1 Introduction

1.1 Overview

1.1.1 On General Machine Learning

ML has seen a huge (exponential) increase in academic interest since 2000. It is categorized as follows:

1. Supervised and unsupervised Learning (SL/ UL): Only involves generalization $\implies$ Differs in whether there is labelling
2. Imitation Learning (IL): behaviour cloning, assuming input demonstrations of GOOD policies
3. **Reinforcement Learning (RL): model-based learning (i.e. given reward info, states reached and actions taken)**

1.1.2 On Reinforcement Learning

RL refers to **Sequential decision making** to make good decisions under uncertainty.

Steps of RL

RL involves the following four steps:

1. **Optimization**: Find optimal decisions yielding best/ very good outcomes (e.g. minimum distance inter-city route).
2. **Delayed consequences**: Plan and reason long term ramifications (e.g. saving for retirement), through temporal credit assignment (i.e. identifying which past actions led to current rewards?).
3. **Exploration**: Learn/ explore by making decisions (that impacts/ reinforces what we learn about from rewards, e.g. riding and falling from a bike).
4. **Generalization**: Policy maps (= compresses) past experience to action.

Why not always choose the best-known action in decision making?

New actions might lead to even better rewards (exploration-exploitation trade-off).

Example: Trying a new restaurant vs. Going to favorite one?

1.1.3 Course Flow

- High-level learning goals: Understand theoretical and empirical approaches for evaluating reinforcement learning algorithm quality
- Flow: First explore MDP $\implies$ model-free (policy evaluation and control) $\implies$ function approximation $\implies$ policy search + exploration

1.2 Basic Layout of RL

RL algorithms involve **State, actions, reward model and dynamics model**:

- RL aims to select sequence of actions to maximize E[future reward],
- hence involves balancing immediate and long term rewards.

Example: Choose sequence of web advertisement to maximize view time

Mathematical Formulation of RL algorithms

For each time t, a reinforcement learning agent (A) that interacts with the world (W) would...

1. A takes action a_t,
2. W updates a_t, emits observation o_t and reward r_t
3. A receives history h_t containing a_i, o_i, r_i for all time i, cumulative history is called state s_t.

In such sequential dynamic programming problems of RL, we have the following considerations:

1. **States**: Is the state Markov? Is the world partially observable?
2. **Dynamics**: Are dynamics deterministic or stochastic?
3. **Horizon of effect**: is future states taken into account?

1.2.1 States: Markov Assumption

In many cases, we have the Markov assumption:

Markov Assumption: Definition

The future is independent of past given present (i.e. state is a sufficient statistic of history):

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|h_t, a_t)$$

1.2.2 Dynamics: Model, Policy, Value Function

RL Algos must contain ≥ 1 of Model, Policy, Value Function:

Model

A MDP (Markov Decision Process) contains

1. **Transitional dynamics model**, predicting next agent state $P(s_{t+1} = s'|s_t = s, a_t = a)$
2. **Reward model**, predicting immediate reward $r(s_t = s|a_t = a) = E[r_t|s_t = s, a_t = a]$

Why Markov?

- Simple, as long as history is in part of state (in practice, assume $s_t = o_t$)
- s_t affects computational complexity, data required and result performance.

Why do we start with MDP?

Structured framework to model decision-making in uncertain environments.
$\implies$ Easier to apply RL algorithms, defining states, actions, rewards, and transitions.

Policy

Policy (denoted by π) determines how actions are chosen (i.e. $\pi : S ß A$). It can be

1. **Determinstic**: $\pi(s) = a$, or
2. **Stochastic**: $\pi(a|s) = P(a_t = a|s_t = s)$

Value Function

Value function (denoted by V^π) discounts sum of future rewards under policy π, to quantify goodness of states/ actions:

$$V^\pi(s_t = s) = \mathbb{E}_\pi \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s \right]$$

Discounting Factor Considerations

Discount factor γ weighs immediate vs future rewards:

- $\gamma = 0$: only cares about immediate;
- $\gamma = 1$: possible for finite episode length.
- To prevent infinite returns, conveniently set γ ¡ 1.

1.3 Agent and Feedback: MRP

RL agents can be either

1. **Model-based**: explicit model, may (not) have policy and value functions;
2. **Model-free**: no model, but value and/or polict functions must be explicit.

Learning Mechanisms

Given a model of world (dynamics and reward), of finite set of states and actions.
Two actions for the RL agent include:

1. **Evaluation**: Estimate/ predict expected rewards given policy.
2. **Control**: find the best policy via optimization.

Now consider the learning algorithm under Markov Chain (Markov Process):

- Markov processes involve sequences of random states with Markov property
  They are defined by (finite) set of states S and transition model P, specifying $P(s_{t+1} = s' | s_t = s)$
- In a transition model, there is no reward, hence no actions are defined.
  For finite (N) states, express P as $N \times N$ matrix.
- To bridge the stochasticity in quantifying rewards/ value, Markov Reward Processes (MRP) and Markov Decision Processes (MDP) are used.

1.3.1 Markov Reward Processes (MRP)

MRP includes the reward into consideration.

- It requires states $\mathbf{S}$, transition model $\mathbf{P}_{n \times n}$, reward $R(s_t = s)$, and discount factor $\gamma \in [0, 1]$.
- With H steps in each episode (can be infinite), the return

$$G_t = r_t + \gamma r_{t+1} + \ldots + \gamma^{H-1} r_{t+H-1}$$

- Define the state value $V(s) = E[G_t | S_t = s] = R(s) + \gamma \sum_{s' \in S} \mathbb{P}(s'|s) V(s')$.
  The first term is the immediate reward; the second term is the discounted sum of future rewards.

Obviously we are interested in computing the value of a MRP. There are two methods:

Analytical and Iterative Solutions of MRP

Method 1 (Matrix inverse) involves writing in matrix-vector forms:

- If the number of states n is finite, rewards $\mathbf{R}$ is a vector.
- Express $V(s)$ in the MRP using a matrix equation

$$\mathbf{V}_{n\times 1} = \mathbf{R}_{n\times 1} + \gamma \mathbf{P}_{n\times n}\mathbf{V}_{n\times 1}$$

- Rearranging, the solution $\mathbf{V} = (\mathbf{I} - \gamma\mathbf{P})^{-1}\mathbf{R}$.
  Direct solve takes around $O(n^3)$ time.

Method 2 (Iterative Solution) involves looping until convergence:

- Dynamic programming by initializing $V_0(s) = 0$ for all s
- For each iteration k, and all states $s \in S$: $$V_k(s) = R(s) + \gamma \sum_{s' \in S} \mathbb{P}(s'|s)V_{k-1}(s')$$
- Each iteration takes $O(n^2)$ time (because $n = |S|$).

Why $\mathbf{I} - \gamma\mathbf{P}$ is Invertible?

With $\gamma < 1$ (ensuring convergence), and the fact that $\mathbf{P}$ is stochastic (rows sum to 1):

- The eigenvalues of P are at most 1.
- When scaled by $\gamma < 1$, largest eigenvalue $\lambda_1 < \gamma = 1$.
  This ensures $\mathbf{I} - \gamma\mathbf{P}$ is full rank (diagonally dominant).

2 How to make good decisions given MDP?

2.1 Solving MDP by reducing to MRP

MDP as MRP Expansion

- MRP(S, P, R, γ) + Actions A = MDP(S, A, P, R, γ)

  Example: Consider a robot vacuum moving around a room:
  - Only models probabilities of movement without considering actions: MRP
  - Includes actions like 'turn left' or 'move forward: MDP

- MDP(S, A, P, R, γ) + Policy $\pi(a|s)$ = MRP(S, P^π, R^π, γ)
  Use same technique of computing MRP to evaluate MDP policy!
- Various settings of MDPs:
  1. **Single-state**: Bandits
  2. **Continuous states**: Optimal control
  3. **If state is history**: Partially observable MDPs (POMDPs)

We can solve MDP as an MRP as follows:

Solving MDP Iteratively

For $k = 1$ till convergence, for any $s \in S$, the state value function V is iterated with:

$$V_k^\pi(s) = \sum_a \pi(a|s)[r(s|a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{k-1}^\pi(s')] \qquad \text{where } s' \in S$$

This is called a Bellman backup equation.

- Current value at kth iteration is obtained by summing the policy probabilities varying state s, weighted by the value (current reward + future rewards discounted by one step).
- For deterministic $\pi(s)$, i.e. $P(\pi(s)|s) = 1$, the formula reduces to
$$V_k^\pi(s) = r(s|\pi(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi(s))V_{k-1}^\pi(s')$$

Where does Bellman Equation arise?

It arises naturally when breaking down long-term decision problems (by recursion) into smaller subproblems, as in policy evaluation.

2.2 How to find the optimal policy under MDP?

- Given $|A|$ distinct options for each of the $|S|$ states, there are a total of $|A|^{|S|}$ deterministic policies possible.
- Optimal policy may NOT be unique (but is deterministic and stationary (does not depend on time step) for MDP in infinite horizon!), but there exists a **unique optimal value function**.
- Such optimal policy can be found by "controlling", i.e. optimizing $\pi(s) = \arg\min_\pi V_\pi(s)$

Why is such optimal policy deterministic?

- Randomness in an MDP comes from state transitions, not from the policy itself.
- Given enough iterations, an optimal policy learns best action in every state *implies* randomness is unnecessary.

There are primarily **THREE methods** in finding the optimal policy under MDP:

2.2.1 Policy iteration (PI)

PI is more efficient than enumeration. Its process is as follows:

PI Process

(1) **Initialization**: Initialize $\pi_0(s)$ randomly for all states s.

(2) **Check L1 Norm in Loop**: Iterate when `i == 0` OR $||\pi_i - \pi_{i-1}||_1 > 0$,
i.e. *if policy changes for any state.*
Each time, compute $V_\pi(i)$ (policy evaluation), and π_{i+1} (policy improvement) to iterate new `i`

But why does PI guarantee generating better policies?

- Define the **state action value** (Q-function):
$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_\pi(s')$$
It differs from V_π that action a is taken in addition to following the policy.
- Then for new policy π_{i+1},

$$\pi_{i+1}(s) = \arg\min_a Q^{\pi_i}\pi(s,a) \forall s \in S$$

is a deterministic quantity.

- To improve the state-action value by varying a,

$$\max_a Q^{\pi_i}(s,a) \geq Q^{\pi_i}(s,\pi_i(s)) = R(s,\pi(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_i(s))V^{\pi_i}(s') = V^{\pi_i}(s)$$

This is more rigorously defined as follows:

Monotonic Improvement in Policy

Definition: $V^{\pi_1} \geq V^{\pi_2}$ if $V^{\pi_1}(s) \geq V^{\pi_2}(s)$ for all states s. (*Strict inequality* $V^{\pi_{i+1}} > V^{\pi_i}$ *if* π_i *is suboptimal.*)

Implication: By taking $\pi_{i+1}(s)$ for one action and following π_{i+1} forever, expected sum of rewards is at least as good as always following π_i. Since we always pick a better action, policies never get worse!

Regarding change (and iteration) of policies in PI

- If policy doesn't change, it cannot be changed anymore.
- There is a maximum number of policy iterations (since a finite number of options for each finite state).

Why state action value is defined? Why store Q instead of V?

- It evaluates the expected return taking action a a first, then act optimally afterward.
- Instead of storing $V(s)$, store $Q(s,a)$ to directly optimize actions.

2.2.2 Value iteration (VI)

The idea is to maintain **optimal value** of starting in state s, if finite number of steps k left in episode.
Then, iterate to consider longer and longer episodes:

Bellman Operator

For a value function, the Bellman Operator B returns a new value that improves if possible:

$$BV(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s') \right]$$

For a policy,

$$B^\pi V(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s)V(s')$$

Example: Updating est. travel time in Google Maps as new traffic data arrives (i.e. iteratively refines value estimates).

How to "maintain optimal value"? Why does a disappear?

- Maintaining optimal value means ensuring that for every state, compute best expected return by considering all possible actions.
- As we only keep the highest value across actions, we remove a.
- We only care about values during iteration: Extract optimal policy by checking which action led to that value

VI Process

(1) **Initialization**: $k = 1$, $V_0(s) = 0$ for all states s.

(2) **Check Norm in Loop**: Loop until convergence (i.e. checking $L_\infty = ||V_{k+1} - V_k||_\infty \leq \epsilon$)

For each state s in S, iterate $V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]$

This iteration step is equivalently $V_{k+1} = BV_k$

For the optimal policy for finite horizon H, iterate (loop) from $i = 0 : H$ (exclusive of H):

- $V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]$
- $\pi_{k+1}(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]$

But why does VI guarantee convergence?

- Quantify convergence with contraction operator O, satisfying

$$||OV - OV'||_n \leq ||V - V'||_n \text{ for any } L_n \text{ norm.}$$

- Converges when $\gamma < 1$ OR ending up in terminal state with probability 1!
- VI converges to **unique solution** for discrete state and action spaces
- Policy evaluation = compute **fixed point** of B_π, by repeatedly applying operator until V stops changing, i.e.

$$V^\pi = B^\pi B^\pi \dots B^\pi V$$

What does "fixed point" here mean?

- Applying the operator doesn't change the result anymore.
- Signifies convergence (e.g. in PI, determine whether the policy should be updated).

Does the initialization of values in value iteration impact anything?

- It only affects convergence speed, but same final result.
- Because Bellman backup is a contraction mapping.

A brief summary of PI vs VI

Type	PI	VI
To compute...	Infinite horizon value	Optimal finite k-horizon value
Use	Select another (better) policy	Find optimal policy, by incrementing time horizon
Convergence	#Iterations $\leq$ Enumeration of all deterministic policies, i.e. of size $\lvert A\rvert^{\lvert S\rvert}$	Depends on discounting factor γ , possible to exceed $\lvert A\rvert^{\lvert S\rvert}$

2.3 Monte Carlo (MC) Methods for MDP

Given dynamics and reward models, can do policy evaluation through DP.

$$V_k^\pi(s) = r(s|\pi(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) V_{k-1}^\pi(s')$$

Here, V_k^π is an estimate of V^π, and the second part can be replaced by **bootstrapping** if dynamics/ reward models are unknown but deterministic : when there is no access to true MDP.

- MC follows policy π to generate sample trajectories and take expectation on sample return to approximate (estimate) $V^{\pi}(s)$.

- Rationale: Mean return converges to value (by LLN)
  Note: All trajectories may not be of the same length because of MDP.

- Requires episodic settings (terminal state/ finite horizon) to end episode
  $\implies$ Terminate episode before averaging.

Advantages of MC

- Weak assumption (No need Markov assumption/ knowing MDP dynamics and rewards)
- Sometimes preferred over DP for policy evaluation even with known dynamics model and reward. (Explain why?)

The process and variations of MC is as follows:

MC Process

1. **Initialize**: Number of samples $N(s) = 0$, grand total $G(s) = 0$ for each state $s \in S$.
2. **Loop**:
   - For each episode i, sample trajectories $(s_{i,1}, a_{i,1}, r_{i,1}), (s_{i,2}, a_{i,2}, r_{i,2}), \dots, s_{i,t_i}$
   - Compute return value $G_{i,t}$ as return from time step t:
     $$G_{i,t} = r_{i,t} + \gamma r_{i,t+1} + \cdots + \gamma^{T_i - t} r_{i,T_i}$$
   - Update state values for each time step t until end of episode i:
     Increment # visits: $N(s) = N(s) + 1$
     Increment total return: $G(s) = G(s) + G_{i,t}$
     Update estimate $V^{\pi}(s) = G(s)/N(s)$

There are **Three variations** for MC:

Variations of MC

(i) First-time MC: Increment $N(s)$ for the first time visiting state s in episode i, i.e. $N(s) = 1$
(ii) Every-time MC: Increment $N(s)$ each time visiting state s in episode i
(iii) Incremental MC: After obtaining the new return value $G_{i,t}$, rewrite the update from $V^{\pi}(s)$ to $V^{\pi}(s)'$:

$$\begin{aligned}
V^{\pi}(s)' &= \frac{G(s)'}{N(s)'} = \frac{G(s) + G_{i,t}}{N(s)+1} = \frac{G(s)}{N(s)+1} + \frac{G_{i,t}}{N(s)+1} \\
&= V^{\pi}(s) \times \frac{N(s)}{N(s)+1} + \frac{G_{i,t}}{N(s)+1} \\
&= V^{\pi}(s) + \frac{1}{N(s)+1}\left(G_{i,t} - V^{\pi}(s)\right) \\
V^{\pi}(s)' &= V^{\pi}(s) + \alpha\left(G_{i,t} - V^{\pi}(s)\right)
\end{aligned}$$

The general form replaces the learning rate $\alpha = 1/[N(s)+1]$ to other rates. Viewed from this perspective:

MC Type	Every Visit	First Visit	General in
α	$\frac{1}{N(s_{i,t})}$	1 if $N(s) = 0$, 0 otherwise	V
Properties	Biased, but consistent and offers better MSE	Unbiased estimator of $V^{\pi}(s)$ by LLN	Rate larger than $1/N(s_{i,t})$ is

Q: What is an intuitive explanation to such every-visit "bias"? What does it mean by helpful in non-stationary domains?

2.4 Evaluating Policy Estimation Quality

We take theoretical and computational aspects for consideration:

- Theoretical: Statistical efficiency, consistency and empirical accuracy (MSE)
- Computation: Runtime and Memory Complexity

2.4.1 Bias and Variance

The empirical accuracy of learning models is quantified by Mean Squared Error (MSE):

$$\text{MSE}(\hat{\theta}) = \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta})^2$$

- $\text{Var}(\hat{\theta}) = \mathbb{E}_{x|\theta}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$
- $\text{Bias}(\hat{\theta})^2 = (\mathbb{E}_{x|\theta}[\hat{\theta}] - \theta)^2$

Consistency quantifies how reliable an estimator is:

- If there are n data points of x, $\lim\limits_{n\to\infty} \mathbb{P}(|\hat{\theta}_n - \theta| > \epsilon) = 0$ for arbitrary ϵ.
- In other words, estimates (produced by the estimator) "converge" to the true parameter value.

An unbiased estimator may not be consistent. Two simple examples:

- The variance estimator $\dfrac{1}{n}\sum(y_i - y_n)^2$ is biased but consistent.
- The data point X_1 as an estimator of μ for $N(\mu, \sigma^2)$ is unbiased but NOT consistent.

2.4.2 How does MC fare under those criteria?

(1) On Convergence: For MC, under varying learning rate α, incremental MC will converge to true value of policy, $V^\pi(s_j)$ if

1. $\sum\limits_{n=1}^{\infty} \alpha_n(s_j) = \infty$, but
2. $\sum\limits_{n=1}^{\infty} \alpha_n^2(s_j) < \infty$.

(2) On MSE: MC yields a high-variance estimator that takes a lot of data to reduce.

3 Value Function Approximation

Recall in optimizing policy π, we can either **evaluate** (compute Q^π) or **improve** (update π given Q^π).

Problem: Deterministic policies in approximation yield no exploration!

- If π deterministic, $Q(s,a)$ is undefined for $a \neq \pi(s)$. Can't learn about actions without trying them!
- Exploitation of past experience that proved high reward.

Solution: Assign probability on unvisited pathways!

- ϵ-greedy pathways selects argmax action with $1-\epsilon$ probability, else random. i.e. $\pi(a|s) = \begin{cases} \arg\max\limits_a Q(s,a), & \text{probability } 1 \\ a' \neq \arg\max\limits_a Q(s,a), & \text{probability } \end{cases}$
- ϵ-greedy policy guarantees monotonic improvement, and can estimate $Q(s,a) \forall s, a$.

3.0.1 Greedy in Limit of Infinite Exploration (GLIE)

GLIE: If $\lim_{i \to \infty} N_i(s,a) \to \infty$ (i.e. all state-action pairs visited infinite times)

- Behavior policy (policy used to act in the world) converges to greedy policy
- GLIE is ϵ-greedy if $\epsilon = \frac{1}{i} \to 0$.
- MC under GLIE converges to optimal value, i.e. $Q(s,a) \to Q^*(s,a)$

Q: What is the convergence to optimal Q^* function and the expected performance of MC control with ϵ-greedy policy? When may MC online control fail to compute the optimal Q^*?

Other than on-policy learning (through direct experience following the policy), **off-policy** learning exists (experience gathered from following a different policy).
In particular, Q-learning is a popular area of off-policy learning:

Q-learning Definition and Properties

- Q-learning estimates Q^{π^*} by acting with another π_b by maintaining Q estimates and bootstrapping.
- Update step:
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma \arg\max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \right)$$
where $\arg\max_{a'} Q(s_{t+1}, a')$ is by following another behaviour policy.
- Take $a_t \sim \pi_b(s_t)$: Sample action from ϵ-greedy policy. Here π_b is ϵ-greedy w.r.t. Q.
- It builds results through stochastic approximation, decreasing step sizes to satisfy Bellman contraction property, and utilizing bounded rewards and value function.

Q-learning for finite-state and finite-action MDPs converges to optimal value $Q^*(s,a)$ given:

- Policy sequence $\pi_t(a|s)$ satisfies conditions of GLIE.
- $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

3.1 Function Approximation

Function **approximation** carries the following two purposes:

1. Avoid explicit storing/ learning for every single state and action (model/ value/ policy)
2. Compact representation, generalized across states and actions (i.e. less memory and computation needed)

There are **THREE** forms of approximation: bootstrapping, sampling, and VFA.

With a model, we carry out VFA by assuming we can "query (s,a)" to return $Q^\pi(s,a)$, i.e. supervised learning:

Model-based VFA: Process

- **Objective**: **approximate representation** of Q^π given parameterized form $\hat{Q}(s, a; \mathbf{w})$
- **Input**: $\mathbf{w}$ is a parameter vector of weights.
- **Method**: Minimize loss function between $Q^\pi(s, a)$ and approximation $\hat{Q}(s, a; \mathbf{w})$:

$$J(\mathbf{w}) = \mathbb{E}_\pi[\Big(Q^\pi(s, a) - \hat{Q}(s, a; \mathbf{w})\Big)^2]$$

- **Means**: Find local minimum with gradient descent

$$\delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w})$$

- **Improvement**: Stochastic gradient descent (SGD) uses finite (often one) sample to approximate gradient

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -2\mathbb{E}_\pi[Q^\pi(s, a) - \hat{Q}(s, a; \mathbf{w})]\nabla_{\mathbf{w}}\hat{Q}$$

How does SGD differ from GD? What is "finite # samples"? [See answer from GPT]

3.1.1 VFA without Model

Without a model (oracle), follow policy π/ prior data to estimate V^π and Q^π.

Model-free VFA: Process

- **Input**: Generated sample paths by following fixed policy π.
- **Method**: Through lookup table storing estimates of V^π or Q^π
  $\rightarrow$ Update estimates after each episode (MC)/ step (TD)
- Change the estimate update step to include fitting function approximator.
  How? Return G_t is unbiased but noisy sample of true $Q^\pi(s_t, a_t)$
  $\implies$ Do supervised learning on $<(s_i, a_i), G_i>$.

3.2 VFA under TDL

Recall that TDL uses bootstrapping and sampling to approximate V^π:

$$V^\pi(s) = V^\pi(s) + \alpha\left(r + \gamma V^\pi(s') - V^\pi(s)\right)$$

where the temporal target $r + \gamma V^\pi(s')$ is **biased** against $V^\pi(s)$.

- When with a look table, represent each state value with a separate table entry.
- in VFA, target is $r + \gamma V^\pi(s'; \mathbf{w})$, also **biased** against $V^\pi(s)$.
- Equivalent to supervised learning on $<(s_i, r_i), \gamma\hat{V}^\pi(s_{i+1}; \mathbf{w})>$ and finding weights $\mathbf{w}$ to minimize MSE with SGD:

$$J(\mathbf{w}) = \mathbb{E}_\pi[\Big(r_j + \gamma V^\pi(s_{j+1}, \mathbf{w}) - \hat{V}(s_j; \mathbf{w})\Big)^2]$$

Question: Why the function involve V here but not Q like in model-based VFA?

3.2.1 Control using VFA

Intuitively the few main points are:

- **Estimation**: $Q^\pi(s, a; \mathbf{w}) \approx Q^\pi$ by **sampling** gradients from SGD, for local minimum.

- **Interleaving**: Perform ϵ-greedy policy improvement between VFA approximations.

There are a few incremental approaches to determine the increment $\Delta\mathbf{w}$ for action-value function approximation:

1. **General form**: use target value

$$\Delta\mathbf{w} = \alpha(Q(s_t, a_t) - \hat{Q}(s_t, a_t; \mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}(s_t, a_t; \mathbf{w})$$

2. **MC Simulation**: use return G_t as substitute target

$$\Delta\mathbf{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}(s_t, a_t; \mathbf{w})$$

3. **SARSA**: use TD target, i.e. $r + \gamma\hat{Q}(s', a'; \mathbf{w})$

$$\Delta\mathbf{w} = \alpha(r + \gamma\hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}(s, a; \mathbf{w})$$

4. **Q-Learning**: use related TD target to SARSA

$$\Delta\mathbf{w} = \alpha(r + \gamma \arg\max_{a'} Q(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}(s, a; \mathbf{w})$$

Q: What is the original purpose of interleaving? How does it cause problems? Q: Recall, how is the target value in general form VFA increment determined? Q: Recall why is there t in MC/ General form for s and a but NOT SARSA/ Q-learning? Q: Recall: What are the major differences between SARSA and Q-learning by tweaking the TD target?

3.3 Deadly Triad and Remedies

Applying bootstrapping, Function Approximation and Off-policy Learning leads to oscillations and non-convergence!

- Reason: Bellman operators on updates are contractions, but VFA may be expansions.
- Q-learning with VFA diverges with correlation between samples and non-stationary targets.

Solution presented includes **Deep Q-learning Network (DQN)**, featuring replay and fixed Q-targets.

Remedy 1: DQN with Replay

Extract sample experience tuples $(s_i, a, r, s_{i+1}) \sim D$ (a replay buffer from prior experience)

- Target value: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$; plug into SGD
- The replay buffer is usually of fixed size
- SGD to update network weights: $\Delta\mathbf{w} = \alpha \left(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w})\right) \nabla_{\mathbf{w}}\hat{Q}(s, a; \mathbf{w})$

Remedy 2: Fix Target Weights

Fix the target weights used in the target calculation for multiple updates

- Different set of target network weights ($\mathbf{w}^-$) than the weights being updated ($\mathbf{w}$)
- Fix $\mathbf{w}^-$ for multiple updates before replacing it with new $\mathbf{w}$ for next multiple steps
- Target value: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-)$

3.4 DQN in Practice with Atari

Below discusses an example in practice:

Workflow of applying DQN in Atari

- Learn $Q(s,a)$ from pixel s;
- Input state s = raw pixels from last 4 frames
- Output: $Q(s,a)$ for all (joystick, button)
- Reward: Δ Score
- Architecture: Deep NN with CNN

Outcome: DQN mostly outperforms best linear learners!

Caveats of Atari DQN

- Deep network *without* fixed Q in fact hurts performance! (worse than linear)
- Replays are hugely important!
- Immediate improvements: Double DQN (Van Hasselt, 2016), Prioritized Replay (Schaul et. al., 2016), Dueling DQN (Wang et. al., 2016)

Q: ”Bellman operators are contractions, but value function approximation fitting can be an expansion”. What does this mean? Q: How are replays hugely important beyond decorrelating between samples?