<CAPS>

Session 1 – Foundations Bootcamp

Leo Klenner, Henry Fung, Cory Combs

Outline

1. Admin
2. Introduction to CAPS
3. Case Study on Learning in Human Teams
4. Reinforcement Learning Basics

Admin

- Two sessions per week, 6:00-7:30 pm
  - Wednesdays in Nitze 507
  - Fridays in BOB 736
- Skills course policy
  - Attend all session to receive transcript certification
- Course website: github.com/capsseminar

Introduction to CAPS

- What skills do we need to guide the implementation of artificial intelligence technologies into the international arena?
  - Explore a relevant skill set through the fictional **Forward Deployed Policy Engineer**

Training as a Forward Deployed Policy Engineer (FDPE)

- Differentiate levels of autonomous decision making based on latent performance factors
- Navigate core algorithmic learning architectures, such as reinforcement learning
- Transform the specification of a learning algorithm into decision-relevant intelligence
- Infer a system‘s underlying specifications from its observed behavior
- Find misspecifications in a software architecture based on fuzzy end-user feedback
- Identify performance-relevant links in complex human-machine ecosystems
- Communicate technical topics to senior policy makers and policies to engineers

Case Study: Learning in a COIN Team

- Army Captains Jason Emory and Mark Nutsch
  - Deployed as one the first Special Forces detachments to Afghanistan in 2001
  - Our goal is to understand how they relate their training to their deployment
- Understand general conceptual trade-offs that also apply to machine learning
- Coverage of training, modes of learning, relevance of training data set

Markov Decision Processes

- **Agent** senses its current **state** $s_{t,}$ chooses an **action** a_{t} from a set of all possible actions A, and performs it.
- In response, **environment** “gives” the agent two things:
  - Immediate **reward** r_{t}
  - Subsequent state $s_{\mathrm{t+1}}$
- The task of the agent is to learn a “**policy**” that dictates the choice of its action a_{t} based on its current observed state $s_{\mathrm{t:}}$

$$\pi(s_t) = \forall s_t \in S$$

The Cumulative Discounted Reward

- Overtime the agent gets a sequence of rewards for each action that it takes:

$$V^{\pi}(s_t) = \ r_t + \gamma r_{t+1} + y^2 r_{t+2} + \cdots$$

- The agent starts in s_t, and repeatedly uses policy π to select its actions.
- The agent gets a cumulative reward V from starting in s_t and following π thereafter.

- What is the optimal policy the robot should learn?

The Learning Task

> We require the agent to learn a policy π that maximizes its cumulative reward for all s. we call such a policy $\pi*$ the “optimal policy”.

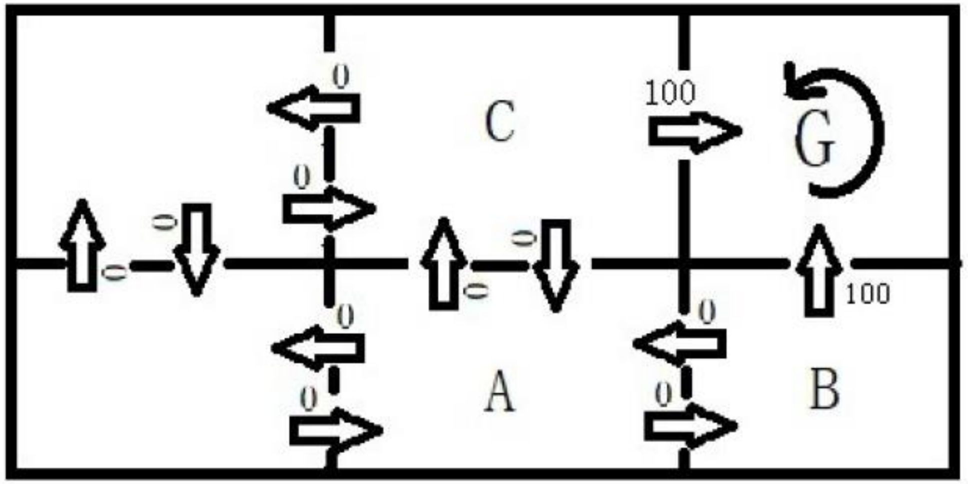
$$\pi^* = \underset{\pi}{argmax} V^{\pi}(s)$$

> In other words, the agent must follow a sequence of actions, dictated by policy $\pi*$, that allows it to receive the maximum possible cumulative reward V*(s).

Components of the Six-Grid World

- G is the goal state
- The discount factor $y = 0.9$
- Arrows represent the possible actions

  that the agent can perform in each state
- The numbers on the arrows are the immediate rewards
- **The agent‘s learning task**: starting from an arbitrary state, move to G on the shortest path

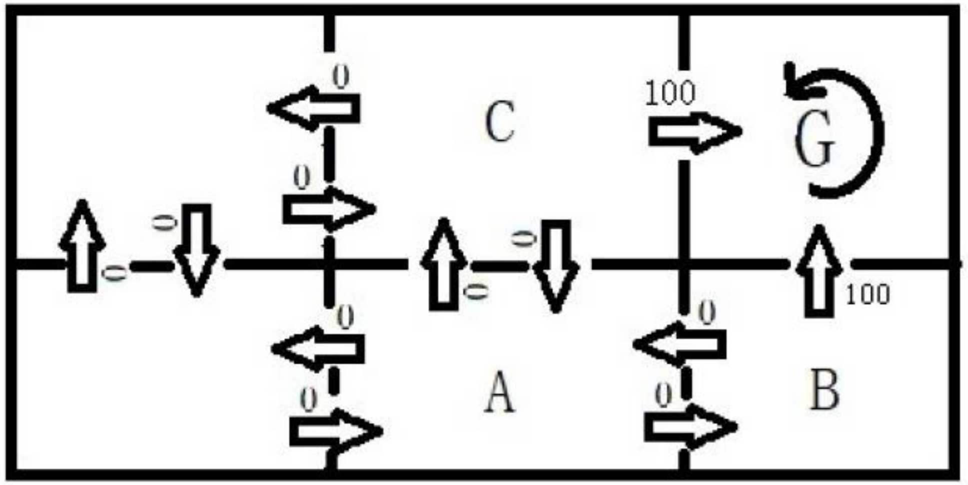


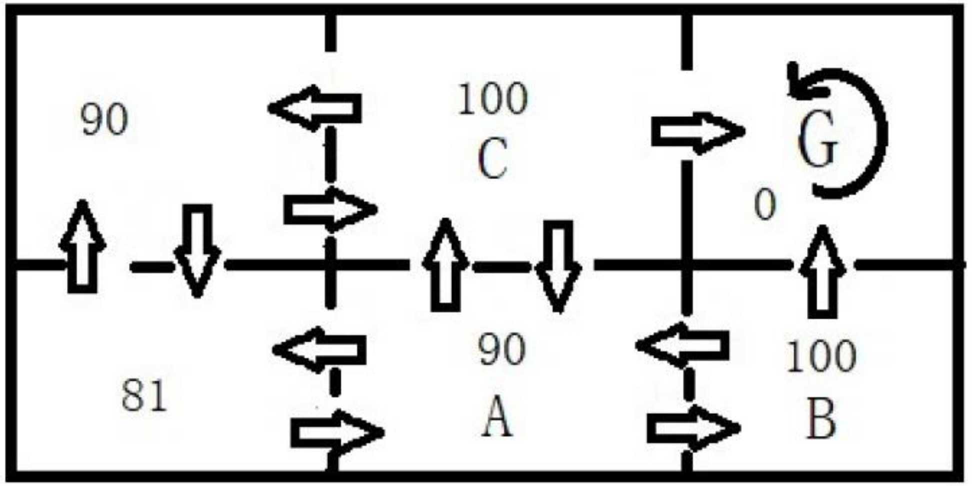
Six-Grid World V*(s) values

> The numbers labelled in each grid in the bottom figure represents the largest possible cumulative reward that the agent receives if it follows the optimal policy

> What is an optimal policy if the agent starts in A?

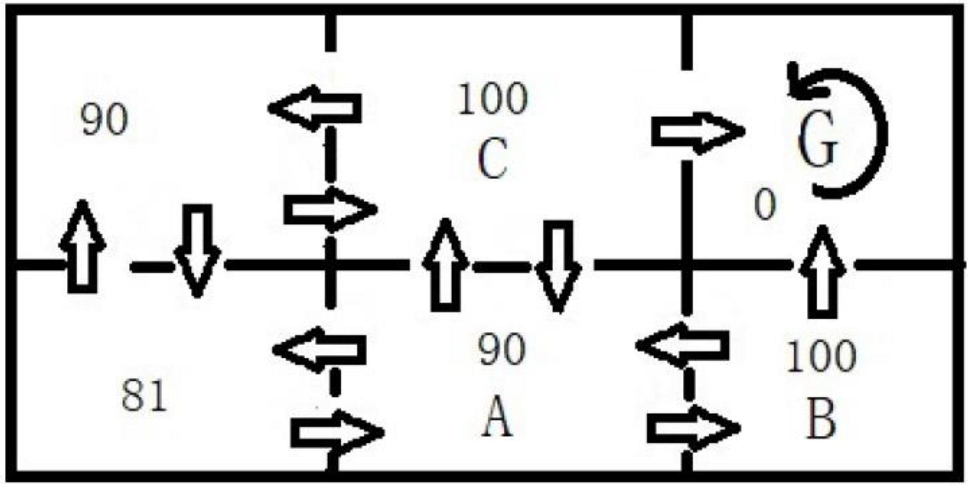
$$V^{\pi}(s = A) = 0 + 0.9 * 100 = 90$$





V*(s) as an Evaluation Function

- It is difficult for the agent to learn π* since it can only observe a sequence of immediate rewards
- Trick: “Learn” V*(s) by playing the game many times. Then, use V*(s) to develop a preference over different states
- Suppose the agent has learned V*(s), the values in the figure. If the agent is in A, what is the “best” action that it should perform?



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

Problems with Using V*(s)

> What if we don‘t have previous knowledge of the immediate reward and the subsequent state for each (s, a)?

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

> Solution: Need to do some clever rewriting and massage the above equation to bypass this problem.

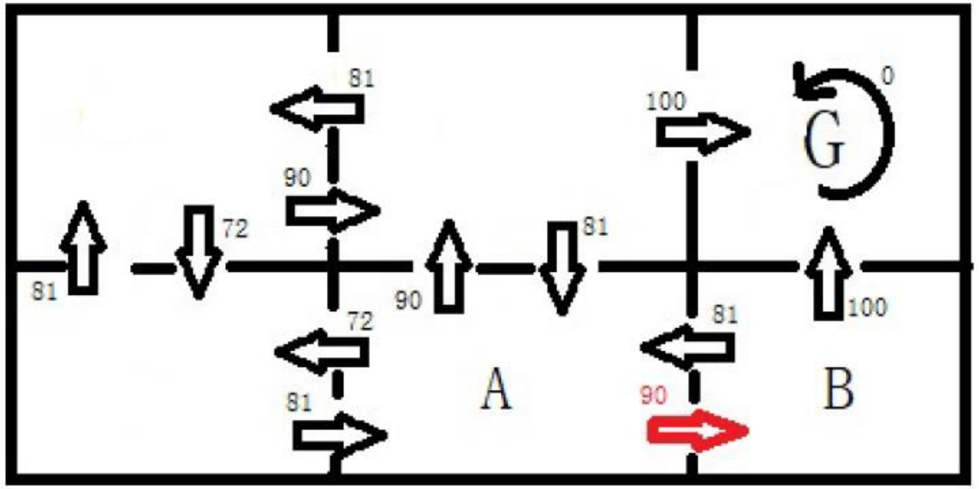
The Q-function

> Instead of evaluating each state, we now evaluate each state-action pair". How good it is to perform action a in state s?

> The value **Q** is the immediate reward, plus the discounted cumulative reward that the robot gets by following the optimal policy thereafter.

> Example:

$$Q(s = A, a = right) = 0 + 0.9 * 100 = 90$$



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

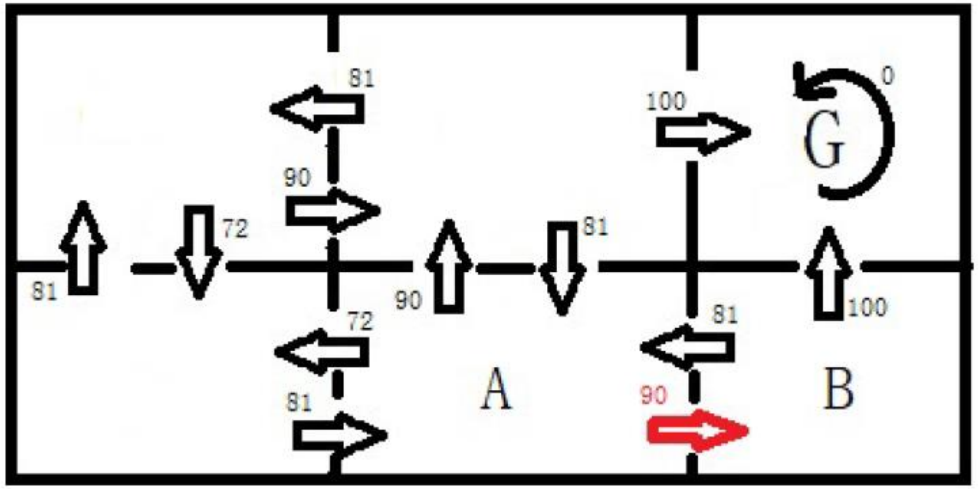
Selecting Optimal Actions with Q-values

> To choose the optimal action a in state s, we simply select the action with the highest Q-value.

> What is the agent‘s action in state A?

> In state B?

> The agent doesn‘t need prior knowledge of the reward and the subsequent state that it will recieve from the environment by performing a to select ist optimal action a*, once it learns Q.



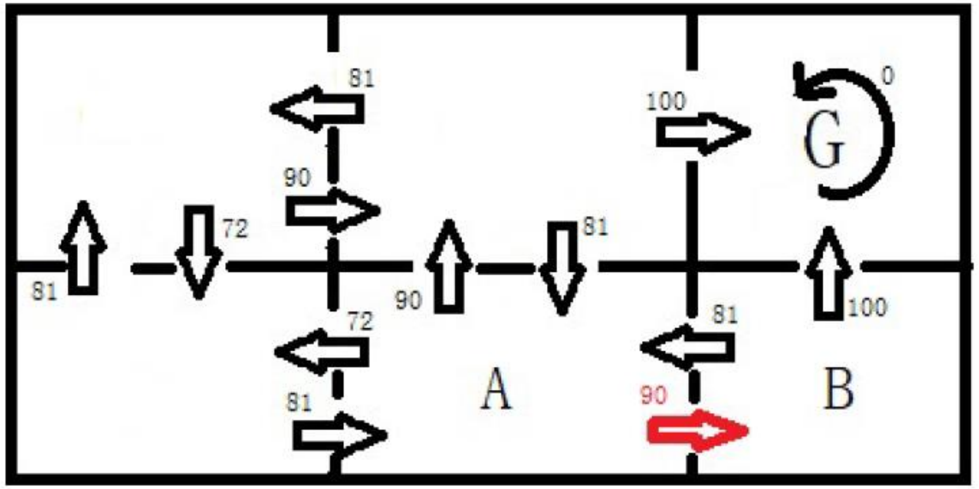
$$\pi^*(s) = \underset{a}{argmax}\ Q(s,a)$$

The Bellman Equation

- The Q-function can be rewritten as the Bellmann Equation:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q\left(\delta(s,a), a'\right)$$

- Learning Q corresponds to learning the optimal policy.
- Armed with the Bellman Equation, we can use a *Value Iteration* algorithm to estimate the Q values (Session3).
- What is Q(s=A, a=right) from the Bellmann Equation?



81
100
0
G
90
72
81
81
90
100
72
81
81
A
90
B