Robotics and Autonomous Systems

Lecture 28: Learning in Robots

Richard Williams

Department of Computer Science
University of Liverpool

UNIVERSITY OF LIVERPOOL

Today

- We have seen how difficult it is to program, down to the finest details, a control policy (a controller) for robots to carry out even simple tasks like obstacle avoidance
- Ideally, one would like to be able to simply specify what the robot is supposed to achieve:
  - Go from A to B without hitting obstacles

  without having to say how precisely that should be done
- Machine learning techniques
  - in particular Reinforcement Learning

  can be used to that purpose: the robot will then learn the how of the what the programmer has specified as task
- This is our subject today.

Little Dog

https://www.youtube.com/watch?v=nUQsRPJ1dYw

How learning might fit in

- Recall that in Jason you write rules/plans that invoke actions that are defined by the environment model.
- Imagine if these could be learnt.

How to decide what to do

- Consider being offered a bet in which you pay £2 if an odd number is rolled on a die, and win £3 if an even number appears.
- Is this a good bet?

How to decide what to do

- Consider being offered a bet in which you pay £2 if an odd number is rolled on a die, and win £3 if an even number appears.
- Is this a good bet?
- To analyse this, we need the expected value of the bet.

How to decide what to do

- We do this in terms of a random variable, which we will call X.
- X can take two values:
  3 if the die rolls odd
  -2 if the die rolls even
- And we can also calculate the probability of these two values
  $\Pr(X = 3) = 0.5$
  $\Pr(X = -2) = 0.5$

How to decide what to do

- The expected value is then the weighted sum of the values, where the weights are the probabilities.
- Formally the expected value of X is defined by:

$$E(X) = \sum_k k \Pr(X = k)$$

  where the summation is over all values of k for which $\Pr(X = k) \neq 0$.
- Here the expected value is:

$$E(X) = 0.5 \times 3 + 0.5 \times -2$$

- Thus the expected value of X is £0.5, and we take this to be the value of the bet.
- As opposed to £0 if you don't take the bet.

How to decide what to do

- Not the value you will get.
- But a value that allows you to make a decision.

How to decide what to do

- Another bet: you get £1 if a 2 or a 3 is rolled, £5 if a six is rolled, and pay 3 otherwise.
- The expected value here is:

$$E(X) = 0.333 \times 1 + 0.166 \times 5 + 0.5 \times -3$$

which is -0.33.

How an agent might decide what to do

- Consider an agent with a set of possible actions A available to it.
- Each $a \in A$ has a set of possible outcomes s_a.
- Which action should the agent pick?

How an agent might decide what to do

- The action a^* which a rational agent should choose is that which maximises the agent's utility.
- In other words the agent should pick:

$$a^* = arg \max_{a \in A} u(s_a)$$

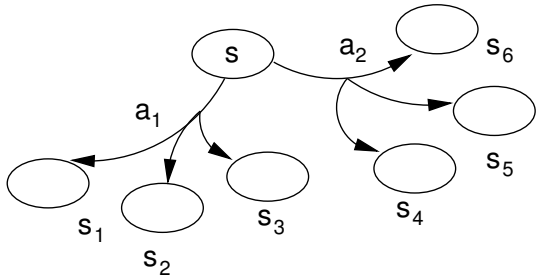
- The problem is that in any realistic situation, we don't know which s_a will result from a given a, so we don't know the utility of a given action.
- Instead we have to calculate the expected utility of each action and make the choice on the basis of that.

How an agent might decide what to do

- In other words, for the set of outcomes s_a of each action each a, the agent should calculate:

$$E(u(s_a)) = \sum_{s' \in s_a} u(s').\Pr(s_a = s')$$

and pick the best.



Sequential decision problems

- These approaches give us a battery of techniques to apply to individual decisions by agents.
- However, they aren't really sufficient.
- Agents aren't usually in the business of taking single decisions
  - Life is a series of decisions.

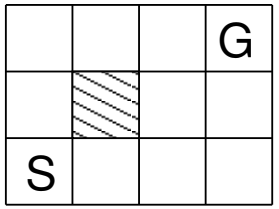
  The best overall result is not necessarily obtained by a greedy approach to a series of decisions.
- The current best option isn't the best thing in the long-run.

Sequential decision problems

- Otherwise I'd only ever eat chocolate cake.

Sequential decision problems

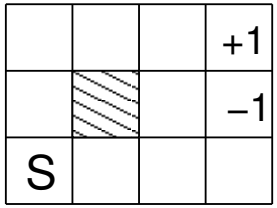
- Need to think about sequential decision problems where the agent's utility depends on a sequence of decisions.
- We saw something like this at the start of the semester.
  - Runs of an agent.



- To get from the start point (S) to the goal (G), an agent needs to repeatedly make a decision about what to do.

Rewards

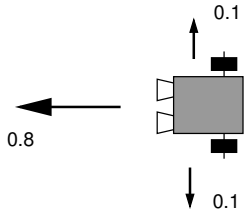
- Here we exchange the notion of a goal for the notion of a reward.
- It is easy to see which is the “goal” in this case:



- The action model is more complex than we saw before.
- Now actions are non-deterministic.

Motion model

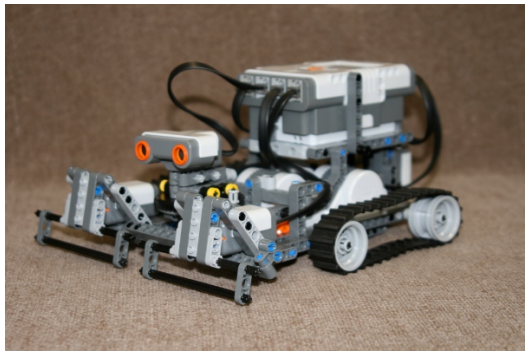
- If the agent chooses to move in some direction, there is a probability of 0.8 it will move that way.



- Probability of 0.2 it will move in the perpendicular direction.
- If the agent hits a wall, it doesn’t move.

Motion model

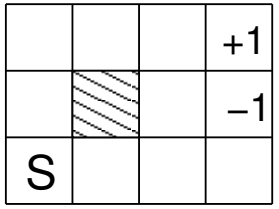
- As you know by know, this is an approximation to how a robot moves.



- Arguably a more accurate approximation than assuming that it will always do what it is programmed to do.

Motion model

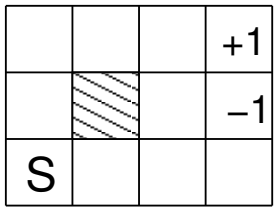
- If the agent goes $\{Up, Up, Right, Right, Right\}$



- It will get to the goal with probability $0.8^5 = 0.32768$ doing what it expects/hopes to do.

Motion model

- It can also reach the goal going around the obstacle the other way, with probability = $0.1^4 \times 0.8$.



- Total probability of reaching the goal is 0.32776.

Rewards

- To complete the description, we have to give a reward to every state.
- To give the agent an incentive to reach the goal quickly, we give each non-terminal state a reward of -0.04.
  - Equivalent to a cost for each action.
- So if the goal is reached after 10 steps, the agent's overall reward is 0.6.

Markov Decision Process

- This kind of problem is a Markov Decision Process (MDP).
- We have:
  - a set of states S.
  - an initial state s_0.
  - a set A of actions.
  - A transition model $\Pr(s'|s,a)$ for $s, s' \in S$ and $a \in A$; and
  - A reward function $R(s)$ for $s \in S$.
- What does a solution look like?

Policies

- A plan — a sequence of actions — is not much help.
  - Isn't guaranteed to find the goal.
- Better is a policy π, which tells us which action $\pi(s)$ to do in every state.
- Then the non-determinism doesn't matter.
  - However badly we do as a result of an action, we will know what to do.

Policies

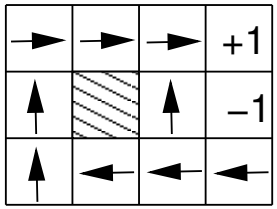
- Because of the non-determinism, a policy will give us different sequences of actions different times it is run.
- To tell how good a policy is, we can compute the expected value.
  - Compute value you get when you run the policy.
  - Can compute it by running the policy
- The optimal policy π^* is the one that gives the highest expected utility.
  - On average it will give the best reward.

Policies

- Given π^* an agent doesn't have to think — it just does the right action for the state it is in.

Policies

- The optimum policy is then:



- Note that this is specific to the value of the reward $R(s)$ for non-terminal states — different rewards will give different policies.

Bellman Equation

- How do we find the best policy (for a given set of rewards)?
- Turns out that there is a neat way to do this, by first computing the utility of each state.
- We compute this using the Bellman equation

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s,a) U(s')$$

- γ is a discount factor.

Value iteration

- In an MDP wth n states, we will have n Bellman equations.
- Hard to solve these simultaneously because of the max operation
  - Makes them non-linear
- Instead use an iterative approach
  - value iteration.
- Start with arbitrary values for utilities (say 0) and then update with:

$$U_{i+1} \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s,a) U_i(s')$$

- Repeat until the value stabilises.

Applications

- The Bellman equation(s)/update are widely used.

- D. Romer, It's Fourth Down and What Does the Bellman Equation Say? A Dynamic Programming Analysis of Football Strategy, NBER Working Paper No. 9024, June 2002

Applications

This paper uses play-by-play accounts of virtually all regular season National Football League games for 1998-2000 to analyze teams' choices on fourth down between trying for a first down and kicking. Dynamic programming is used to estimate the values of possessing the ball at different points on the field. These estimates are combined with data on the results of kicks and conventional plays to estimate the average payoffs to kicking and going for it under different circumstances. Examination of teams' actual decisions shows systematic, overwhelmingly statistically significant, and quantitatively large departures from the decisions the dynamic-programming analysis implies are preferable.

Partial observability

- For all their complexity, MDPs are not an accurate model of the world.
  - Assume accessibility/observability
- To deal with partial observability we have the Partially observable Markov decision process (POMDP).
- We don't know which state we are in, but we know what probability we have to being in every state.
- That is all we will say on the subject.

Reinforcement learning

- Ok, now we have the notion of an MDP, imagine we don't know what the model is.
- We don't know $R(s)$
- We don't know $\Pr(s'|s,a)$
- But it is simple to learn them — the agent just moves around the environment.

  `http://vimeo.com/13387420`

Reinforcement learning

- Since it knows what state s' it gets to when it executes a in s, it can count how often particular transitions occur to estimate:

$$\Pr(s'|s,a)$$

  as the proportion of times executing a in s takes the agent to s'.

Reinforcement learning

- Similarly the agent can see what reward it gets in s to give it $R(s)$.

Reinforcement learning

- If the agent wanders randomly for long enough, it will learn the probability and reward values.
- (How would it know what "long enough" was?)
- With these values it can apply the Bellman equation(s) and start doing the right thing.

Reinforcement learning

- The agent can also be smarter, and use the values as it learns them.
- At each step it can solve the Bellman equation(s) to compute the best action given what it knows.
- This means it can learn quicker, but also it may lead to sub-optimal performance.

Q-learning

- Q-learning is a model-free approach to reinforcement learning.
  - It doesn't need to learn $P(s'|s,a)$.
- Revolves around the notion of $Q(s,a)$, which denotes the value of doing a in s.

$$U(s) = \max_a Q(s,a)$$

- We can write:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) max_{a'} Q(s',a')$$

  and we could do value-iteration style updates on this.
- (Wouldn't be model-free.)

Q-learning

- However, we can write the update rule as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

  and recalculate everytime that a is executed in s and takes the agent to s'.
- α is a learning rate
  - Controls how quickly we update the Q-value when we have new information.

Introducing some supervision

- RL allows the agent to learn control policies from scratch
- However, when the state-space and the action-space are large, supervision can help bootstrap the learning process
- With supervision an agent is exposed to instances of positive and negative behavior, which get it starting in building its value function (the table of state-action pairs)
- Those instances do not need to show the robot the best solutions! The robot uses that input only as a starting point for its own learning

`http://vimeo.com/13387420`

Summary

- This lecture has introduced machine learning.
  - One aspect, reinforcement learning
- The idea is to have the robot figure out for itself how to do things.
- Just give it feedback.
- And, perhaps, some examples to help it get started.