ECE-517: Reinforcement Learning in Artificial Intelligence

Lecture 2: Evaluative Feedback (Exploration vs. Exploitation)

August 25, 2015

Dr. Itamar Arel

College of Engineering
Electrical Engineering & Computer Science Department
The University of Tennessee
Fall 2015
Outline

- Recap
- What is evaluative feedback
- N-arm Bandit Problem (test case)
  - Action-Value Methods
  - Softmax Action Selection
  - Incremental Implementation
  - Tracking nonstationary rewards
  - Optimistic Initial Values
  - Reinforcement Comparison
  - Pursuit methods
  - Associative search
Recap

RL revolves around learning from experience by interacting with the environment
- Unsupervised learning discipline
- Trial-and-error based
- Delayed reward - main concept (value functions, etc.)

Policy maps from situations to actions

Exploitation vs. Exploration is key challenge

We looked at the Tic-Tac-Toe example where:

\[ V(s) \leftarrow V(s) + \alpha[V(s') - V(s)] \]
What is Evaluative Feedback?

- RL uses training information that evaluates the actions taken rather than instructs by giving correct actions
  - Necessitates trail-by-error search for good behavior
  - Creates need for active exploration

- Pure evaluative feedback indicates how good the action taken is, but not whether it is the best or the worst action possible

- Pure instructive feedback, on the other hand, indicates the correct action to take, independent of the action actually taken
  - Corresponds to supervised learning
  - e.g. artificial neural networks
n-Armed Bandit Problem

Let’s look at a simple version of the \textit{n}-\textit{armed bandit problem}

- First step in understanding the full RL problem

Here is the problem description:

- An agent is repeatedly faced with making one out of \( n \) actions
- After each step a reward value is provided, drawn from a stationary probability distribution that depends on the action selected
- The agent’s objective is to maximize the expected total reward over time
- Each action selection is called a \textit{play} or \textit{iteration}

Extension of the classic slot machine (“one-armed bandit”)
n-Armed Bandit Problem (cont.)

- Each action has a value – an expected or mean reward given that the action is selected
  - If the agent knew the value of each function – the problem would be trivial
- The agent maintains estimates of the values, and chooses the highest
  - Greedy algorithm
  - Directly associated with (policy) exploitation
- If agent chooses non-greedily – we say it explores
  - Under uncertainty the agent must explore
  - A balance must be found between exploration & exploitation
- Initial condition: all levers assume to yield reward = 0
- We’ll see several simple balancing methods and show that they work much better than methods that always exploit
We’ll look at simple methods for estimating the values of actions.

Let $Q^*(a)$ denote the true (actual) value of $a$, and $Q_t(a)$ its estimate at time $t$.

- The true value equals the mean reward for that action.

Let’s assume that by iteration (play) $t$, action $a$ has been taken $k_a$ times – hence we may use the sample-average...

$$Q_t(a) = \frac{1}{k_a} \sum_{i=1}^{k_a} r_i = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$$

The greedy policy selects the highest sample-average, i.e.

$$a^* = \arg \max_a Q_t(a) \quad \rightarrow \quad Q_t(a^*) = \max_a Q_t(a)$$
Action-Value Methods (cont.)

- A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability $\varepsilon$, instead select an action at random.
  - This is called an $\varepsilon$-greedy method.

We simulate the 10-arm bandit problem, where ...

- $r_a \sim N(Q^*(a),1)$ (noisy readings of rewards)
- $Q^*(a) \sim N(0,1)$ (actual, true mean reward for action $a$)

We further assume that there are 2000 machines (tasks), each with 10 levers.

The rewards distributions are drawn independently for each machine.

Each iteration, choose a lever on each machine and calculate the average reward from all 2000 machines.
Action-Value Methods (cont.)

\[ \epsilon = 0.1 \]
\[ \epsilon = 0.01 \]

Average reward

\[ \% \text{ Optimal action} \]
\[ \epsilon = 0.1 \]
\[ \epsilon = 0.01 \]
\[ \epsilon = 0 \text{ (greedy)} \]
Side note: the optimal average reward

\[ E\left\{ \max \left( z_1, z_2, \ldots, z_n \right) \right\} \]

\[ z_i \sim N(0,1) \]
The advantage of $\varepsilon$-greedy methods depends on the task

- If the rewards have high variance $\Rightarrow$ $\varepsilon$-greedy would have stronger advantage
- If the rewards have zero variance $\Rightarrow$ greedy algorithm would have sufficed

If the problem was non-stationary (true rewards values changed slowly over time)

- $\varepsilon$-greedy would have been a must
- Q: Perhaps some better methods exist?
So far we assumed that while exploring (using $\epsilon$-greedy) we chose equally among the alternatives.

This means we could have chosen really bad, as opposed (for example) to choosing the next-best action.

The obvious solution is to **rank the alternatives** ...

- Generate a probability density/mass function to estimate the rewards from each action.
- All actions are ranked/weighted.
- Typically use Boltzmann distribution, i.e. choose action $a$ on iteration $t$ with probability

$$ Pr\{\text{action } = a\} = \pi_a(t) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}} $$
Softmax Action Selection (cont.)

- **$\tau = 1$**
- **$\tau = 4$**
- **$\tau = 20$**

Action index

Average value

- $\tau = 1$
- $\tau = 4$
- $\tau = 20$
Incremental Implementation

- Sample-average methods require linearly-increasing memory (storage of reward history)
- We need a more memory-efficient method ...

\[
Q_{k+1} = \frac{1}{k+1} \sum_{i=1}^{k+1} r_i \\
= \frac{1}{k+1} \left( r_{k+1} + \sum_{i=1}^{k} r_i \right) \\
= \frac{1}{k+1} \left( r_{k+1} + Q_k + Q_k - Q_k \right) \\
= \frac{1}{k+1} \left( r_{k+1} + Q_k (k+1) - Q_k \right) \\
= Q_k + \frac{1}{k+1} \left[ r_{k+1} - Q_k \right]
\]
The previous result is consistent with a recurring theme in RL which is

$$\text{New\_Estimate} \leftarrow \text{Old\_Estimate} + \text{StepSize} [\text{Target} - \text{Old\_Estimate}]$$

The StepSize may be fixed or adaptive (in accordance with the specific application)
Tracking a Nonstationary Problem

- So far we have considered stationary problems
- In reality, many problems are effectively nonstationary
- A popular approach is to weigh recent rewards more heavily than older ones
- One such technique is called fixed step size

\[
Q_k = Q_{k-1} + \alpha[r_k - Q_{k-1}]
\]

\[
= \alpha r_k + (1 - \alpha)Q_{k-1} = \alpha r_k + (1 - \alpha)\alpha r_{k-1} + (1 - \alpha)^2 Q_{k-2}
\]

\[
= \alpha r_k + (1 - \alpha)\alpha r_{k-1} + (1 - \alpha)^2 r_{k-2} + ... + (1 - \alpha)^{k-1} \alpha r_1 + (1 - \alpha)^k Q_0
\]

\[
= (1 - \alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1 - \alpha)^{k-i} r_i
\]

- This is a weighted average that exponentially decreases
Optimistic Initial Values

- All methods discussed so far depended, to some extent, on the initial action-value estimates, $Q_0(a)$
- For sample-average methods - this bias disappears when all actions have been selected at least once
- For fixed step-size methods, the bias disappears with time (geometrically decreasing)
- In the 10-arm bandit example with $\alpha = 0.1$ ...
  - If we were to set all initial reward guesses to +5 (instead of zero)
  - Exploration is guaranteed, since true values are $\sim N(0,1)$
Reinforcement Comparison

An intuitive element in RL is that …
- higher rewards → made more likely to occur
- lower rewards → made less likely to occur

How is the learner to know what constitutes a high or low reward?
- To make a judgment, one must compare the reward to a reference reward - $\bar{r}_t$
  - Natural choice - average of previously received rewards
  - These methods are called reinforcement comparison methods

The agent maintains an action preference value, $p_t(a)$, for each action $a$

The preference might be used to select an action according to a softmax relationship

$$\pi_t(a) = \frac{e^{p_t(a)}}{\sum_{b=1}^{n} e^{p_t(b)}}$$
Reinforcement Comparison (cont.)

- The reinforcement comparison idea is used in updating the action preferences

\[ p_{t+1}(a_t) = p_t(a_t) + \beta [r_t - \overline{r}] \]

- High reward increases the probability of an action to be selected, and visa versa

- Following the action preference update, the agent updates the reference reward

\[ r_{t+1} = r_t + \alpha [r_t - \overline{r}] \]

→ allows us to differentiate between rates for \( r_t \) and \( p_t \)
Pursuit Methods

Another class of effective learning methods are *pursuit* methods.

They maintain both *action-value estimates* and *action preferences*.

The preferences continually “pursue” the greedy actions.

Letting $a_{t+1}^*$ denote the greedy action, the update rules are:

$$
\pi_{t+1}(a_{t+1}^*) = \pi_t(a_{t+1}^*) + \beta \left[ 1 - \pi_t(a_{t+1}^*) \right] \\
\pi_{t+1}(a_{t+1}) = \pi_t(a_{t+1}) + \beta \left[ 0 - \pi_t(a_{t+1}) \right]
$$

for $a = a_{t+1}^*$

for $a \neq a_{t+1}^*$

The action value estimates, $Q_{t+1}(a)$, are updated using one of the ways described (e.g. sample averages of observed rewards).
Pursuit Methods (cont.)

% Optimal action

$\epsilon$-greedy
$\epsilon = 0.1, \ \alpha = 1/k$

reinforcement comparison

pursuit

Plays

0 200 400 600 800 1000

0% 20% 40% 60% 80% 100%
Associative Search

So far we've considered *nonassociative* tasks in which there was no association of *actions* with *states*

- Find the single best action when task is stationary, or
- Track the best action as it changes over time

However, in RL the goal is to learn a *policy* (i.e. state to action mappings)

A natural extension of the *n*-arm bandit:

- Assume you have *K* machines, but only one is played at a time
- The agent maps the state (i.e. machine played) to the action

This would be called an *associative search* task

- It is like the full RL problem in that it involves a policy
- However, it lacks the long-term reward prospect of full RL
Summary

- We’ve looked at various action-selection schemes
- Balancing exploration vs. exploitation
  - $\epsilon$-greedy
  - Softmax techniques
- There is no “single-best” solution for all problems
- We’ll see more of this issue later …