CS545: Representing the Value Function as a Table

Chuck Anderson

Department of Computer Science
Colorado State University

Fall, 2009
Outline

State-Action Value Function as a Table
Table as a Function

Tabular Q for the Maze Problem
Representing the Q Table
Updating the Q Table
Agent-World Interaction Loop

Tabular Q for Tic-Tac-Toe
Difference from Maze
Representing the Q Table
Agent-World Interaction Loop
Recall that the state-action value function is a function of both state and action and its value is a prediction of the expected sum of future reinforcements.
State-Action Value Function

- Recall that the state-action value function is a function of both state and action and its value is a prediction of the expected sum of future reinforcements.
State-Action Value Function

- Recall that the state-action value function is a function of both state and action and its value is a prediction of the expected sum of future reinforcements.


- We can select our current belief of what the optimal action, $a_t^o$, is in state $s_t$ by

$$a_t^o = \arg\max_a Q(s_t, a)$$
Outline

State-Action Value Function as a Table
  Table as a Function

Tabular Q for the Maze Problem
  Representing the Q Table
  Updating the Q Table
  Agent-World Interaction Loop

Tabular Q for Tic-Tac-Toe
  Difference from Maze
  Representing the Q Table
  Agent-World Interaction Loop
For the Maze

- For the maze world we discussed last time,

\[ a_t^o = \arg\max_a Q(s_t, a) \]

looks like

\[
\begin{align*}
\arg\max_{\{\uparrow \rightarrow \downarrow \leftarrow\}} \{ Q(\text{S}, \uparrow), \} , \\
Q(\text{S}, \rightarrow), \\
Q(\text{S}, \downarrow), \\
Q(\text{S}, \leftarrow)\}
\end{align*}
\]
For the Maze

For the maze world we discussed last time,

\[ a_t^o = \arg\max_a Q(s_t, a) \]

looks like

\[
\arg\max_{\{\uparrow \rightarrow \downarrow \leftarrow\}} \{ Q(s, \uparrow), Q(s, \rightarrow), Q(s, \downarrow), Q(s, \leftarrow) \}
\]

A bit more mathematically, let the current state be given by position in \( x \) and \( y \) coordinates and actions are integers 1 to 4. Then

\[ a_t^o = \arg\max_{a \in \{1,2,3,4\}} Q((x, y), a) \]
This

\[ a_t^o = \arg\max_{a \in \{1,2,3,4\}} Q((x, y), a) \]

is okay for math, but how do we implement the Q function?
This

\[ a_t^o = \arg\max_{a \in \{1, 2, 3, 4\}} Q((x, y), a) \]

is okay for math, but how do we implement the Q function?

For the maze problem, we know we can...
This

\[ a^*_t = \operatorname{argmax}_{a \in \{1, 2, 3, 4\}} Q((x, y), a) \]

is okay for math, but how do we implement the Q function?

For the maze problem, we know we can

- enumerate all the states (positions) the set of which is finite \((10 \cdot 10)\),
This

\[ a_t^o = \arg\max_{a \in \{1,2,3,4\}} Q((x,y), a) \]

is okay for math, but how do we implement the Q function?

For the maze problem, we know we can

- enumerate all the states (positions) the set of which is finite (10 · 10),
- enumerate all actions, the set of which is finite (4),
This

\[ a_t^o = \arg\max_{a \in \{1,2,3,4\}} Q((x,y),a) \]

is okay for math, but how do we implement the Q function?

For the maze problem, we know we can

- enumerate all the states (positions) the set of which is finite \((10 \cdot 10)\),
- enumerate all actions, the set of which is finite \((4)\),
- calculate the new state from the old state and an action, and
This

\[ a_t^o = \text{argmax}_{a \in \{1,2,3,4\}} Q((x,y), a) \]

is okay for math, but how do we implement the Q function?

For the maze problem, we know we can

- enumerate all the states (positions) the set of which is finite \((10 \cdot 10)\),
- enumerate all actions, the set of which is finite \((4)\),
- calculate the new state from the old state and an action, and
- represent in memory all state-action combinations \((10 \cdot 10 \cdot 4)\).
This

\[ a_t^o = \arg\max_{a \in \{1,2,3,4\}} Q((x,y), a) \]

is okay for math, but how do we implement the Q function?

For the maze problem, we know we can

- enumerate all the states (positions) the set of which is finite (10 \cdot 10),
- enumerate all actions, the set of which is finite (4),
- calculate the new state from the old state and an action, and
- represent in memory all state-action combinations (10 \cdot 10 \cdot 4).

So, let’s just store the Q function in table form.
Q Table for the Maze

- The Q table will need three dimensions, for $x$, $y$, and the action.

![Diagram of a 3D Q table with states and actions]

How do we look up the Q values for a state?

Q values are steps to goal, so we are minimizing. Select right or down action.
Q Table for the Maze

- The Q table will need three dimensions, for $x$, $y$, and the action.

- How do we look up the Q values for a state?
Q Table for the Maze

- The Q table will need three dimensions, for $x$, $y$, and the action.

How do we look up the Q values for a state?

Q values are steps to goal, so we are minimizing. Select right or down action.
Q Table for the Maze

- The Q table will need three dimensions, for \( x \), \( y \), and the action.

  ![Diagram of a 3D Q table]

- How do we look up the Q values for a state?

  ![Diagram showing how to look up Q values]

- Q values are steps to goal, so we are minimizing. Select right or down action.
Q Table in R for the Maze

- Now, in R. How can we make a three-dimensional table of Q values, if \( x \) and \( y \) have 10 possible values and we have 4 actions?

```r
Q_table <- array(0, c(10, 10, 4))
```

How should we initialize the table? Above line initializes all values to be zero. What effect will this have as Q values for actions taken are updated to estimate steps to goal?

Actions not yet tried will have lowest (0) Q value. Forces the agent to try all actions from all states—lots of exploration.
Q Table in R for the Maze

- Now, in R. How can we make a three-dimensional table of Q values, if \( x \) and \( y \) have 10 possible values and we have 4 actions?

\[
Q \leftarrow \text{array}(0, c(10, 10, 4))
\]
Q Table in R for the Maze

- Now, in R. How can we make a three-dimensional table of Q values, if x and y have 10 possible values and we have 4 actions?

```r
Q <- array(0,c(10,10,4))
```

- How should we initialize the table? Above line initializes all values to be zero. What effect will this have as Q values for actions taken are updated to estimate steps to goal?
Q Table in R for the Maze

- Now, in R. How can we make a three-dimensional table of Q values, if x and y have 10 possible values and we have 4 actions?

\[ Q \leftarrow \text{array}(0,c(10,10,4)) \]

- How should we initialize the table? Above line initializes all values to be zero. What effect will this have as Q values for actions taken are updated to estimate steps to goal?

- Actions not yet tried will have lowest (0) Q value. Forces the agent to try all actions from all states—lots of exploration.
TD Update for Q Table

- What must we do after observing $s_t$, $a_t$, $r_{t+1}$, $s_{t+1}$, and $a_{t+1}$?
TD Update for Q Table

- What must we do after observing \( s_t, a_t, r_{t+1}, s_{t+1}, \) and \( a_{t+1} \)?

- Calculate the temporal-difference error
  \[ r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \]
  and use it to update the Q value stored for \( s_t \) and \( a_t \):

\[
Q(s_t, a_t) = Q(s_t, a_t) + \rho (r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))
\]
TD Update for Q Table

- What must we do after observing \( s_t, a_t, r_{t+1}, s_{t+1}, \) and \( a_{t+1} \)?
- Calculate the temporal-difference error
  \[ r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \]
  and use it to update the Q value stored for \( s_t \) and \( a_t \):
  \[
  Q(s_t, a_t) = Q(s_t, a_t) + \rho (r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))
  \]
- And, in R? Assume position, or state, \((2,3)\) is implemented as \texttt{state} \( \gets \texttt{c}(2,3) \).
TD Update for Q Table

- What must we do after observing $s_t$, $a_t$, $r_{t+1}$, $s_{t+1}$, and $a_{t+1}$?

- Calculate the temporal-difference error
  
  $r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$

  and use it to update the Q value stored for $s_t$ and $a_t$:

  \[
  Q(s_t, a_t) = Q(s_t, a_t) + \rho(r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))
  \]

- And, in R? Assume position, or state, $(2, 3)$ is implemented as `state <- c(2,3)`.

  ```
  r <- 1
  Qold <- Q[stateOld[1], stateOld[2], actionOld]
  Qnew <- Q[state[1], state[2], action]
  TDError <- r + Qnew - Qold
  Q[stateOld[1], stateOld[2], actionOld] <- Qold + rho * TDError
  ```
Final step must be handled differently. There is no $s_{t+1}$. 

The update $Q(s_t, a_t) = Q(s_t, a_t) + \rho (r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$ becomes $Q(s_t, a_t) = Q(s_t, a_t) + \rho (r_{t+1} - Q(s_t, a_t))$.

In R, add test for being at goal. Let maze be character array containing a 'G' at the goal position.

```r
Qold <- Q[stateOld[1], stateOld[2], actionOld]
Qnew <- Q[state[1], state[2], action]
if (maze[state[1], state[2]] == 'G')
  TDerror <- r - Qold
else
  TDerror <- r + Qnew - Qold
Q[stateOld[1], stateOld[2], actionOld] <- Qold + rho * TDerror
```
Final step must be handled differently. There is no $s_{t+1}$.

The update

$$Q(s_t, a_t) = Q(s_t, a_t) + \rho (r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

becomes

$$Q(s_t, a_t) = Q(s_t, a_t) + \rho (r_{t+1} - Q(s_t, a_t))$$
Final step must be handled differently. There is no $s_{t+1}$.

The update

$$Q(s_t, a_t) = Q(s_t, a_t) + \rho(r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

becomes

$$Q(s_t, a_t) = Q(s_t, a_t) + \rho(r_{t+1} - Q(s_t, a_t))$$

In R, add test for being at goal. Let maze be character array containing a G at the goal position.
Final step must be handled differently. There is no $s_{t+1}$.

The update

$$Q(s_t, a_t) = Q(s_t, a_t) + \rho(r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

becomes

$$Q(s_t, a_t) = Q(s_t, a_t) + \rho(r_{t+1} - Q(s_t, a_t))$$

In R, add test for being at goal. Let maze be character array containing a G at the goal position.

```r
r <- 1
Qold <- Q[stateOld[1], stateOld[2], actionOld]
Qnew <- Q[state[1], state[2], action]

if (maze[state[1], state[2]] == 'G')
  TDerror <- r - Qold
else
  TDerror <- r + Qnew - Qold

Q[stateOld[1], stateOld[2], actionOld] <- Qold + rho * TDerror
```
Interaction Loop

For our agent to interact with its world, we must implement

Initialize Q.
Choose random, non-goal, state.
Repeat:

If at goal,
    Update Qold with TD error \((1 - Qold)\)
    Pick new random state

Otherwise (not at goal),
    Select next action.
    If not first step, update Qold with TD error \((1 + Q_{\text{new}} - Qold)\)
    Shift current state and action to old ones.
    Apply action to get new state.
Interaction Loop in R

```r
goal <- c(5,2)
Q <- array(0,c(10,10,4))
while( all (goal == (state <- round(c(runif(1,1,10), runif (1,1,10))))))
  nInteractions <- 0
step <- 0
while ( nInteractions < 500000 ) {
  Qold <- Q[stateOld[1],stateOld [2], actionOld]
    Q[stateOld [1], stateOld [2], actionOld] <-
    Qold + rho * (1 − Qold)
    while( all (goal == (state <- round(c(runif(1,1,10), runif (1,1,10))))))
  } else {
    action <- which.min(Q[state[1],state [2],])
    if (step > 0) {
      Qnew <- Q[state[1],state [2], action]
      Q[stateOld [1], stateOld [2], actionOld] <-
      Qold + rho * (1 + Qnew − Qold)
    }
    stateOld <- state
    actionOld <- action
  }
  state <- state + actions[action ,]
  nInteractions <- nInteractions + 1
}
```
Outline

State-Action Value Function as a Table
  Table as a Function

Tabular Q for the Maze Problem
  Representing the Q Table
  Updating the Q Table
  Agent-World Interaction Loop

Tabular Q for Tic-Tac-Toe
  Difference from Maze
  Representing the Q Table
  Agent-World Interaction Loop
Tic-Tac-Toe

How does Tic-Tac-Toe differ from the maze problem?
Tic-Tac-Toe

- How does Tic-Tac-Toe differ from the maze problem?
- Different state and action sets.
Tic-Tac-Toe

- How does Tic-Tac-Toe differ from the maze problem?
- Different state and action sets.
- Two players rather than one.
Tic-Tac-Toe

- How does Tic-Tac-Toe differ from the maze problem?
- Different state and action sets.
- Two players rather than one.
- Reinforcement is 0 until end of game, when it is 1 for win, 0 for draw, or -1 for loss.
How does Tic-Tac-Toe differ from the maze problem?

- Different state and action sets.
- Two players rather than one.
- Reinforcement is 0 until end of game, when it is 1 for win, 0 for draw, or -1 for loss.
- Maximizing sum of reinforcement rather than minimizing.
How does Tic-Tac-Toe differ from the maze problem?

- Different state and action sets.
- Two players rather than one.
- Reinforcement is 0 until end of game, when it is 1 for win, 0 for draw, or -1 for loss.
- Maximizing sum of reinforcement rather than minimizing.
- Anything else?
The Q Table

- State is board configuration. There are $3^9$ of them, though not all are reachable. Is this too big?
The Q Table

- State is board configuration. There are $3^9$ of them, though not all are reachable. Is this too big?
- A bit less than 20,000. Not bad. Is this the full size of the Q table?
The Q Table

- State is board configuration. There are $3^9$ of them, though not all are reachable. Is this too big?
- A bit less than 20,000. Not bad. Is this the full size of the Q table?
- Must add the action dimension. There are at most 9 actions, one for each cell on the board.
The Q Table

- State is board configuration. There are $3^9$ of them, though not all are reachable. Is this too big?
- A bit less than 20,000. Not bad. Is this the full size of the Q table?
- Must add the action dimension. There are at most 9 actions, one for each cell on the board.
- So Q table will be contain about $20,000 \cdot 9$ values or about 200,000. No worries.
The Q Table

- State is board configuration. There are $3^9$ of them, though not all are reachable. Is this too big?
- A bit less than 20,000. Not bad. Is this the full size of the Q table?
- Must add the action dimension. There are at most 9 actions, one for each cell on the board.
- So Q table will be contain about $20,000 \cdot 9$ values or about 200,000. No worries.
- Q table is two-dimensional, with rows indexed by board configuration and columns by action.
The Q Table

- State is board configuration. There are $3^9$ of them, though not all are reachable. Is this too big?
- A bit less than 20,000. Not bad. Is this the full size of the Q table?
- Must add the action dimension. There are at most 9 actions, one for each cell on the board.
- So Q table will contain about $20,000 \cdot 9$ values or about 200,000. No worries.
- Q table is two-dimensional, with rows indexed by board configuration and columns by action.
- Need a way to represent a board, and a way to map from a board to a Q table row.
The Q Table

- Represent this board as \( c(0,0,1, 0,2,0, 0,0,0) \)

\[
\begin{array}{|c|c|c|}
\hline
\text{X} & \text{O} & \\
\hline
\text{X} & \text{O} & \\
\hline
\end{array}
\]

Now we can convert this to a row index to the Q table by treating these 9 digits as a base 3 number. Can do this “encode” step and its inverse with the \( \text{encode} \) function:

\[
\text{encode}(\text{board}) = \sum (\text{board} \times 3^{(0:8)}) + 1
\]

And the \( \text{decode} \) function:

\[
\text{decode}(\text{index}) = \text{rep}(0,9) \quad \text{for} \quad (i \in 2:9) \\
\text{board}[i] = \text{index} \mod 3 \\
\text{index} = \text{floor}(\text{index} / 3)
\]

## test it
\[
\text{decode(encode(c(1,0,0, 2,0,0, 0,1,0)))}
\]

## returns 1 0 0 2 0 0 0 1 0
The Q Table

- Represent this board as `c(0,0,1, 0,2,0, 0,0,0)`

```
  X

  O

  X
```

- Now we can convert this to a row index to the Q table by treating these 9 digits as a base 3 number. Can do this “encode” step and its inverse with

```r
encode <- function (board) {
  sum(board * 3^(0:8)) + 1
}
```

```r
decode <- function(index) {
  board <- rep(0,9)
  board[1] <- index %% 3
  for (i in 2:9) {
    index <- floor(index / 3)
    board[i] <- index %% 3
  }
  board
}
```

```r
## test it
decode(encode(c(1,0,0, 2,0,0, 0,1,0)))
## returns 1 0 0 2 0 0 0 1 0
```
Interaction Loop

For our agent to interact with its world, we must implement

Initialize Q.
Set initial state, as empty board.
Repeat:

  Agent chooses next X move.
  If X wins, set Qnew to 1.
  Else, if board is full, set Qnew to 0.
  Else, let O take move.
  If O won, update Qnew by (−1 − Qnew)
  For all cases, update Qold by Qnew − Qold
  Shift current state and action to old ones.