Lecture Outline

- Curse of dimensionality and generalisation
- Value function approximation
- Stochastic gradient descent
- Linear value functions and feature construction
- Semi-gradient TD control
Curse of Dimensionality

Theory so far has assumed:

- **Unlimited space**: can store value function as table
- **Unlimited data**: many (infinite) visits to all state-action pairs

In practice these assumptions are usually violated, because...

**Curse of Dimensionality:**

- Number of states grows *exponentially* with number of state variables
- If state described by $k$ variables with values in $\{1, \ldots, n\}$, then $O(n^k)$ states

Go: $10^{170}$ states

Hydrogen atoms: $10^{80}$
Two problems...

- Not enough memory to store value function as table.

\[ v(s) / q(s; a) \]

- Need compact representation of value function. (But sometimes can be enough to store only partial value function; e.g. MCTS.)

- No data (or not enough data) to estimate return in each state.

- Many states may never be visited.

- Need to generalise observations to unknown state-action pairs.
Compact Value Functions and Generalisation

Two problems...

**Not enough memory to store value function as table**

- Tabular $v(s)/q(s, a)$ use storage proportional to $|S|$
- Need **compact representation** of value function
  (But sometimes can be enough to store only partial value function; e.g. MCTS)
Compact Value Functions and Generalisation

Two problems...

Not enough memory to store value function as table

- Tabular $v(s)/q(s, a)$ use storage proportional to $|S|$
- Need compact representation of value function
  (But sometimes can be enough to store only partial value function; e.g. MCTS)

No data (or not enough data) to estimate return in each state

- Many states may never be visited
- Need to generalise observations to unknown state-action pairs
Blue circle must move to red goal
- Agent uses optimal policy (shortest path)

Suppose we have return estimates (steps to go) for locations S1–S6
- e.g. $v(S5) = -3$, $v(S4) = -6$, $v(S2) = -31$

We have no data for locations S7 and S8 (not visited yet)
- Can we estimate $v(S7)$ and $v(S8)$ based on other return estimates?
Replace tabular value function with *parameterised function*:

\[
\hat{v}(s, w) \approx v_\pi(s)
\]

\[
\hat{q}(s, a, w) \approx q_\pi(s, a)
\]

\(w \in \mathbb{R}^d\) is parameter ("weight") vector

e.g. linear function, neural network, regression tree, ...

- **Compact**: number of parameters \(d\) much smaller than \(|S|\)
- **Generalises**: changing one parameter value may change value estimate of many states/actions
Learning a value function is a form of supervised learning:

Examples are pairs of states and return estimates, \((S_t, U_t)\), e.g.

- **MC**: \( U_t = G_t \)
- **TD(0)**: \( U_t = R_{t+1} + \gamma \hat{V}(S_{t+1}, w_t) \)
- **n-step TD**: \( U_t = R_{t+1} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n \hat{V}(S_{t+n}, w_{t+n-1}) \)
Desired properties in supervised learning method:

- **Incremental updates**
  
  update \( w \) using only partial data, e.g. most recent \((S_t, U_t)\) or subset
Desired properties in supervised learning method:

- **Incremental updates**
  
  update $w$ using only partial data, e.g. most recent $(S_t, U_t)$ or subset

- **Ability to handle noisy targets**
  
  e.g. different MC updates $G_t$ for same state $S_t$
Desired properties in supervised learning method:

- **Incremental updates**
  update $w$ using only partial data, e.g. most recent $(S_t, U_t)$ or subset

- **Ability to handle noisy targets**
  e.g. different MC updates $G_t$ for same state $S_t$

- **Ability do handle non-stationary targets**
  e.g. changing target policy, bootstrapping

$\Rightarrow$ If $\hat{v}/\hat{q}$ differentiable, **stochastic gradient descent** is suitable method
Gradient Descent

- Let $J(\mathbf{w})$ be differentiable function of $\mathbf{w}$

- Gradient of $J(\mathbf{w})$ is
  
  $$\nabla J(\mathbf{w}) = \left( \frac{\partial J(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_d} \right)^T$$

- To find local minimum of $J(\mathbf{w})$, adjust $\mathbf{w}$ in negative direction of gradient
  
  $$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2} \alpha \nabla J(\mathbf{w}_t)$$

- $\alpha$ is step-size parameter
  convergence requires standard $\alpha$-reduction
Objective: find parameter vector $\mathbf{w}$ by minimising \textit{mean-squared error} between approximate value $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2]$$
Stochastic Gradient Descent

**Objective:** find parameter vector $\mathbf{w}$ by minimising *mean-squared error* between approximate value $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$

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

- Gradient descent finds local minimum:
  $$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2} \alpha \nabla J(\mathbf{w}_t)$$
  $$= \mathbf{w}_t + \alpha \mathbb{E}_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w}_t)) \nabla \hat{v}(s, \mathbf{w}_t)]$$
Stochastic Gradient Descent

Objective: find parameter vector $\mathbf{w}$ by minimising mean-squared error between approximate value $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$

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

- Gradient descent finds local minimum:
  $$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2} \alpha \nabla J(\mathbf{w}_t)$$
  $$= \mathbf{w}_t + \alpha \mathbb{E}_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w}_t)) \nabla \hat{v}(s, \mathbf{w}_t)]$$

- Stochastic gradient descent samples the gradient:
  $$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha [U_t - \hat{v}(S_t, \mathbf{w}_t)] \nabla \hat{v}(S_t, \mathbf{w}_t)$$
Stochastic gradient descent samples the gradient:

$$w_{t+1} = w_t + \alpha [U_t - \hat{\nabla}(S_t, w_t)] \nabla \hat{\nabla}(S_t, w_t)$$  \hspace{1cm} (1)
Stochastic Gradient Descent — Convergence

Stochastic gradient descent *samples* the gradient:

\[ w_{t+1} = w_t + \alpha [U_t - \hat{v}(S_t, w_t)] \nabla \hat{v}(S_t, w_t) \]

(1)

- \( w_t \) will converge to **local optimum** under standard \( \alpha \)-reduction and if \( U_t \) is unbiased estimate \( \mathbb{E}_\pi[U_t|S_t] = v_\pi(S_t) \)

\( \Rightarrow \) MC update is unbiased but TD update is biased (why?)
Stochastic Gradient Descent — Convergence

Stochastic gradient descent samples the gradient:

\[ w_{t+1} = w_t + \alpha [U_t - \hat{V}(S_t, w_t)] \nabla \hat{V}(S_t, w_t) \]  \hspace{2cm} (1)

- \( w_t \) will converge to local optimum under standard \( \alpha \)-reduction and if \( U_t \) is unbiased estimate \( \mathbb{E}_\pi[U_t|S_t] = v_\pi(S_t) \)

\[ \Rightarrow \text{MC update is unbiased but TD update is biased (why?)} \]

- Note: (1) is not a true TD gradient because \( U_t \) also depends on \( w \)

\[ U_t = R_{t+1} + \gamma \hat{V}(S_{t+1}, w) \]

Hence, we call it semi-gradient TD
Semi-gradient TD(0) for Policy Evaluation

Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : S^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)

Loop for each episode:
  Initialize $S$
    Loop for each step of episode:
      Choose $A \sim \pi(\cdot|S)$
      Take action $A$, observe $R, S'$
      $w \leftarrow w + \alpha [R + \gamma \hat{v}(S', w) - \hat{v}(S, w)] \nabla \hat{v}(S, w)$
      $S \leftarrow S'$
    until $S$ is terminal
Linear value function approximation:

\[ \hat{v}(s, w) = w^\top x(s) = \sum_{i=1}^{d} w_i x_i(s) \]

- \( x(s) = (x_1(s), ..., x_d(s))^\top \) is feature vector of state \( s \)
- Simple gradient: \( \nabla \hat{v}(s, w) = \left( \frac{\partial w^\top x}{\partial w_1}, \ldots, \frac{\partial w^\top x}{\partial w_d} \right)^\top = x(s) \)
- Gradient update: \( w_{t+1} = w_t + \alpha [U_t - \hat{v}(S_t, w_t)] x(S_t) \)
Linear Value Function Approximation

Linear value function approximation:

\[ \hat{V}(s, w) \doteq w^\top x(s) = \sum_{i=1}^{d} w_i x_i(s) \]

- \( x(s) = (x_1(s), ..., x_d(s))^\top \) is feature vector of state \( s \)
- Simple gradient: \( \nabla \hat{V}(s, w) = \left( \frac{\partial w^\top x}{\partial w_1}, \ldots, \frac{\partial w^\top x}{\partial w_d} \right)^\top = x(s) \)
- Gradient update: \( w_{t+1} = w_t + \alpha [U_t - \hat{V}(S_t, w_t)] x(S_t) \)

In linear case, there is only one optimum!

\( \Rightarrow \) MC gradient updates converge to global optimum
\( \Rightarrow \) TD gradient updates converge near global optimum (TD fixed point)

See Tutorial 6
Feature Vectors

Remember:
State must be Markov

$x(s) = \begin{pmatrix} x\text{-pos}(s) \\ y\text{-pos}(s) \end{pmatrix}$

$x(s) = \begin{pmatrix} \theta(s) \\ \theta\text{-vel}(s) \\ x\text{-pos}(s) \\ \vdots \end{pmatrix}$
Exact representation:

\[ x(s) = \begin{pmatrix} x\text{-pos}(s) \\ y\text{-pos}(s) \end{pmatrix} \]

Generalise with state aggregation:

- Partition states into disjoint sets \( S_1, S_2, \ldots \) with indicator functions \( x_k(s) = [s \in S_k]_1 \)

\[ x(s) = \begin{pmatrix} \text{in-}S_1(s) \\ \text{in-}S_2(s) \\ \text{in-}S_3(s) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \]
State Aggregation

Exact representation:

\[ x(s) = \begin{pmatrix} x\text{-}pos(s) \\ y\text{-}pos(s) \end{pmatrix} \]

Generalise with state aggregation:

- Partition states into disjoint sets \( S_1, S_2, \ldots \) with indicator functions \( x_k(s) = [s \in S_k]_1 \)

Special case: every state \( s \) has its own set \( S_s = \{s\} \)

\( \implies \) Same as tabular representation!
Coarse/Tile Coding

State aggregation generalises only within sets $S_1, S_2, ...$

- Allow generalisation *across* sets by allowing $S_k$ to overlap
- e.g. coarse coding and tile coding

![Diagram](image-url)
Example: Random Walk

- States: numbered 1 to 1000, start at state 500
- Policy: randomly jump to one of 100 states to left, or one of 100 states to right
- If jump goes beyond 1/1000, terminates with reward \(-1/+1\)
- State aggregation: 10 groups of 100 states each
Random Walk: MC and TD Prediction

Linear gradient MC:

Approximate MC value $\hat{v}$

State distribution $d$

Value scale

-1 0 1

State

1000

Linear gradient TD:

Approximate TD value $\hat{v}$

True value $v_\pi$

Distribution scale

-1 0 1

After 100,000 episodes with $\alpha = 2 \times 10^{-5}$
Approximate Control in Episodic Tasks

- Estimate state-action values: \( \hat{q}(s, a, w) \approx q_\pi(s, a) \)

- For linear approx., features defined over states and action:
  \[
  \hat{q}(s, a, w) = \sum_{i=1}^{d} w_i x_i(s, a)
  \]

- Stochastic gradient descent:
  \[
  w_{t+1} = w_t + \alpha [U_t - \hat{q}(S_t, A_t, w_t)] \nabla \hat{q}(S_t, A_t, w_t)
  \]
Approximate Control in Episodic Tasks

- Estimate state-action values: \( \hat{q}(s, a, w) \approx q_\pi(s, a) \)

- For linear approx., features defined over states and action:
  \[
  \hat{q}(s, a, w) = \sum_{i=1}^{d} w_i x_i(s, a)
  \]

- Stochastic gradient descent:
  \[
  w_{t+1} = w_t + \alpha \left[ U_t - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{q}(S_t, A_t, w_t)
  \]

  e.g. **Sarsa**: \( U_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) \)

  **Q-learning**: \( U_t = R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w_t) \)

  **Expected Sarsa**: \( U_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, w_t) \)
Episodic Semi-gradient Sarsa

Input: a differentiable action-value function parameterization \( \hat{q} : S \times A \times \mathbb{R}^d \to \mathbb{R} \)

Algorithm parameters: step size \( \alpha > 0 \), small \( \varepsilon > 0 \)

Initialize value-function weights \( \mathbf{w} \in \mathbb{R}^d \) arbitrarily (e.g., \( \mathbf{w} = \mathbf{0} \))

Loop for each episode:
- \( S, A \leftarrow \) initial state and action of episode (e.g., \( \varepsilon \)-greedy)

Loop for each step of episode:
  - Take action \( A \), observe \( R, S' \)
  - If \( S' \) is terminal:
    - \( \mathbf{w} \leftarrow \mathbf{w} + \alpha \left[ R - \hat{q}(S, A, \mathbf{w}) \right] \nabla \hat{q}(S, A, \mathbf{w}) \)
    - Go to next episode
  - Choose \( A' \) as a function of \( \hat{q}(S', \cdot, \mathbf{w}) \) (e.g., \( \varepsilon \)-greedy)
  - \( \mathbf{w} \leftarrow \mathbf{w} + \alpha \left[ R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w}) \right] \nabla \hat{q}(S, A, \mathbf{w}) \)
  - \( S \leftarrow S' \)
  - \( A \leftarrow A' \)
Example: Mountain Car with Linear Semi-Gradient Sarsa

STATES:
- car's position and velocity

ACTIONS:
- three thrusts: forward, reverse, none

REWARDS:
- always $-1$ until car reaches the goal

Episodic, No Discounting, $\gamma=1$

Semi-gradient Sarsa with linear approximation over 8 8x8 tilings

$\epsilon = 0$ (optimistic initial values $\hat{q}(s, a, w) = 0$)
Learned Action Values in Mountain Car

Cost-to-go:
\[- \max_a \hat{q}(s, a, w)\]
Mountain Car
Steps per episode
log scale
averaged over 100 runs

Learning Curves in Mountain Car
Convergence to Global Optimum in Episodic Control

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Tabular</th>
<th>Linear</th>
<th>Non-linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC control</td>
<td>yes</td>
<td>chatter*</td>
<td>no</td>
</tr>
<tr>
<td>(semi-gradient) n-step Sarsa</td>
<td>yes</td>
<td>chatter*</td>
<td>no</td>
</tr>
<tr>
<td>(semi-gradient) n-step Q-learning</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

*Chatters near optimal solution because optimal policy may not be representable under value function approximation*
Deadly Triad

Risk of divergence arises when the following three are combined:

1. Function approximation
2. Bootstrapping
3. Off-policy learning

Possible fixes:

- Use importance sampling to warp off-policy distribution into on-policy distribution
- Use gradient TD methods which follow true gradient of projected Bellman error (see book)
Required (RL book):

- Chapter 9 (9.1–9.5)
  (Box “Proof of Convergence of Linear TD(0)” in Sec 9.4 is not examined)
- Chapter 10 (10.1)
- Chapter 11 (11.1)

Optional:

- Remaining sections of chapters