Lecture 11: Reinforcement Learning
Cognitive Systems - Machine Learning

Part III: Learning Programs and Strategies

Q Learning, Dynamic Programming

last change January 14, 2011
Motivation

**addressed problem**: How can an autonomous agent that senses and acts in its environment learn to choose optimal actions to achieve its goals?

- consider building a learning robot (i.e., agent)
  - the agent has a set of *sensors* to observe the *state* of its environment and
  - a set of *actions* it can perform to alter its state
  - the task is to learn a control strategy, or *policy*, for choosing actions that achieve its goals

**assumption**: goals can be defined by a *reward function* that assigns a numerical value to each distinct action the agent may perform from each distinct state
Motivation

- **considered settings:**
  - deterministic or nondeterministic outcomes
  - prior background knowledge available or not

- **similarity to function approximation:**
  - approximating the function $\pi : S \rightarrow A$
    where $S$ is the set of states and $A$ the set of actions

- **differences to function approximation:**
  - Delayed reward: training information is not available in the form $<s, \pi(s)>$. Instead the trainer provides only a sequence of immediate reward values.
  - Temporal credit assignment: determining which actions in the sequence are to be credited with producing the eventual reward
differences to function approximation (cont.):

- exploration: distribution of training examples is influenced by the chosen action sequence
  - which is the most effective exploration strategy?
  - trade-off between exploration of unknown states and exploitation of already known states
- partially observable states: sensors only provide partial information of the current state (e.g. forward-pointing camera, dirty lenses)
- life-long learning: function approximation often is an isolated task, while robot learning requires to learn several related tasks within the same environment
The Learning Task

- based on Markov Decision Processes (MDP)
  - the agent can perceive a set $S$ of distinct states of its environment and has a set $A$ of actions that it can perform.
  - at each discrete time step $t$, the agent senses the current state $s_t$, chooses a current action $a_t$ and performs it.
  - the environment responds by returning a reward $r_t = r(s_t, a_t)$ and by producing the successor state $s_{t+1} = \delta(s_t, a_t)$.
  - the functions $r$ and $\delta$ are part of the environment and not necessarily known to the agent.
  - in an MDP, the functions $r(s_t, a_t)$ and $\delta(s_t, a_t)$ depend only on the current state and action.
The Learning Task

- the task is to learn a policy $\pi: S \rightarrow A$
- one approach to specify which policy $\pi$ the agent should learn is to require the policy that produces the greatest possible cumulative reward over time (discounted cumulative reward)

$$V^\pi(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $V^\pi(s_t)$ is the cumulative value achieved by following an arbitrary policy $\pi$ from an arbitrary initial state $s_t$

$r_{t+i}$ is generated by repeatedly using the policy $\pi$ and $\gamma$ ($0 \leq \gamma < 1$) is a constant that determines the relative value of delayed versus immediate rewards
hence, the agent’s learning task can be formulated as

\[ \pi^* \equiv \arg\max_{\pi} V^\pi(s), (\forall s) \]
Illustrative Example

- The left diagram depicts a simple grid-world environment:
  - Squares $\approx$ states, locations
  - Arrows $\approx$ possible transitions (with annotated $r(s, a)$)
  - $G \approx$ goal state (absorbing state)

- $\gamma = 0.9$

- Once states, actions, and rewards are defined and $\gamma$ is chosen, the optimal policy $\pi^*$ with its value function $V^*(s)$ can be determined.
Illustrative Example

- The right diagram shows the values of $V^*$ for each state.

- E.g., consider the bottom-right state:
  - $V^* = 100$, because $\pi^*$ selects the “move up” action that receives a reward of 100.
  - Thereafter, the agent will stay in $G$ and receive no further awards.
  - $V^* = 100 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \ldots = 100$.

- E.g., consider the bottom-center state:
  - $V^* = 90$, because $\pi^*$ selects the “move right” and “move up” actions.
  - $V^* = 0 + \gamma \cdot 100 + \gamma^2 \cdot 0 + \ldots = 90$.

- Recall that $V^*$ is defined to be the sum of discounted future awards over infinite future.
Q Learning

- it is easier to learn a numerical evaluation function than implement
  the optimal policy in terms of the evaluation function

- **question:** What evaluation function should the agent attempt to
  learn?

- one obvious choice is $V^*$

- the agent should prefer $s_1$ to $s_2$ whenever $V^*(s_1) > V^*(s_2)$

- **problem:** the agent has to chose among actions, not among
  states

$$
\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]
$$

the optimal action in state $s$ is the action $a$ that maximizes the sum
of the immediate reward $r(s, a)$ plus the value of $V^*$ of the
immediate successor, discounted by $\gamma$
Q Learning

- thus, the agent can acquire the optimal policy by learning $V^*$, provided it has perfect knowledge of the immediate reward function $r$ and the state transition function $\delta$
- in many problems, it is impossible to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state
- the $Q$ function provides a solution to this problem
  - $Q(s, a)$ indicates the maximum discounted reward that can be achieved starting from $s$ and applying action $a$ first

\[
Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))
\]

\[
\Rightarrow \pi^*(s) = \arg\max_a Q(s, a)
\]
Q Learning

- hence, learning the Q function corresponds to learning the optimal policy $\pi^*$
- if the agent learns Q instead of $V^*$, it will be able to select optimal actions even when it has no knowledge of $r$ and $\delta$
- it only needs to consider each available action $a$ in its current state $s$ and chose the action that maximizes $Q(s, a)$
- the value of $Q(s, a)$ for the current state and action summarizes in one value all information needed to determine the discounted cumulative reward that will be gained in the future if $a$ is selected in $s$
Q Learning

- the right diagram shows the corresponding $Q$ values
- the $Q$ value for each state-action transition equals the $r$ value for this transition plus the $V^*$ value discounted by $\gamma$
Q Learning Algorithm

- **key idea**: iterative approximation
- relationship between $Q$ and $V^*$

\[ V^*(s) = \max_{a'} Q(s, a') \]

\[ Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \]

- this recursive definition is the basis for algorithms that use iterative approximation
- the learner’s estimate $\hat{Q}(s, a)$ is represented by a large table with a separate entry for each state-action pair
Q Learning Algorithm

Algorithm

For each \( s, a \) initialize the table entry \( \hat{Q}(s, a) \) to zero
Observe the current state \( s \)
Do forever:

- Select an action \( a \) and execute it
- Receive immediate reward \( r \)
- Observe new state \( s' \)
- Update the table entry for \( \hat{Q}(s, a) \) as follows
  \[
  \hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')
  \]
- \( s \leftarrow s' \)

⇒ Using this algorithm the agent’s estimate \( \hat{Q} \) converges to the actual \( Q \), provided the system can be modeled as a deterministic Markov decision process, \( r \) is bounded, and actions are chosen so that every state-action pair is visited infinitely often.
Illustrative Example

\[ \hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \cdot \max_{a'} \hat{Q}(s_2, a') \]

\[ \leftarrow 0 + 0.9 \cdot \max \{63, 81, 100\} \]

\[ \leftarrow 90 \]

- the old values are read from our \( \hat{Q} \)-table, which are about to be updated in each step
- each time the agent moves, Q Learning propagates \( \hat{Q} \) estimates \textit{backwards} from the new state to the old and updates the corresponding value in the table
Illustrative Example

Table before the move

<table>
<thead>
<tr>
<th>(s, a)</th>
<th>( \hat{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, ( \rightarrow )</td>
<td>72</td>
</tr>
<tr>
<td>2, ( \leftarrow )</td>
<td>63</td>
</tr>
<tr>
<td>2, ( \rightarrow )</td>
<td>100</td>
</tr>
<tr>
<td>2, ( \downarrow )</td>
<td>81</td>
</tr>
</tbody>
</table>

Table after the move

<table>
<thead>
<tr>
<th>(s, a)</th>
<th>( \hat{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, ( \rightarrow )</td>
<td>90</td>
</tr>
<tr>
<td>2, ( \leftarrow )</td>
<td>63</td>
</tr>
<tr>
<td>2, ( \rightarrow )</td>
<td>100</td>
</tr>
<tr>
<td>2, ( \downarrow )</td>
<td>81</td>
</tr>
</tbody>
</table>
Experimentation Strategies

- The algorithm does not specify how actions are chosen by the agent.
- **Obvious strategy**: select action $a$ that maximizes $\hat{Q}(s, a)$
  - Risk of overcommitting to actions with high $\hat{Q}$ values during earlier trainings
  - Exploration of yet unknown actions is neglected
- **Alternative**: probabilistic selection

$$P(a_i|s) = \frac{k\hat{Q}(s,a_i)}{\sum_j k\hat{Q}(s,a_j)}$$

$k > 0$ indicates how strongly the selection favors actions with high $\hat{Q}$ values

- $k$ large $\Rightarrow$ exploitation strategy
- $k$ small $\Rightarrow$ exploration strategy
so far, the target function is represented as an explicit lookup table.

The algorithm performs a kind of rote learning and makes no attempt to estimate the $Q$ value for yet unseen state-action pairs.

⇒ unrealistic assumption in large or infinite spaces or when execution costs are very high.

Incorporation of function approximation algorithms such as BACKPROPAGATION:

- table is replaced by a neural network using each $\hat{Q}(s, a)$ update as training example ($s$ and $a$ are inputs, $\hat{Q}$ the output).
- a neural network for each action $a$. 

Relationship to Dynamic Programming

- Q Learning is closely related to dynamic programming approaches that solve Markov Decision Processes
  - **dynamic programming**
    - assumption that $\delta(s, a)$ and $r(s, a)$ are known
    - focus on how to compute the optimal policy
    - mental model can be explored (no direct interaction with environment)
      ⇒ *offline system*
  - Q Learning
    - assumption that $\delta(s, a)$ and $r(s, a)$ are not known
    - direct interaction inevitable
      ⇒ *online system*
Relationship to Dynamic Programming

The relationship is apparent by considering the Bellman’s equation, which forms the foundation for many dynamic programming approaches solving Markov Decision Processes:

$$(\forall s \in S) V^*(s) = E[r(s, \pi(s)) + \gamma V^*(\delta(s, \pi(s)))]$$
Advanced Topics

- different updating sequences
- proof of convergence
- nondeterministic rewards and actions
- temporal difference learning
Learning Terminology

Q Learning

<table>
<thead>
<tr>
<th>Supervised Learning</th>
<th>unsupervised learning</th>
</tr>
</thead>
</table>

Approaches:

<table>
<thead>
<tr>
<th>Concept / Classification</th>
<th>Policy Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbolic</td>
<td>statistical / neuronal network</td>
</tr>
<tr>
<td>inductive</td>
<td>analytical</td>
</tr>
</tbody>
</table>

Learning Strategy:

⇒ learning from experience