COMSM0305: Learning in Autonomous Systems

Steve Gregory

(thanks to Tim Kovacs)

1. Introduction

Autonomous systems

- A system which learns autonomously is one which does not have a teacher
  - Most machine learning systems assume a 'teacher' in the form of a training set of (mostly) correctly labelled examples
    - I.e., supervised learning
  - We'll see how to learn from trial-and-error rather than labelled examples

Autonomous systems

- We cover two paradigms:
  - Evolutionary Computing (EC) / Evolutionary Algorithms (EAs)
    - mainly for black-box function optimization/design
    - e.g., find a wing shape minimizing drag and weight and maximizing lift
  - Reinforcement Learning (RL)
    - mainly for optimizing sequences of decisions, where each decision affects the next one
    - e.g., finding your way out of a maze

Autonomous systems

- Not normally taught together
  - closely related in aims (autonomous learning)
  - less close in how they work (algorithms)

Connections

- Both areas are related to many other (overlapping) areas, sometimes closely
  - Evolutionary Computing
    - Stochastic Optimisation
    - Biology (Evolutionary Theory)
  - Reinforcement Learning
    - Psychology (Learning)
    - Neuroscience
    - Control Theory
  - Both
    - Operations Research
    - Artificial Intelligence
    - Statistics

Biological analogies

- EC is analogous to adaptation of a species
- RL is analogous to adaptation of an individual
- We can use them together, in which case they interact
  - Genetics influences individuals
  - Individuals transmit genes
- Can be used to model nature (Computational Biology); e.g.:
  - EC models of population dynamics and interactions between learning and evolution
  - RL models of neural function
  - 'Life as we know it'

Biological analogies

- Can also simulate life (Artificial Life); e.g.:
  - Evolution of cellular automata
  - RL as brain of artificial agents
  - 'Life as it might be'
- However, both are usually used as optimization methods for (e.g.):
  - scheduling and routing problems
  - control systems for industrial processes or mobile robots
  - games like backgammon

Biological inspiration

Some other nature-inspired / bio-inspired computing paradigms:

- Neural networks
- Ant colony optimization
- Swarm intelligence
- Artificial Chemistry
- DNA computing
- Artificial Immune Systems

Biological inspiration

Nature is a great source of inspiration, but

- all these paradigms are simplified compared to nature
  - has something important been left out?
  - has something unimportant been included?
  - what exactly makes them work?
- nature's solutions may not be best for a computer
  - brains: highly parallel, very slow
  - desktop computer: mostly serial, very fast
  - solutions in nature are a subset of possible solutions
    - e.g., child can inherit genes from 10 parents in simulation; harder in life
  - nature cannot encode learned skills or knowledge into genes for our children; computer simulations can (Lamarckian evolution)
    - we can’t inherit knowledge of calculus from parents!

Biological inspiration

Nature is a great source of inspiration, but

- It can be hard to compare paradigms, due to different perspectives and terminology
  - may be easier to think of them as algorithms or statistical methods
  - what underlying algorithmic or statistical principles are being used?
  - often hard to analyse mathematically: complex systems
- Biological metaphors may take us only so far
  - statistical and computational principles may be a better guide
  - paradigms tend to start with biological inspiration but move away from them as the field progresses
- Ultimately what matters is how well these methods work
  - Just because it's cool, or something like what happens in nature, doesn't mean it's the best solution
  - There's no magic!

Engineering view

- RL is firmly rooted in Dynamic Programming
  - Can be seen as a form of asynchronous DP
- EC is a form of stochastic (randomized) search
  - Very much like (e.g.) simulated annealing
  - Defining feature: interaction of a population of solutions
- Both are optimization methods
  - There is a (possibly infinite) space of solutions
  - We have some measure of solution quality
  - We seek the highest quality solution in the least time
    - Enumerating all solutions is guaranteed to find the best
    - ...but is rather slow!
  - Both try to induce better solutions based on previous attempts
- Both are tools
  - There are other tools we will not cover: e.g., mathematical programming
  - You need the right tool for the current problem
  - Other tools can be simpler and more effective than EC or RL
  - Typically EC and RL are used when traditional methods cannot be used

RL: Training a dog

- RL models trial and error learning through rewards and punishments
- Like training a dog
- Dogs don't understand English, but...
  - Dogs learn to do things which are rewarded
  - Dogs learn to not do things which are punished
- We communicate with/program the dog using rewards and punishments
  - We don't tell it what we want
  - We don't need to show it examples of good or bad behaviour; we just need to recognize them
- We simply reward/punish the dog's behaviour
  - We don't know how the dog generates behaviour
  - We don't know how its mind or body work
- Easy for us, hard for the dog
  - The dog works out what we want from rewards/punishments
  - The dog rewires its brain and activates its muscles

RL: Training a computer (1)

- The computer's rewards and punishments are just numbers
  - E.g., positive numbers are rewards, negative numbers are punishments
  - Actually we only need some numbers to be higher than others
  - From now on we refer only to reward, without loss of generality
- We program it by rewarding behaviours
  - We don't tell it what we want
  - We don't need to show it examples of good or bad behaviour; we just need to recognize them
  - We use a *reward function* which defines the reward for each action an agent takes

RL: Training a computer (1)

- Suppose we want the computer to play backgammon
  - We can simply reward only the outcome of the game
    - E.g. +1 for winning, -1 for losing, 0 at all other times
  - We don't need to know how to play the game; we just need to recognize wins and losses
- How does this compare with supervised learning?
  - How much information does learning agent get?

RL: Training a computer (2)

- Easy for us, hard for the computer
  - The computer can learn to play better than we can
  - The computer can learn to do things we cannot
- ... at least in principle
  - In practise, the more help we give the computer, the better it will do
  - Just rewarding the outcome of games is not very helpful
    - Most of the time it gets reward 0, which offers little guidance

Nature trains dogs

- Dogs learn without humans
- Nature rewards and punishes dogs for their behaviour
  - Catch fresh food: reward (nutrition)
  - Eat rotten food: punishment (sick stomach)
- To train dogs, we hijack the mechanisms nature uses

Nature evolves dogs

- Dogs try to get as much reward as possible
- In evolution, an agent also tries to get as much reward as possible
  - In evolution, 'reward' = having children
  - We call this fitness
- Nature rewards effective behaviours with more children
  - Good at catching food: more children
  - Effective immune system: more children
  - Good eyesight: more children
  - Tend to fall off cliffs: fewer children
- Nature is a 'blind watchmaker'
  - It creates complex things which look like they must have been consciously designed
  - But really it just rewards success
- To breed dogs, we hijack the mechanisms nature uses
  - To get better eyesight, we select parents with good eyesight

Simulating evolution

We must have

- A population of individuals
- A way to evaluate their fitness: a *fitness function*
- A way to select parents, biased towards fitter ones
- A way to mix genes from parents to create offspring

- We specify what we want through the fitness function
  - Analogous to the reward function in RL
  - But we assign a fitness to the entire individual, based on its genes
    - instead of rewarding specific actions (like taking a queen, winning a game)

The law of unintended consequences

- Unwritten assumption: it’s easier to specify a reward or fitness function than to find the solution yourself
- Generally true, but designing the function can be difficult
- Sutton and Barto (3.2): it is not the place to put prior knowledge
- E.g., giving a reward for taking a queen may result in losing the game.

Further reading

- The Blind Watchmaker. Richard Dawkins, Norton & Co., 1986.
  - Video of Horizon (45 min.): http://www.dailymotion.com/video/x1jprj5_bbc-horizon-the-blind-watch-maker-richard-dawkins-1987_tech