1. Generalisation and function approximation

2. Linear function approximation

3. Linear TD($\lambda$)
Half Field Offense


How many states are there?

An infinite number!

Shivaram Kalyanakrishnan (2022)
CS 747, Autumn 2022
Half Field Offense

- Decision-making restricted to offense player with ball.
- Based on state, choose among Dribble, Pass, Shoot.
Decision-making restricted to offense player with ball.
- Based on state, choose among Dribble, Pass, Shoot.
- How many states are there?
Half Field Offense

- Decision-making restricted to offense player with ball.
- Based on state, choose among DRIBBLE, PASS, SHOOT.
- How many states are there? An infinite number!
Half Field Offense

- Decision-making restricted to offense player with ball.
- Based on state, choose among DRIBBLE, PASS, SHOOT.
- How many states are there? An infinite number!
- What to do?
Features

- State $s$ is defined by positions and velocities of players, ball.

1. $x_1(s)$: Distance to teammate.
2. $x_2(s)$: Distance to nearest opponent.
3. $x_3(s)$: Largest open angle to goal.
4. $x_4(s)$: Distance of teammate to goal.
Features

- State $s$ is defined by positions and velocities of players, ball.
- Velocities might not be important for decision making.
- Position coordinates might not generalise well.

Define features $x$:

- $x_1(s)$: Distance to teammate.
- $x_2(s)$: Distance to nearest opponent.
- $x_3(s)$: Largest open angle to goal.
- $x_4(s)$: Distance of teammate to goal.
Features

- State $s$ is defined by positions and velocities of players, ball.
- Velocities might not be important for decision making.
- Position coordinates might not generalise well.
- Define features $x : S \rightarrow \mathbb{R}$. Idea is that states with similar features will have similar consequences of actions, values.

- $x_1(s)$: Distance to teammate.
- $x_2(s)$: Distance to nearest opponent.
- $x_3(s)$: Largest open angle to goal.
- $x_4(s)$: Distance of teammate to goal.
Compact Representation of $\hat{Q}$
- Illustration of $\hat{Q}$ approximated using a neural network.
- Input: (features of) state. One output for each action.
- Similar states will have similar $Q$-values.
- Can we learn weights $w$ so that $\hat{Q}(s, a) \approx Q^*(s, a)$?

$$s \xrightarrow{w} x_1(s), \; x_2(s), \; x_3(s), \; x_4(s) \xrightarrow{\sum, \sigma} \hat{Q}(s, a_1), \; \hat{Q}(s, a_2), \; \hat{Q}(s, a_3)$$

Might not be able to represent $Q^*$!

Unlike supervised learning, convergence not obvious!

Even if convergent, might induce sub-optimal behaviour!
Compact Representation of $\hat{Q}$

- Illustration of $\hat{Q}$ approximated using a neural network.
- Input: (features of) state. One output for each action.
- Similar states will have similar $Q$-values.
- Can we learn weights $w$ so that $\hat{Q}(s, a) \approx Q^*(s, a)$?

$$
\begin{align*}
\sigma &\quad \sum , \quad \sigma \\
\sigma &\quad \sum , \quad \sigma \\
\sigma &\quad \sum , \quad \sigma \\
\sigma &\quad \sum , \quad \sigma \\
\end{align*}
$$

- Might not be able to represent $Q^*$!
- Unlike supervised learning, convergence not obvious!
- Even if convergent, might induce sub-optimal behaviour!
Reinforcement Learning

1. Generalisation and function approximation

2. Linear function approximation

3. Linear TD(\(\lambda\))
Prediction with a Linear Architecture

- Suppose we are to evaluate $\pi$ on MDP $(S, A, T, R, \gamma)$.
- Say we choose to approximate $V^\pi$ by $\hat{V}$: for $s \in S$,

$$\hat{V}(w, s) = w \cdot x(s),$$

where $x : S \rightarrow \mathbb{R}^d$ is a $d$-dimensional feature vector, and $w \in \mathbb{R}^d$ is the weight/coefficient vector.
Prediction with a Linear Architecture

- Suppose we are to evaluate \( \pi \) on MDP \((S, A, T, R, \gamma)\).
- Say we choose to approximate \( V^\pi \) by \( \hat{V} \): for \( s \in S \),

\[
\hat{V}(w, s) = w \cdot x(s),
\]

where

\[
x: S \rightarrow \mathbb{R}^d \text{ is a } d\text{-dimensional feature vector, and}
\]

\[
w \in \mathbb{R}^d \text{ is the weight/coefficient vector.}
\]

- Usually \( d \ll |S| \).
- Illustration with \( |S| = 3, d = 2 \). Take \( w = (w_1, w_2) \).

<table>
<thead>
<tr>
<th>( s )</th>
<th>( V^\pi(s) )</th>
<th>( x_1(s) )</th>
<th>( x_2(s) )</th>
<th>( \hat{V}(w, s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>( 2w_1 - w_2 )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>( 4w_1 )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>-4</td>
<td>2</td>
<td>3</td>
<td>( 2w_1 + 3w_2 )</td>
</tr>
</tbody>
</table>
The Best Approximation

<table>
<thead>
<tr>
<th></th>
<th>$V_\pi(s)$</th>
<th>$x_1(s)$</th>
<th>$x_2(s)$</th>
<th>$\hat{V}(w, s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>$2w_1 - w_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>$4w_1$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>-4</td>
<td>2</td>
<td>3</td>
<td>$2w_1 + 3w_2$</td>
</tr>
</tbody>
</table>

- Observe that for all $w \in \mathbb{R}^2$, $\hat{V}(w, s_2) = \frac{3\hat{V}(w, s_1) + \hat{V}(w, s_3)}{2}$.
- In general, $\hat{V}$ cannot be made equal to $V_\pi$.
The Best Approximation

<table>
<thead>
<tr>
<th>$s$</th>
<th>$V_\pi(s)$</th>
<th>$x_1(s)$</th>
<th>$x_2(s)$</th>
<th>$\hat{V}(w, s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>$2w_1 - w_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>$4w_1$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>-4</td>
<td>2</td>
<td>3</td>
<td>$2w_1 + 3w_2$</td>
</tr>
</tbody>
</table>

Observe that for all $w \in \mathbb{R}^2$, $\hat{V}(w, s_2) = \frac{3\hat{V}(w, s_1) + \hat{V}(w, s_3)}{2}$.

In general, $\hat{V}$ cannot be made equal to $V_\pi$.

Which $w$ provides the best approximation?
The Best Approximation

<table>
<thead>
<tr>
<th>s</th>
<th>$V^\pi(s)$</th>
<th>$x_1(s)$</th>
<th>$x_2(s)$</th>
<th>$\hat{V}(w, s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>$2w_1 - w_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>$4w_1$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>-4</td>
<td>2</td>
<td>3</td>
<td>$2w_1 + 3w_2$</td>
</tr>
</tbody>
</table>

- Observe that for all $w \in \mathbb{R}^2$, $\hat{V}(w, s_2) = \frac{3\hat{V}(w, s_1) + \hat{V}(w, s_3)}{2}$.
- In general, $\hat{V}$ cannot be made equal to $V^\pi$.
- Which $w$ provides the best approximation?
- A common choice is

$$w^* = \arg\min_{w \in \mathbb{R}^d} MSVE(w),$$

$$MSVE(w) \overset{\text{def}}{=} \frac{1}{2} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w, s) \}^2,$$

where $\mu^\pi : S \rightarrow [0, 1]$ is the stationary distribution of $\pi$. 
How to find \( w^\star \)?

\((\mu^\pi\)-scaling not explicitly shown.\)
How to find \( w^\star \)?

(\( \mu^\pi \)-scaling not explicitly shown.)
Geometric View

(\(\mu^\pi\)-scaling not explicitly shown.)

How to find \(w^*\)?
Reinforcement Learning

1. Generalisation and function approximation

2. Linear function approximation

3. Linear TD($\lambda$)
Gradient Descent

- Iteratively take steps in the $w$ space in the direction minimising $MSVE(w)$. 

![Diagram of Gradient Descent](image)
Gradient Descent

- Iteratively take steps in the $w$ space in the direction minimising $MSVE(w)$.

- Feasible here?
Gradient Descent

- Iteratively take steps in the $w$ space in the direction minimising $MSVE(w)$.

- Feasible here? Sort of.
Gradient Descent

- Initialise $w^0 \in \mathbb{R}^d$ arbitrarily. For $t \geq 0$ update as

$$w^{t+1} \leftarrow w^t - \alpha_{t+1} \nabla_w \left( \frac{1}{2} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \}^2 \right)$$

$$= w^t + \alpha_{t+1} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \} \nabla_w \hat{V}(w^t, s).$$

But we don't know $\mu^\pi(s), V^\pi(s)$ for all $s \in S$. We're learning, remember?

Luckily, stochastic gradient descent allows us to update as

$$w^{t+1} \leftarrow w^t + \alpha_{t+1} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \} \nabla_w \hat{V}(w^t, s).$$

since $s_t \sim \mu^\pi$ anyway (as $t \to \infty$).

But still, we don't know $V^\pi(s_t)$! What to do?
Gradient Descent

- Initialise $w^0 \in \mathbb{R}^d$ arbitrarily. For $t \geq 0$ update as

$$w^{t+1} \leftarrow w^t - \alpha_{t+1} \nabla_w \left( \frac{1}{2} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \}^2 \right)$$

$$= w^t + \alpha_{t+1} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \} \nabla_w \hat{V}(w^t, s).$$

- But we don’t know $\mu^\pi(s), V^\pi(s)$ for all $s \in S$. We’re learning, remember?
Gradient Descent

- Initialise $w^0 \in \mathbb{R}^d$ arbitrarily. For $t \geq 0$ update as

$$w^{t+1} \leftarrow w^t - \alpha_{t+1} \nabla_w \left( \frac{1}{2} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \}^2 \right)$$

$$= w^t + \alpha_{t+1} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \} \nabla_w \hat{V}(w^t, s).$$

- But we don’t know $\mu^\pi(s)$, $V^\pi(s)$ for all $s \in S$. We’re learning, remember?

- Luckily, stochastic gradient descent allows us to update as

$$w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ V^\pi(s^t) - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t)$$

since $s^t \sim \mu^\pi$ anyway (as $t \to \infty$).
Gradient Descent

- Initialise $w^0 \in \mathbb{R}^d$ arbitrarily. For $t \geq 0$ update as

$$w^{t+1} \leftarrow w^t - \alpha_{t+1} \nabla_w \left( \frac{1}{2} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \}^2 \right)$$

$$= w^t + \alpha_{t+1} \sum_{s \in S} \mu^\pi(s) \{ V^\pi(s) - \hat{V}(w^t, s) \} \nabla_w \hat{V}(w^t, s).$$

- But we don’t know $\mu^\pi(s), \ V^\pi(s)$ for all $s \in S$. We’re learning, remember?
- Luckily, stochastic gradient descent allows us to update as

$$w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ V^\pi(s^t) - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t)$$

since $s^t \sim \mu^\pi$ anyway (as $t \to \infty$).
- But still, we don’t know $V^\pi(s^t)$! What to do?
Gradient Descent

Although we cannot perform update

\[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ V^\pi(s^t) - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]

we can do

\[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ G_{t:\infty} - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]

since \( \mathbb{E}[G_{t:\infty}] = V^\pi(s^t). \)
Gradient Descent

- Although we cannot perform update
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1}\{ V^\pi(s^t) - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]
  
  we can do
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1}\{ G_{t:\infty} - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]
  
since \( \mathbb{E}[G_{t:\infty}] = V^\pi(s^t) \).

- In practice, we also do
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1}\{ G^{\lambda}_t - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]
  
  for \( \lambda < 1 \), even if \( \mathbb{E}[G^{\lambda}_t] \neq V^\pi(s^t) \) in general.
Gradient Descent

- Although we cannot perform update
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ V^\pi(s^t) - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]

  we can do

  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ G_{t:\infty} - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]

  since \( \mathbb{E}[G_{t:\infty}] = V^\pi(s^t). \)

- In practice, we also do

  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ G_{t} - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]

  for \( \lambda < 1, \) even if \( \mathbb{E}[G_{t}^\lambda] \neq V^\pi(s^t) \) in general. For example, Linear TD(0) performs the update

  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ r^t + \gamma w^t \cdot x(s^{t+1}) - w^t \cdot x(s^t) \} x(s^t). \]
Gradient Descent

- Although we cannot perform update
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ V^\pi(s^t) - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]
  we can do
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ G_{t:\infty} - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]
  since \( \mathbb{E}[G_{t:\infty}] = V^\pi(s^t) \).

- In practice, we also do
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ G^\lambda_t - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t), \]
  for \( \lambda < 1 \), even if \( \mathbb{E}[G^\lambda_t] \neq V^\pi(s^t) \) in general. For example, Linear TD(0) performs the update
  \[ w^{t+1} \leftarrow w^t + \alpha_{t+1} \{ r^t + \gamma w^t \cdot x(s^{t+1}) - w^t \cdot x(s^t) \} x(s^t). \]

- For \( \lambda < 1 \), the process is not true gradient descent. But it still converges with linear function approximation.
Linear TD($\lambda$) algorithm

- Maintains an eligibility trace $z \in \mathbb{R}^d$.
- Recall that $\hat{V}(w, s) = w \cdot x(s)$, hence $\nabla_w \hat{V}(w, s) = x(s)$.

Initialise $w \in \mathbb{R}^d$ arbitrarily.
Repeat for each episode:
- Set $z \rightarrow 0$. //Eligibility trace vector.
- Assume the agent is born in state $s$.
Repeat for each step of episode:
- Take action $a$; obtain reward $r$, next state $s'$.
- $\delta \leftarrow r + \gamma \hat{V}(w, s') - \hat{V}(w, s)$.
- $z \leftarrow \gamma \lambda z + \nabla_w \hat{V}(w, s)$.
- $w \leftarrow w + \alpha \delta z$.
- $s \leftarrow s'$.

See Sutton and Barto (2018) for variations (accumulating, replacing, and dutch traces).
Linear TD($\lambda$) algorithm

- Maintains an eligibility trace $z \in \mathbb{R}^d$.
- Recall that $\hat{V}(w, s) = w \cdot x(s)$, hence $\nabla_w \hat{V}(w, s) = x(s)$.

Initialise $w \in \mathbb{R}^d$ arbitrarily.
Repeat for each episode:
  Set $z \rightarrow 0$.//Eligibility trace vector.
  Assume the agent is born in state $s$.
  Repeat for each step of episode:
    Take action $a$; obtain reward $r$, next state $s'$.
    $\delta \leftarrow r + \gamma \hat{V}(w, s') - \hat{V}(w, s)$.
    $z \leftarrow \gamma \lambda z + \nabla_w \hat{V}(w, s)$.
    $w \leftarrow w + \alpha \delta z$.
    $s \leftarrow s'$.

See Sutton and Barto (2018) for variations (accumulating, replacing, and dutch traces).
Linear TD(\(\lambda\)) algorithm

- Maintains an eligibility trace \(z \in \mathbb{R}^d\).
- Recall that \(\hat{V}(w, s) = w \cdot x(s)\), hence \(\nabla_w \hat{V}(w, s) = x(s)\).

\[
\begin{align*}
\text{Initialise } w \in \mathbb{R}^d \text{ arbitrarily.} \\
\text{Repeat for each episode:} \\
&\quad \text{Set } z \rightarrow 0.//\text{Eligibility trace vector.} \\
&\quad \text{Assume the agent is born in state } s. \\
&\quad \text{Repeat for each step of episode:} \\
&\quad \quad \text{Take action } a; \text{ obtain reward } r, \text{ next state } s'. \\
&\quad \quad \delta \leftarrow r + \gamma \hat{V}(w, s') - \hat{V}(w, s). \\
&\quad \quad z \leftarrow \gamma \lambda z + \nabla_w \hat{V}(w, s). \\
&\quad \quad w \leftarrow w + \alpha \delta z. \\
&\quad \quad s \leftarrow s'.
\end{align*}
\]

- See Sutton and Barto (2018) for variations (accumulating, replacing, and dutch traces).
Convergence of Linear TD($\lambda$)

$$MSVE(w_\lambda^\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} MSVE(w^*).$$
Convergence of Linear TD(\(\lambda\))

\[
MSVE(w_\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} MSVE(w^*).
\]
Linear function approximation is implemented in the control by approximating
\[ Q(s, a) \approx w \cdot x(s, a). \]

Linear Sarsa(\(\lambda\)) is a very popular algorithm.
RL on Half Field Offense

- Uses Linear Sarsa(0) with tile coding.

Reinforcement Learning

1. Generalisation and function approximation

2. Linear function approximation

3. Linear TD(\(\lambda\))