Reinforcement Learning

1. Generalisation and function approximation

2. Linear function approximation

3. Linear TD(λ)
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!

Shivaram Kalyanakrishnan (2023)
Half Field Offense

- Decision-making restricted to offense player with ball.
- Based on state, choose among Dribble, Pass, Shoot.
Half Field Offense

- 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.

$x_1(s)$: 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.
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 $\hat{Q}(s, a) \approx Q^*(s, a)$?

\[
\begin{align*}
\hat{Q}(s, a_1) & = \sigma^1 \left( \sum_{x_1(s)} w \right) \\
\hat{Q}(s, a_2) & = \sigma^2 \left( \sum_{x_2(s)} w \right) \\
\hat{Q}(s, a_3) & = \sigma^3 \left( \sum_{x_3(s)} w \right) \\
\hat{Q}(s, a) & = \sigma^4 \left( \sum_{x_4(s)} w \right)
\end{align*}
\]
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 $\hat{Q}(s, a) \approx Q^*(s, a)$?

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

- $x : S \to \mathbb{R}^d$ is a $d$-dimensional feature vector, and
- $w \in \mathbb{R}^d$ is the weight/coefficients 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 \( s \in S \),

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

\( x : S \rightarrow \mathbb{R}^d \) is a \( d \)-dimensional feature vector, and \( w \in \mathbb{R}^d \) 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>$s$</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 $V^\pi$.
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

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

where

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

and $\mu^\pi : S \rightarrow [0, 1]$ is the stationary distribution of $\pi$. 
Geometric View

$(\mu^\pi\text{-scaling not explicitly shown.})$
Geometric View

$V^\pi$

$\mathbf{w}^*$

$(\mu^\pi\text{-scaling not explicitly shown.})$
Geometric View

How to find $w^*$?

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

1. Generalisation and function approximation

2. Linear function approximation

3. Linear TD(λ)
Gradient Descent

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

$$\nabla \pi^w$$
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

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

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

  $$w_{t+1} \leftarrow w_t + \alpha_{t+1} \{ G_t^\lambda - \hat{V}(w^t, s^t) \} \nabla_w \hat{V}(w^t, s^t),$$

  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 \(\nabla_w \hat{V}(w, s) = x(s)\).
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 \(s'\).
    - \(\delta \leftarrow r + \gamma \hat{V}(w, s') - \hat{V}(w, s)\).
    - \(z \leftarrow \gamma \lambda z + \nabla_w \hat{V}(w, s)\).
    - \(w \leftarrow w + \alpha \delta z\).
    - \(s \leftarrow s'\).
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:
   
   \[
   \text{Set } z \rightarrow 0. // \text{Eligibility trace vector.}
   \]
   
   Assume the agent is born in state \(s\).
   Repeat for each step of episode:
   
   \[
   \begin{align*}
   \delta & \leftarrow r + \gamma \hat{V}(w, s') - \hat{V}(w, s) \\
   z & \leftarrow \gamma \lambda z + \nabla_w \hat{V}(w, s) \\
   w & \leftarrow w + \alpha \delta z \\
   s & \leftarrow s'.
   \end{align*}
   \]

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

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

$$MSVE(w_\lambda^\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} MSVE(w^*).$$
Control with Linear Function Approximation

- 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(λ)