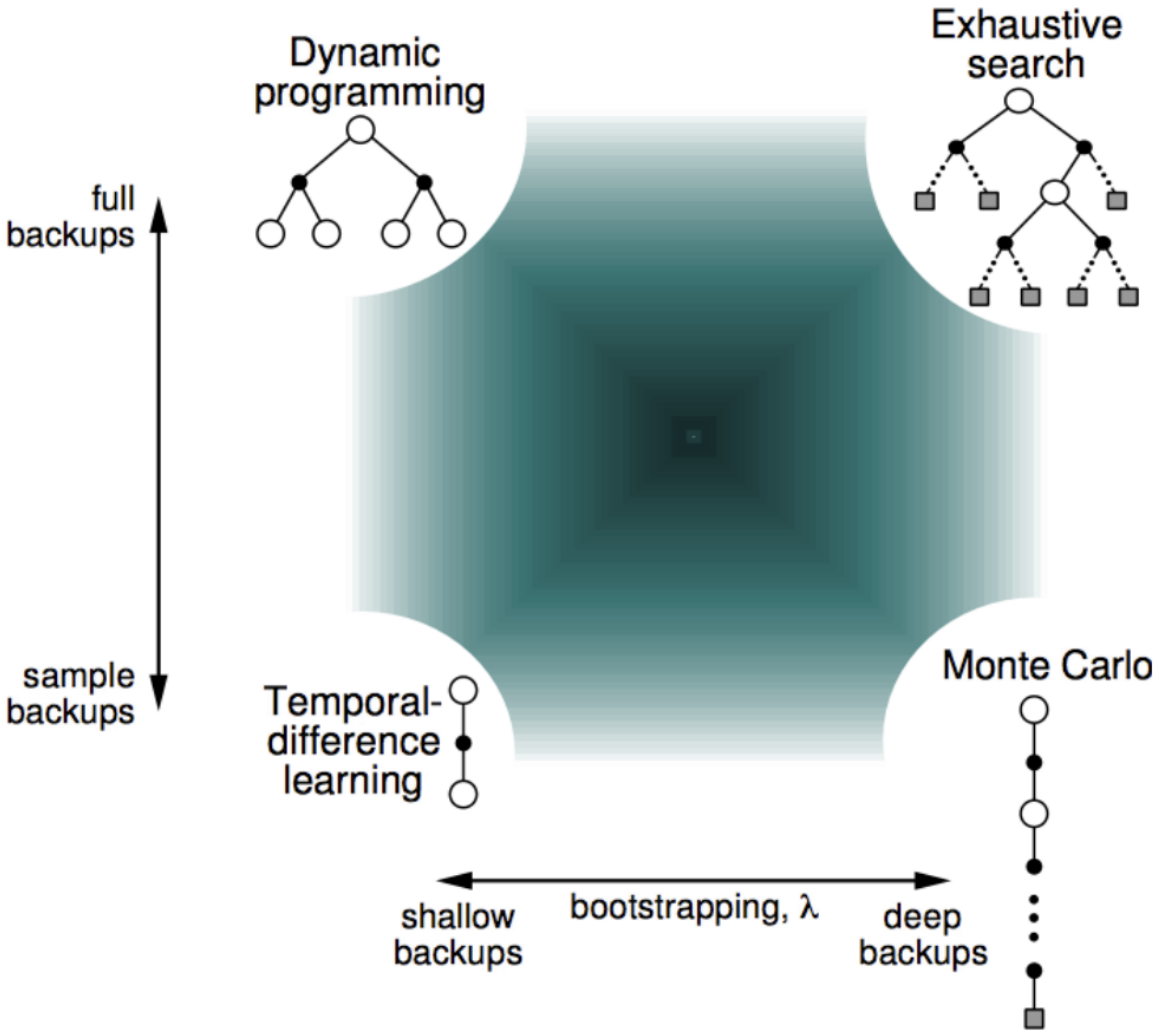


Figure. D. Silver, lecture slides

Linear value approximation

Assume $J_\theta^\mu(i) = \phi(i)^T\theta$ where $\theta \in \mathbb{R}^d$, then

$$\arg\min_\theta \sum_{k=0}^{\infty} \Big(g(i_k, u_k, i_{k+1}) + \alpha\phi(i_{k+1})^T\theta - \phi(i_k)^T\theta \Big)^2$$

Denote $g(i_k, u_k, i_{k+1}) := g_k$, treat $\phi(i_{k+1})^T\theta$ as a constant target, set the gradient of the loss to zero, reorganize, and solve

$$\sum_{k=0}^{\infty} \Big[g_k\phi(i_k) - \phi(i_k)(\phi(i_k) - \alpha\phi(i_{k+1}))^T\theta \Big] = 0$$

$$\therefore \quad \theta = \left[\sum_{k=0}^{\infty} \phi(i_k)(\phi(i_k) - \alpha\phi(i_{k+1}))^T \right]^{-1} \sum_{k=0}^{\infty} g_k\phi(i_k)$$

Online updates

Define the solution for time step k as

$$A_k = \sum_{m=0}^{k} \phi(i_m)(\phi(i_m) - \alpha\phi(i_{m+1}))^T, \qquad b_k = \sum_{m=0}^{k} g_m \phi(i_m)$$

Use Sherman-Morrison formula to incrementally update A_{k+1}^{-1}:

$$\begin{aligned} A_{k+1}^{-1} &= \left[A_k + \phi(i_k)(\phi(i_k) - \alpha\phi(i_{k+1}))^T\right]^{-1} \\ &= A_k^{-1} - \frac{A_k^{-1}\phi(i_k)(\phi(i_k) - \alpha\phi(i_{k+1}))^T A_k^{-1}}{1 + (\phi(i_k) - \alpha\phi(i_{k+1}))^T A_k^{-1}\phi(i_k)}. \end{aligned}$$

Remark that the ordering of the matrix product in the denominator is important as A_k^{-1} may not be symmetric.

Least Squares Temporal Differences (LSTD) algorithm

$A^{-1} := \epsilon^{-1} I, b := 0$
repeat
  choose random i
  repeat
    act $u \sim \mu(i)$, observe i', calculate $g(i, u, i')$
    $v := (\phi(i) - \alpha\phi(i'))^T A^{-1}$
    $A^{-1} := A^{-1} - A^{-1}\phi(i)v^T/(1 + v\phi(i))$
    $b := b + g(i, u, i')\phi(i)$
    $i := i'$
  until $i := 0$
until convergence
return $J^\mu := A^{-1}b$

Stochastic Gradient Descent (SGD)

One update requires a full pass on the entire data set

$$\theta := \theta - \gamma \underbrace{\left[\sum_{k=0}^{N-1} g(i_k, u_k, i_{k+1}) - J_\theta^\mu(i_0)\right] \nabla_\theta J_\theta^\mu(i_0)}_{\text{true gradient}}$$

This

- is expensive if the data set is large.
- delays model fit.

Choose a random k and collect **minibatch** of size $\widetilde{N}$

$$\widetilde{\mathcal{D}} = \{(i_{k+m}, u_{k+m}, i_{k+m+1}) | m = 0, \ldots, \widetilde{N} - 1\}$$

and update

$$\theta := \theta - \gamma \underbrace{\left[\frac{N}{\widetilde{N}} \sum_{k=0}^{\widetilde{N}-1} g(i_{k+m}, u_{k+m}, i_{k+m+1}) - J_\theta^\mu(i_0)\right] \nabla_\theta J_\theta^\mu(i_0)}_{\text{stochastic gradient}}$$

Robbins-Monro Theorem (1951)

The stochastic gradient provides an **unbiased** estimator of the true gradient if updates are performed following a learning rate series satisfying the two properties below

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \tag{1}$$

$$\sum_{t=1}^{\infty} \epsilon_t^2 < \infty. \tag{2}$$

(1) reach at points arbitrarily far away
(2) stop learning at some point

SGD with and without replacement

- **without replacement:**
  - choose a sample only once until all samples are chosen, i.e. until an **epoch** is complete.
  - more common in standard ML due mainly to practical reasons, e.g. to decide whether the epoch is over.
- **with replacement:**
  - keep random sampling without caring about coverage.
  - allow multiple selection of a sample within an epoch.
  - more common in RL.
  - called **experience replay**.

Semi-gradient $TD(0)$

repeat
  choose (random) i
  repeat
    act $u \sim \mu(i)$, observe i', calculate $g(i, u, i')$
    $\theta := \theta - \gamma \Big[g(i, u, i') + \alpha J_\theta^\mu(i') - J_\theta^\mu(i) \Big] \nabla_\theta J_\theta^\mu(i)$
  until $i := 0$
until convergence
return $J_\theta^\mu(\cdot)$

Gradient-descept step does bootstrapping. This breaks the Robbins-Monro assumptions, i.e. introduces estimator bias and doesn't ensure convergence.

Semi-gradient off-policy control: Deep Q-learning

Define a parametric Q-factor $Q_\theta(u;i) = W_2^T \sigma(W_1^T \phi(i)))$ where W_2 is a matrix with one output dimension per control.

$\mathcal{D} := \emptyset$ ▷ Init replay buffer
repeat
  choose (random) i ▷ Episode start
  repeat
    act $u \sim \text{softmax}(-Q_\theta(\cdot;i))$, observe i', $g_i := g(i,u,i')$
    $\mathcal{D} := \mathcal{D} \cup (i,u,g_i,i')$, $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$ ▷ Sample minibatch
    $\theta := \theta - \frac{\gamma}{|\widetilde{\mathcal{D}}|} \sum_{i \in \widetilde{\mathcal{D}}} \left[g_i + \alpha \min_v Q_\theta(v;i') - Q_\theta(u;i) \right] \nabla_\theta Q_\theta(u;i)$
    $i := i'$
  until $i := 0$
until convergence
return $Q_\theta(\cdot;\cdot)$

Replay buffer is a queue: $|\mathcal{D}| > \tau \Rightarrow \mathcal{D} \backslash (i_0, u, g_0, i_1)$ for memory size τ.

Semi-gradient on-policy control: Deep Sarsa

On-policy: The policy used for taking actions and the policy used for the Bellman backup are the same.

$\mathcal{D} := \emptyset$ ▷ Init replay buffer

repeat

  choose (random) i ▷ Episode start

  repeat

    act $u \sim \text{softmax}(-Q_\theta(\cdot; i))$, observe i', $g_i := g(i, u, i')$

    $u' \sim \text{softmax}(-Q_\theta(\cdot; i'))$

    $\mathcal{D} := \mathcal{D} \cup (i, u, g_i, i', u')$, $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$ ▷ Sample minibatch

    $\theta := \theta - \frac{\gamma}{|\widetilde{\mathcal{D}}|} \sum_{i \in \widetilde{\mathcal{D}}} \Big[g_i + \alpha Q_\theta(u'; i') - Q_\theta(u; i) \Big] \nabla_\theta Q_\theta(u; i)$

    $i := i'$

  until $i := 0$

until convergence

return $Q_\theta(\cdot; \cdot)$

High variance. Requires $\gamma << 1$.

Deep Expected Sarsa

$\mathcal{D} := \emptyset$ ▷ Init replay buffer

repeat

  choose (random) i ▷ Episode start

  repeat

    $\mu(\cdot|i) := \text{softmax}(-Q_\theta(\cdot; i))$

    act $u \sim \mu(\cdot|i)$, observe i', calculate $g_i := g(i, u, i')$

    $\mathcal{D} := \mathcal{D} \cup (i, u, g_i, i')$, $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$ ▷ Sample minibatch

    $\theta := \theta - \frac{\gamma}{|\widetilde{\mathcal{D}}|} \sum_{i \in \widetilde{\mathcal{D}}} \Big[g_i + \alpha \sum_{v \in U(i)} \mu(v|i') Q_\theta(v; i') - Q_\theta(u; i) \Big] \nabla_\theta Q_\theta(u; i)$

    $i := i'$

  until $i := 0$

until convergence

return $Q_\theta(\cdot; \cdot)$

Less variance, but more computation cost. No longer requires $\alpha << 1$.

Convexity

Consider a parametric line $a\lambda + b(1-\lambda)$ that passes between points a and b and an arbitrary function $f(x)$. If any line passing between $f(a)$ and $f(b)$ is always above $f(x)$, then $f(x)$ is called a **convex function**. More formally, if for any a and b the below inequality satisfies

$$f(a)\lambda + f(b)(1-\lambda) \geq f(a\lambda + b(1-\lambda)),$$

then $f(x)$ is said to be convex.

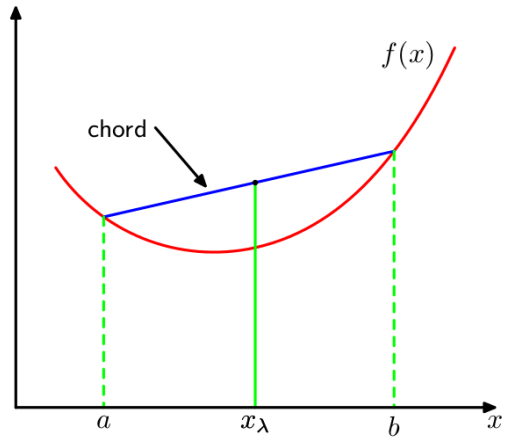


Figure: C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

Jensen's inequality

We can prove by induction that convexity holds also for more than two points:

$$f\left(\sum_{i=1}^{M}\lambda_i x_i\right) \leq \sum_{i=1}^{M}\lambda_i f(x_i),$$

such that $\{x_1, \cdots, x_M\}$ is a set of points on the function domain and $\sum_{i=1}^{M}\lambda_i = 1$ with $\lambda_i \geq 0$. We can think of $\{\lambda_i, \cdots, \lambda_M\}$ as parameters of a categorical distribution with M states. Hence we can have

$$f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)].$$

The difference $\mathbb{E}[f(x)] - f(\mathbb{E}[x])$ is called the **Jensen gap**. This outcome generalizes to continuous variables straightforwardly (use Riemann integration):

$$\int f(x)p(x)dx \geq f\Big(\int xp(x)dx\Big).$$

Minimization bias in Q-learning

Let us re-develop the Bellman equation paying attention to the order of the expectations

$$\begin{aligned} Q_\theta(u;i) &= \mathbb{E}_{p^u_{ii'}}[g_i] + \gamma \mathbb{E}_{p^u_{ii'}}[\mathbb{E}_{\mu(v|i')}[Q_\theta(v;i')]] \\ &= \mathbb{E}_{p^u_{ii'}}[g_i] + \gamma \mathbb{E}_{p^u_{ii'}}[\min_v Q_\theta(v;i')] \end{aligned}$$

since μ is a deterministic greedy policy. Q-learning calculates the TD error evaluating $\min_v Q_\theta(v;i')$ with i' sampled before the $\min$ operator. Assume this process is repeated K times $i'_{(1)}, \ldots, i'_{(K)}$, then asymptotically we have

$$\begin{aligned} \lim_{K\to+\infty} \min_v \frac{1}{K} Q_\theta(i'_{(k)}, v) &= \mathbb{E}_{p^u_{ii'}}[\min_v Q_\theta(i', v)] \\ &\neq \min_v \mathbb{E}_{p^u_{ii'}}[Q_\theta(i', v)]. \end{aligned}$$

Minimization bias and double learning

- Many RL algorithms use $\mathbb{E}[\min(a, b)]$ to approximate $\min(\mathbb{E}[a], \mathbb{E}[b])$, such as in the target calculation of Q-learning, in $\epsilon-$greedy calculation of Sarsa, etc.
- However, $\min$ operator is concave, hence due to Jensen's inequality, we have

$$\min(\mathbb{E}[a], \mathbb{E}[b]) \geq \mathbb{E}[\min(a, b)],$$

  which causes underestimation of the cost-to-go.
- The systematic error $\mathbb{E}[\min(a, b)] - \min(\mathbb{E}[a], \mathbb{E}[b])$ resulting from this approximation is called the **minimization bias**.

Double learning

- The minimization bias problem emerges from using the same samples both to determine the minimizing action and to estimate its value.
- A solution is to use one Q-factor to determine the maximizing action and another one to estimate its value

$$Q^z_{\theta_z}\left(\arg\min_v \; Q^{z'}_{\theta'_z}(v;i');i'\right).$$

- The outcome is an unbiased estimate of the value of the maximizing action.
- The trick can be used anywhere: Q-learning, Sarsa, Expected Sarsa, etc.

The double deep Q-learning (DDQL) algorithm

$\mathcal{D} := \emptyset$ ▷ Init replay buffer

repeat

choose (random) i ▷ Episode start

repeat

$\mu(\cdot|i) := \text{softmax}(-(Q^0_{\theta_0}(\cdot; i) + Q^1_{\theta_1}(\cdot; i))/2)$

act $u \sim \mu(u|i)$, observe i', calculate $g_i := g(i, u, i')$

$\mathcal{D} := \mathcal{D} \cup (i, u, g_i, i')$, $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$ ▷ Sample minibatch

$z \sim \text{Bernoulli}(0.5)$, $z' := 1 - z$

$$\theta_z := \theta_z - \frac{\gamma}{|\widetilde{\mathcal{D}}|} \sum_{i \in \widetilde{\mathcal{D}}} \Big[g_i + \alpha Q^z_{\theta_z}(\arg\min_v Q^{z'}_{\theta'_z}(v; s'); i') - Q^z_\theta(u; i) \Big] \nabla_\theta Q^z_\theta(u; i)$$

$i := i'$

until $i := 0$

until convergence

return $Q^\mu_\theta(\cdot; \cdot)$

Example: Mountain Car

- Drive a car out of a U-shaped valley.
- Gravity is stronger than the car's engine.
- **Reward:** -1 per time step, $+100$ for reaching the goal.
- **Actions:**
  - $+1$ full throttle forward,
  - -1 full throttle backwards,
  - 0 zero throttle.
- The system dynamics are as below

$$i_{k+1} := i_k + \frac{di_{k+1}}{dt}$$
$$\frac{di_{k+1}}{dt} := \frac{di_k}{dt} + 0.001u_k - 0.025\cos(3i_k),$$

where i_k denotes the position and $\frac{di_k}{dt}$ the velocity of the car.

Example: Mountain Car

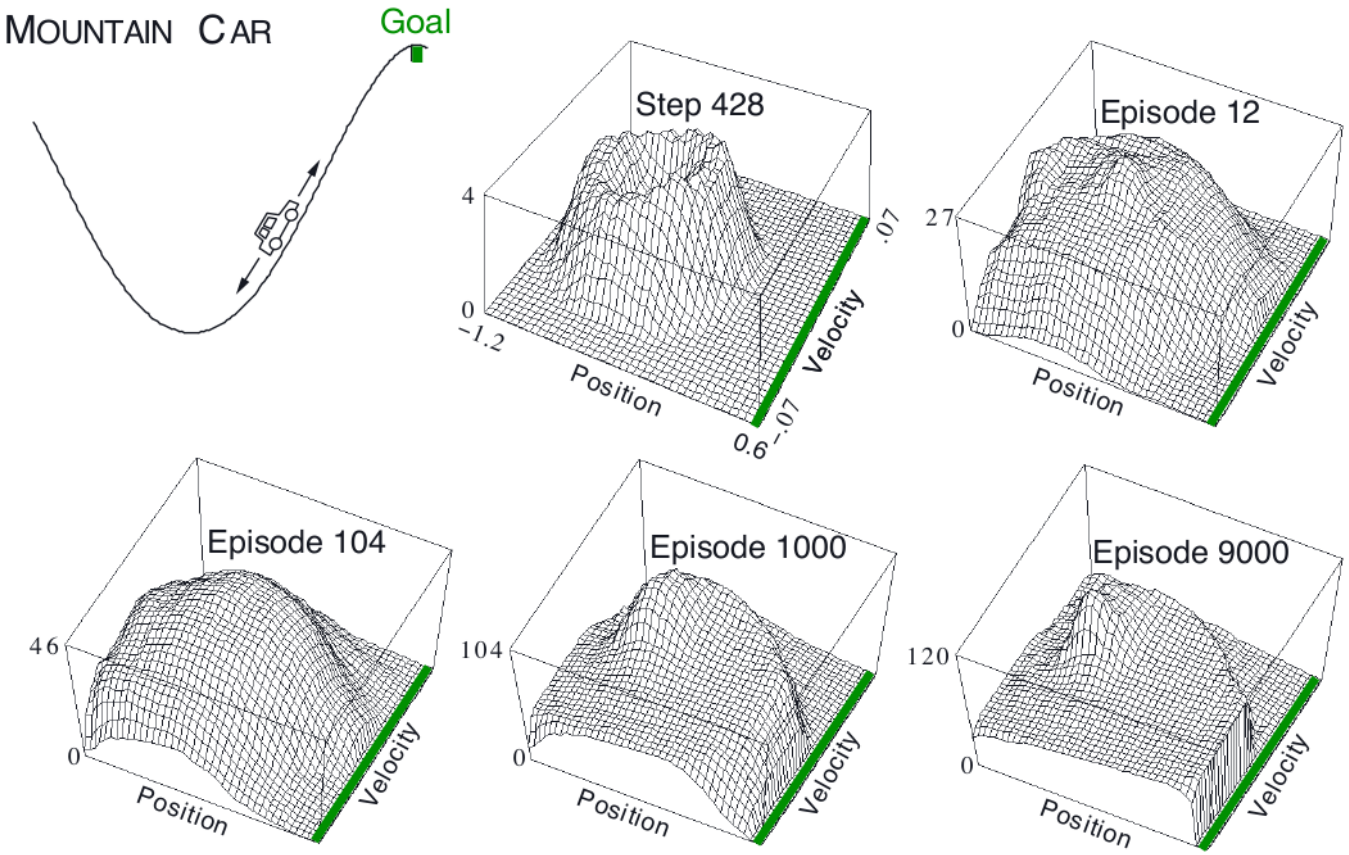


Figure: R. Sutton, A. Barto, MIT Press, 2017

Example: Mountain Car

- Mountain car is a standard application for delayed reward: *Driving towards the exit point is not the right way.*
- Step 428 has a symmetric shape, because all initially visited states are valued worse than the default value unexplored states.
- Consequently, the agent decides to explore for long episodes even though $\epsilon = 0$.

$N-$step semi-gradient Sarsa on Mountain Car

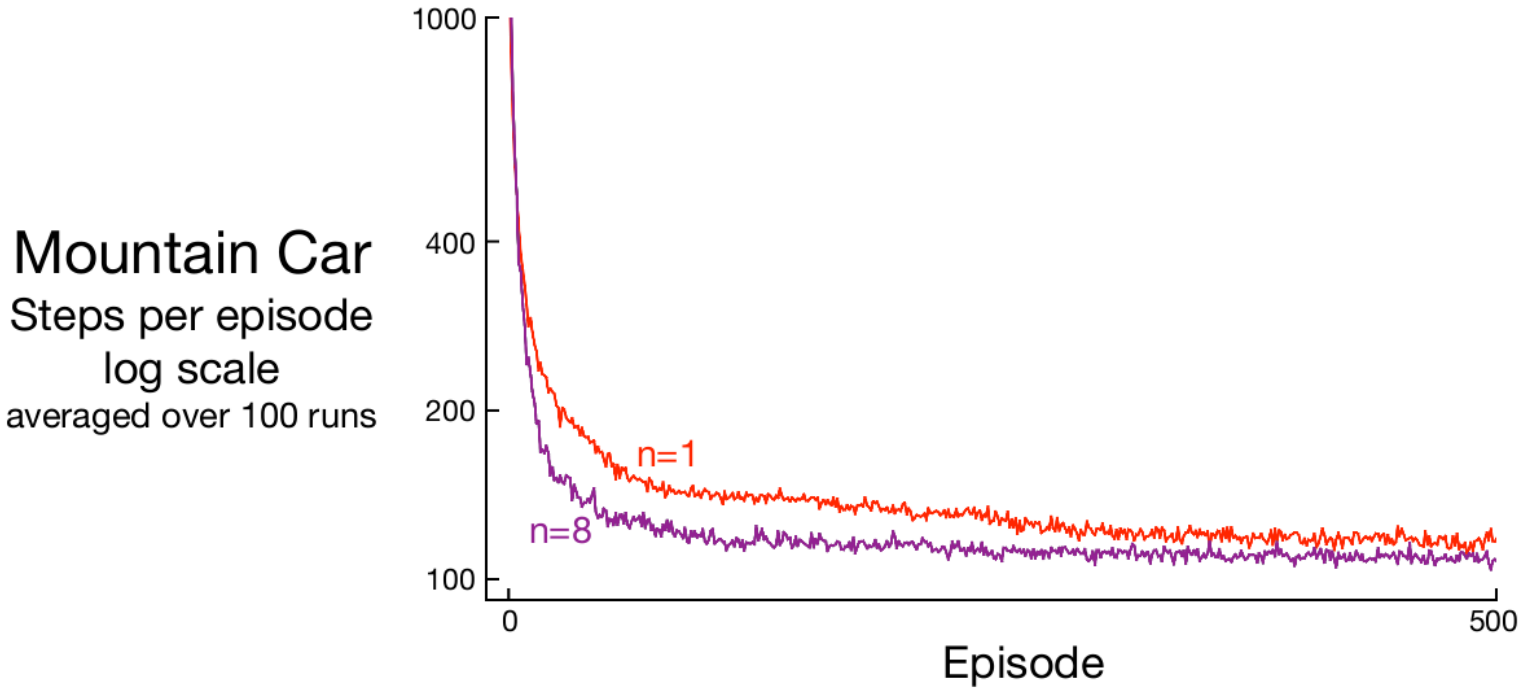


Figure: R. Sutton, A. Barto, MIT Press, 2017

N-step Deep Q-Learning (DQL)

$\mathcal{D} := \emptyset$ ▷ Init replay buffer

repeat

  choose (random) i ▷ Episode start

  repeat

    $g_i^N := 0$

    for do $n = 0, \ldots, N-1$

      act $v \sim \text{softmax}(-Q_\theta(\cdot; i))$

      if $n = 0$ **then** $i_0 := i, u := v$ ▷ Save first state/action

      observe i', calculate $g_i^N := g_i^N + g(i, v, i')$

      $i := i'$

    end for

    $\mathcal{D} := \mathcal{D} \cup (i_0, u, g_i^N, i')$, $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$ ▷ Sample minibatch

    $\theta := \theta - \frac{\gamma}{|\widetilde{\mathcal{D}}|} \sum_{i \in \widetilde{\mathcal{D}}} \left[g_i^N + \alpha^N \min_v Q_\theta(v; i') - Q_\theta(u; i) \right] \nabla Q_\theta(u; i)$

  until $i := 0$

until convergence

return $Q_\theta(\cdot; \cdot)$

Attention Networks

- **Episodic memory:** Timestamped storage of experience in the hippocampus. Prominence is proportional to arousal.
- Idea is to build an artificial hippocampus to protect key events from the **catastrophic interference** of gradient-descent.
- Maintain a memory $M = \{(h_1, V_1), \ldots, (h_R, V_R)\}$, called a **Differential Neural Dictionary (DND)**, consisting of key-value pairs (h_j, V_j). **Key** h_j is the address and **value** V_j is the content of a memory element j.
- For a state i, value retrieval from memory takes place as follows

procedure `attend`(i, M)

$h := e_\psi(i)$ ▷ Generate key

$w_j := k(h, h_j) \Big/ \sum_{j'=1}^{R} k(h, h_{j'}), \forall j = 1, \ldots, R$ ▷ Compute attention

return $V(i) := \sum_{j=1}^{R} w_j V(j)$

for a given similarity score, e.g. $k(h, h') := \langle h, h' \rangle / (||h|| \cdot ||h'||)$.

Semi-tabular TD: Neural Episodic Control

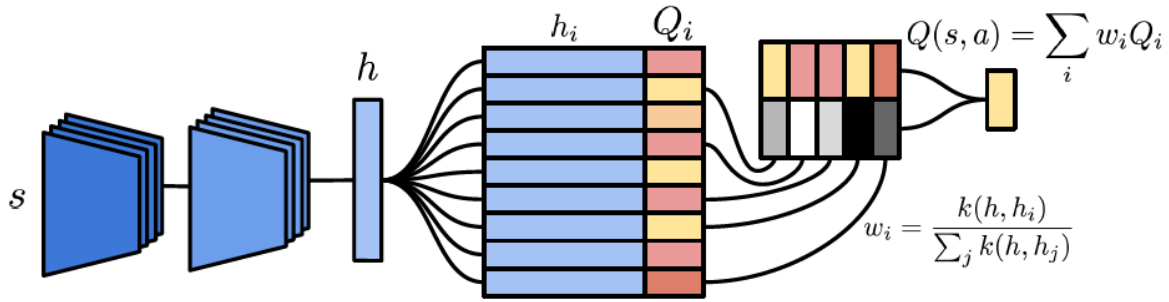


Figure: `https://arxiv.org/abs/1703.01988`

- Values retrieved from the memory can be updated much faster than DQL.
- Fast approximate nearest-neighbor search on the large memory via KD-tree algo
- Aims to achieve essential properties of hippocampus: Long-term memory (DND), sequentiality (N-step), context look-up (attention)
- N-step Q-learning is better for fast **reward propagation**.
- Non-parametric methods are essential for data efficiency.
- Choose $V_u(i) = Q(u; e^u_\psi(i'))$.

The NEC algorithm

$\mathcal{D} := \emptyset, M_u := \emptyset, \forall u \in U$ ▷ Init replay buffer and memory

repeat

  $g_i^N := 0$

  for do $n = 0, \ldots, N-1$

    act $v \sim \text{softmax}(-Q(\cdot; i))$

    if $n = 0$ **then** $i_0 = i, u := v$

    observe i', calculate $g_i^N := g_i^N + g(i, v, i')$

    $i := i'$

  end for

  $G_u^N := g_i^N + \alpha^N \min_v \texttt{attend}(i', M_v)$ ▷ Bellman target

  if $\max_{h \in M_u} k(e_\psi(i), h) < \tau$ **then** $M_u := M_u \cup (e_\psi(i), G_u^N)$

  $\mathcal{D} := \mathcal{D} \cup (i_0, u, G_u^N, i'), \qquad \widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$

  $Q(u; j) := Q(u; j) + \gamma w_j (G_u^N - Q(u; j)), \forall j$ ▷ Tabular update

  $\psi := \psi - \frac{\kappa}{|\widetilde{\mathcal{D}}|} \sum_{i \in \widetilde{\mathcal{D}}} \left[G_u^N - Q(u; e_\psi^u(i)) \right] \nabla_\psi Q(u; e_\psi^u(i))$

until convergence

NEC Results

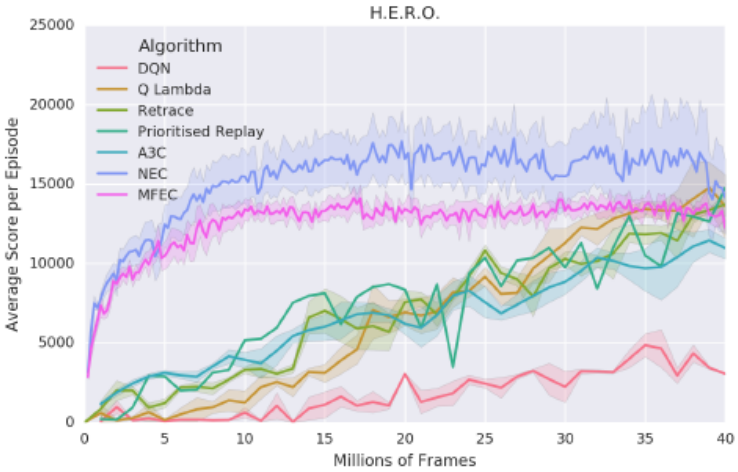


Figure 5. Learning curve on H.E.R.O.

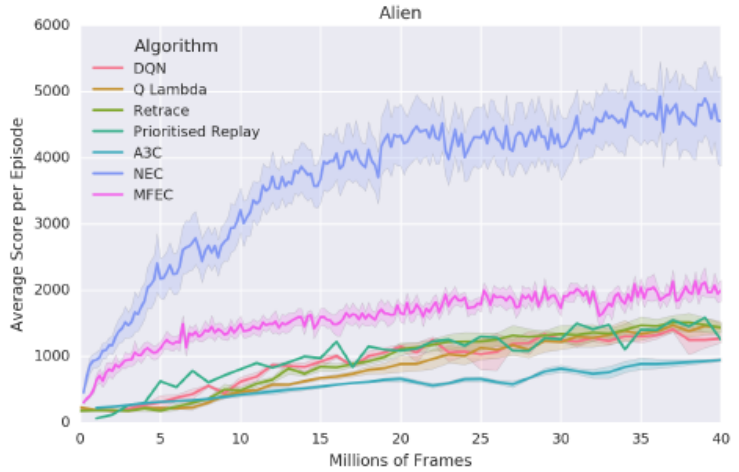
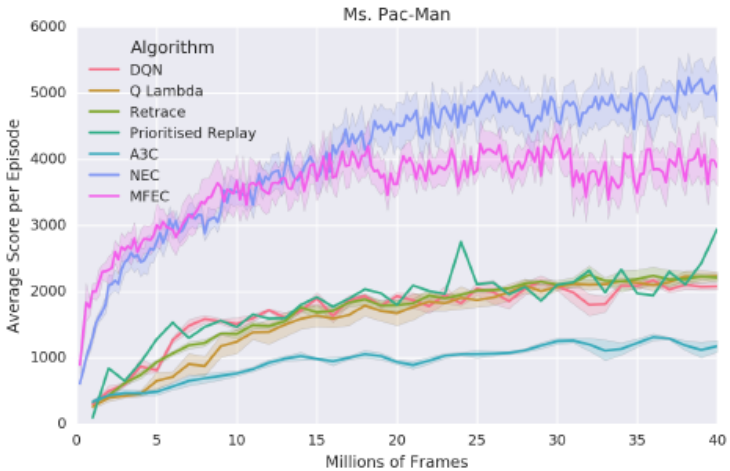


Figure 7. Learning curve on Alien.



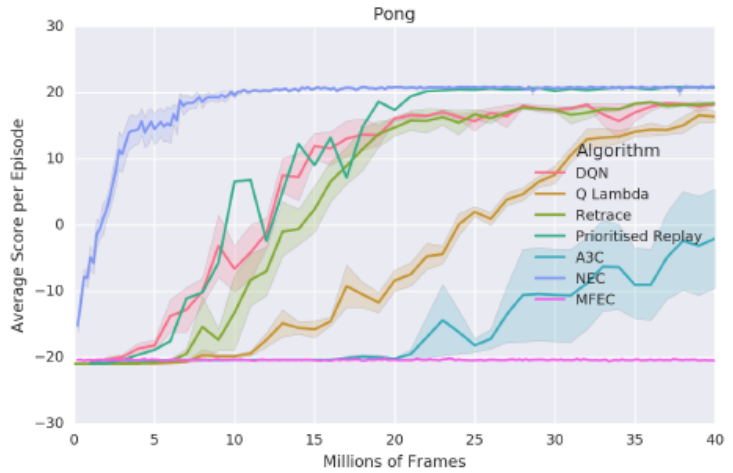


Figure: `https://arxiv.org/abs/1703.01988`