Agent skill learning and keepaway using parameterized policy search

By Rahul Dass

Advisor: Dr. Ubbo Visser

March 26, 2018

Outline

1. My interests: Reinforcement Learning (RL)
    a. RL in a nutshell
    b. MDP: planning and learning
    c. RL families: MFL and MBL

2. RL case study: RoboCup 3D Simulation Domain
    a. Layered Learning (UT Austin)

3. RL application: RoboCanes 3D Simulation Team
    a. How can RL be applied?
        i. Robot skill learning
        ii. Keepaway planning

1. RL in a nutshell

- Let *x*: instance (featurized representation)
  - What type of robot? Number of D.O.F? What kind of skills?
- Let *a*: action taken
  - Sprinted towards a soccer ball
- Let *v*: evaluative feedback
  - +2 for getting to the ball; -5 for falling down
- **Goal: construct a decision rule that maximizes expected value[3, 4]**

Why is this notion exciting?

Advantages

- ➢ Don't tell the learner what to do
- ➢ Learner just needs to know how to score

Challenges

- ➢ Nature of evaluation is **weak!**
- ➢ Receiving scalar values alone, agent does not know what it should be doing
- ➢ Agent cannot tell if the scalar feedback is a reward (+ve) or penalty (-ve)

RL feedback signal

- **Evaluative**
  - Not telling a robot which way to turn. Just when it sees the ball - signal to confirm or not.
- **Sampled**
  - Input that AIBOS gets is a colored histogram coming from cameras. Probably will not see the same colored patterns twice. Has to generalize between color patterns.
- **Sequential**
  - Robot starting in some initial *s*, takes *a*, gets no immediate *v* due to *a* taken.It continues making *(s, a)* pair movements.
  - When ball is located - it was somehow set up by the sequence of *a*'s taken

MDP: planning and learning

Q: How do we make agents take decisions in order to maximize reward?

- **M**arkov **D**ecision **P**rocess contains elements; the environment has 2-functions:
  - States, actions/decisions (discrete)
  - Transitions, rewards stationary, Markovian

- Transition function: $Pr(s' \mid s, a) = T(s, a, s')$
  - T takes current (s, a) pair
  - Output: probability distribution over all next states s'

- Reward function: $E(r \mid s', a) = R(s, a)$
  - Mapping agent being in s and taking a
  - E provides an evaluation of how good that a was

Agent's perspective

Optimal policy: which actions to take for which states that maximizes the total cumulative (expected, discounted) reward

- Policy: $\pi^*(s) = argmax_a\ Q^*(s', a')$
  - Mapping of states → actions
- $Q^*(s, a) = R(s, a) + \gamma\Sigma_i T(s, a, s')\ max_{a'} Q^*(s', a')$
  - Assume from *s' till end of trajectory, agent behaves optimally*
  - Q^* is an estimate of the total expected, discounted future reward for a single-step transition
  - This is the Bellman equation

Solving the Bellman equation = Decision making in a MDP

Solving Bellman: upshot

Given a reasonably sized MDP, there are 3 algorithms that can figure out the optimal way for an agent to behave that maximizes expected reward:

1. Value iteration converges in limit
2. Policy iteration converges in finite time
3. Linear programming runs in polynomial time

Downside:

- Major assumption that we'll be given such a MDP
- RL problem: agent is living in a world, and it needs to figure out on its own how the world works etc.[3]

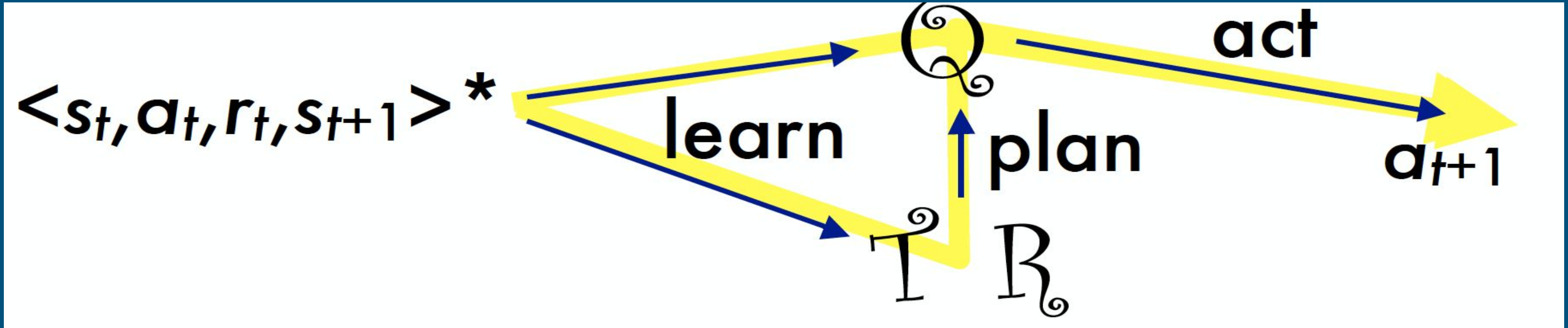
RL families[3]

<u>Model-free learning</u>

- Use experience <s, a, r, s'> to "learn" *Q* directly, then be able to take *a*
- Eg: Q-learning and SARSA

<u>Model-based learning</u>

- Use experience <s, a, r, s'> to "learn" *T* and *R*, which are used to compute *Q* and then can the agent determine what *a* to take
- Eg: R_{MAX} and E^3



2. RL case study

RoboCup 3D Simulation Domain[5, 6]

- Teams of 11 vs 11 autonomous robots playing soccer
- Realistic physics using Open Dynamics Engine (ODE)
- Simulated robots are modeled after the Aldebaran Nao robot
- Robot receives noisy visual information about environment
- Robots can communicate with each other over limited bandwidth channel

Head-to-Head

UT AustinVilla[6]

- **2007-2009**: no goals scored, no games won
- **2010**: improved walk but still unstable and falling frequently
- **2011-2017**: current champion
  - RL based Layered Learning approach

RoboCanes

- **Champion**: German Open (2011), US Open (2015)
- **2nd Place:** World Cup (2012, 2014), Iran Open (2011, 2012), US Open (2014)
- **3rd Place:** Iran Open (2014)
- **4th Place:** World Cup (2017)
- **Quarterfinalist**: World Cup (2015, 2013, 2011, 2010)

Direct policy search[3, 4]

- **Goal:** learn a parameterized policy that determines an agent's behavior

- **Method:** Optimization algorithm that produces candidate parameters for an agent to evaluate an optimization task, eg: robot kicking a ball

- **Evaluation and outcome:** upon task completion, agent evaluates parameters and returns fitness (reward)

- **Repeat:** algorithm generates new parameters to find a (optimal) policy that hopefully improves robot's ability and performance leading to higher fitness

CMA-ES as a baseline

Covariance **M**atrix **A**daptation **E**volutionary **S**trategy[13] is an evolutionary numerical optimization method

- Similar to a genetic algorithm being generation and population based
- However, unlike a genetic algorithm that uses a genetic operator, CMA-ES samples candidate members / parameter sets from a multivariate Gaussian distribution
- One generation of learning to the next, CMA-ES adjust the distribution to move to areas of higher fitness in parameter search space[5, 6]

Layered Learning[7]

- Hierarchical ML paradigm enabling learning of complex behaviors by incrementally learning a series of sub-behaviors
  - Higher layers directly depend on the learned lower layers

Goal: create "overall" optimal behavior policy

1. Sequential Layered Learning[7]
   a. After parameters are learned for first layer, they are frozen, then learn parameters for next layer
   b. Problem: Too limiting in a joint behavior policy search space. Parameters from first layer is cut off from next layer.
2. Concurrent Layered Learning[8]
   a. Learning parameters from first layer are kept open, when parameters for next layer are learning
   b. Problem: Increased dimensionality causes learning to be too hard / slow or intractable

Proposal: Overlapping Layered Learning[9]

Layered learning, both in series and parallel, where some parameters of previously learned layers are left open during learning of subsequent layers

- Tradeoff: freezing and keeping parameters open (overcomes SLL and CLL)
- Optimizes overlap between behaviors

1. Combining Independently Learned Behaviors (CILB)
   a. Learning 2 or more behaviors in parallel, relearn subset in next layer (conflicting behaviors)
2. Partial Concurrent Layered Learning (PCLL)
   a. Parameters are partly open after first layer of learning (highlighting tradoff)
3. Previously Learned Layer Refinement (PLLR)
   a. After behavior is learned + params frozen for two layers, part of first layer opened to relearn in presence of latter layer

3. RL application: RoboCanes 3D Sim Team

- What are we able to do so far?
  - Learning parameters from the walk engine
  - Learning of the kick is via CMA-ES, running

- Agent skill learning:
  - **Task 1:** Find an optimal approach to the ball via parameterized policy search
  - Challenge: Without a stable walk engine, policy search parameters will get affected

  - **Task 2:** Design a new walk based off double inverted pendulum approach (UT Austin)
  - Preliminaries: Cartpole toy problem OpenAI gym

  - **Task 3:** Multi-skill learning
    - Stable transitioning between skills, say: walk → kick → walk
    - Challenges: delay in ball positioning and robot instability

Keepaway planning

- Start with simplest 3 vs 2 keepaway scenario[10]
  - 3 agents from same team keeping possession of the ball from 2 agents attempting to take the ball away
  - Preliminaries: simplify problem in a 2D setting, modeled using the classic RL gridworld problem: agent getting to target with obstacles
  - Challenges: multi-agent skill learning and movement coordination

Thank you!

Dr. Visser - giving me the opportunity to do my Ph.D. here at UM. Being my mentor. Introducing me to the world of RL and robotics.

Dr. Schwartz - constant source of advice and support, especially our discussions of *RL in the brain, and Deep Learning.*

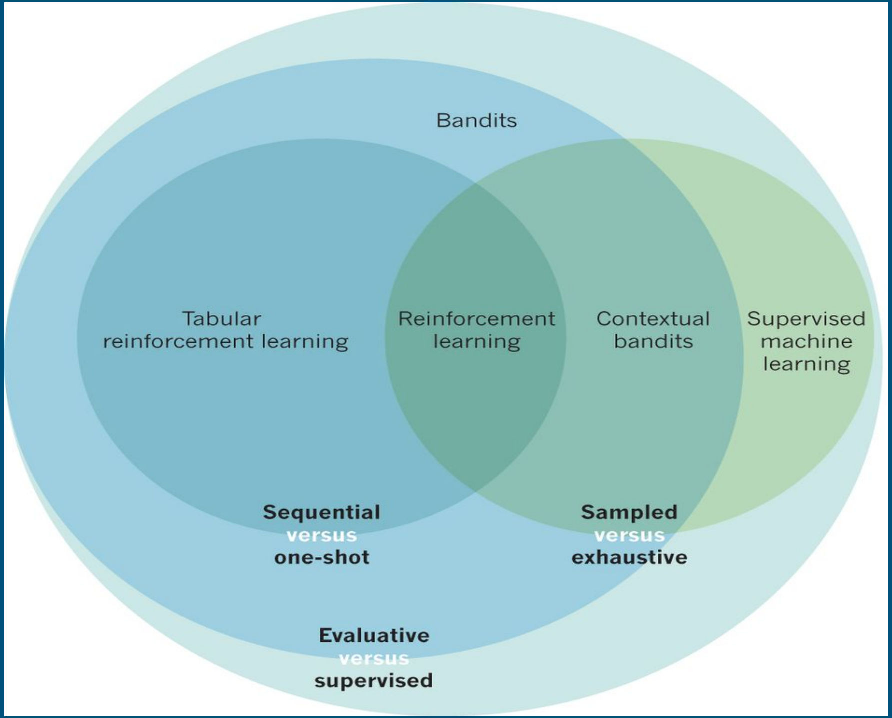
Dr. Sutcliffe - teaching me *how to give a talk!*

RoboCanes family - past, present and future

References

[1] Littman M., *Reinforcement Learning improves behavior from evaluative feedback* (**Nature, 2015**)
[2] Sutton, R. and Barto A. G., *Reinforcement Learning: An Introduction 1st Edition* (**MIT Press, 1998**)
[3] Littman M., *Basics of Computational Reinforcement Learning* (**RLDM, 2015**)
[4] Silver D., *Advanced Topics: Reinforcement Learning Course* (**UCL, 2015**)
[5] Stone P., *Robotic Skill Learning: From Real World to Simulation and Back - CMU RI seminar* (**2017**)
[6] MacAlpine P., *Multilayered Skill Learning and Movement Coordination for Autonomous Robotics Agents - Ph.D. Defense* (**2017**)
[7] Stone, P., and Veloso M., *Layered Learning* (**Springer, 2000**)
[8] Whiteson S., and Stone P., *Concurrent Layered Learning* (**AAMAS, 2003**)
[9] MacAlpine P., and Stone P., *Overlapping Layered Learning* (**AIJ, 2018**)
[10] Stone P., Sutton R., and Singh S., *Reinforcement Learning for 3 vs 2 keepaway* (**2001**)
[11] Hansen, N., Muller S. D., and Koumoutsakos P., *Reducing the Time Complexity of Derandomized Evolutionary Strategy with Covariance Matrix Adaptation (CMA-ES)* (**2003**)

Feedback revisited[1]



...there's quite a lot!

Solving Bellman: 3 algorithms[1-4]

a. Value iteration converges in limit
   i. Guess $Q_0(s, a)$. Substitute in Bellman to compute new Q-values by taking max (nonlinear). Q-value. Keep iterating, in the limit, converges to a Q-value that is solution to Bellman.

b. Policy iteration converges in finite time
   i. Guess $Q_0(s, a)$. Use it to build a policy $\pi_t(s)$ where a taken gives highest estimate of Q-value. Evaluate π_t using Bellman using Q. Either it returns same π as before (optimal), or gives a better one from which we generate new Q-value etc. Linear constraints → finite time.

c. Linear programming runs in polynomial time
   i. Express (if possible) MDP ast a linear program. Solve it. Solution to Bellman found