COMP421 Machine Learning

RL by policy gradient

We have a stochastic policy $\pi_{a|\mathbf{s}}$: the probability of doing action a if you're in state $\mathbf{s}$.

- TD methods solve RL problems by learning a Values (V or Q), on the basis that if you know the true values, a suitable policy follows immediately: simply be greedy w.r.t. V or Q.
- The values are just a look-up table, but we saw how this could be made into a parameterized function instead, and the parameters learned. A major drawback of this approach is behaving greedily w.r.t. an *approximation* to V or Q can result in a terrible policy!

Today we consider a different approach: policy gradient methods.

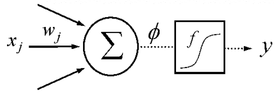
parameterise the policy instead of the value function

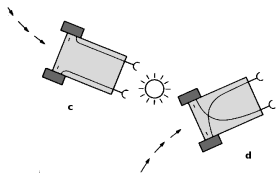
NB. Ignore time, for the moment.

Instead of parameterizing the value function, consider **parameterising the policy directly**. Instead of $\pi_{a|\mathbf{s}}$ being a look-up table or probabilities, make it a function $\pi_{a|\mathbf{s}}(\theta)$ where θ are some learnable parameters.

eg.

a sigmoid "neuron". The vector $\mathbf{s}$ could be the input, weights $\mathbf{w}$ could be the parameters, and the binary output could determine two possible actions (say, left or right for a robot).





learning the policy (in a scenario without time)

An agent in state $\mathbf{s}$ chooses an action a from its policy (updates its neuron..!), which results in reinforcement r.

If r is positive, we might want to "reward" this choice of action by making it more likely to occur next time the agent is in $\mathbf{s}$ (and if $r < 0$ we might want to "punish" the choice).

This can be achieved by this learning rule:

$$\Delta\theta = \eta \; r \; \underbrace{\nabla_\theta \log \pi_{a|\mathbf{s}}}_{\text{makes action } a \text{ more likely, from state } \mathbf{s}}$$

- η is a learning rate
- r is the scalar reinforcement signal
- ∇ is shorthand for the gradient, for each element of a vector

learning the policy (with 2 time steps)

Our agent is actually carrying out actions in a long sequence. Shouldn't we reward previous decisions too?

temporal credit assignment problem

of all the actions that were taken prior to receiving reward r, which should be rewarded?

At time t we could reward both the previous action, **and the one before that**, like this:

$$\Delta\theta = \eta \, r \left[\underbrace{\nabla_\theta \log \pi_{a_t|\mathbf{s}_t}}_{\text{makes } a_t \text{ more likely}} + \gamma \underbrace{\nabla_\theta \log \pi_{a_{t-1}|\mathbf{s}_{t-1}}}_{\text{makes } a_{t-1} \text{ more likely}} \right]$$

- γ is a discounting parameter. If $\gamma = 1$, all previous actions are held equally responsible for the latest reinforcement. If $0 < \gamma < 1$, responsibility fades with time.

The gradient

Generalising this, we have the gradient for a long sequence:

$$\Delta\theta \;=\; \eta\, r \underbrace{\left[\sum_{\tau=0}^{\infty} \gamma^{\tau}\, \nabla_{\theta} \log \pi_{a_{t-\tau}|\mathbf{s}_{t-\tau}} \right]}_{\text{could call this "eligibility" } \xi}$$

It seems we need to remember past gradients for a long time. (*c.f.* learning values: we seemed to need to wait until the end of an episode before updating, which motivated the TD trick).

But the eligibilities "compound" additively, which allows an online learning algorithm....

(One) Policy Gradient learning algorithm (optional)

At each time step t, upon receipt of reinforcement r:

update parameters, θ

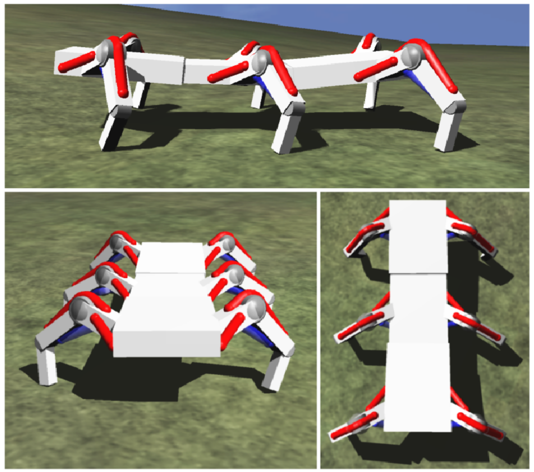
$$\Delta\theta \;=\; \eta\, r\, \xi$$

update eligibilities, ξ

$$\xi \;\longleftarrow\; \gamma\, \xi \;+\; \nabla_{\theta} \log \pi_{a_t|\mathbf{s}_t}$$

- "eligibility" ξ is vector: one for every parameter
- local in time (just like TD)
- *applicable to any parameterised learner!*
- okay for partially observable problems (still converges, provided η small enough etc.), so can use an internal representation of the world.

example



From Tim Field's MSc, see http://tinyurl.com/marcusfrean

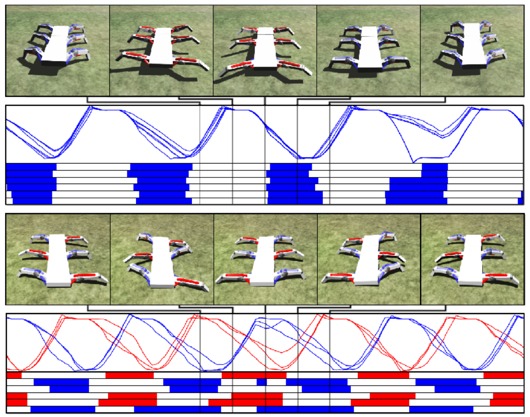


Figure 5.8: **Hexapod Gaits.** These diagrams show the two gaits learnt: the pronking *(top)* and tripod *(bottom)* gaits. The graphs show the lateral angle and foot contact for each leg (separated by colour into each tripod for the tripod gait).

summary on policy gradient RL

pros

- guaranteed convergence, with function approximation
- often policies easier to represent than value functions
- can deal with partial observability

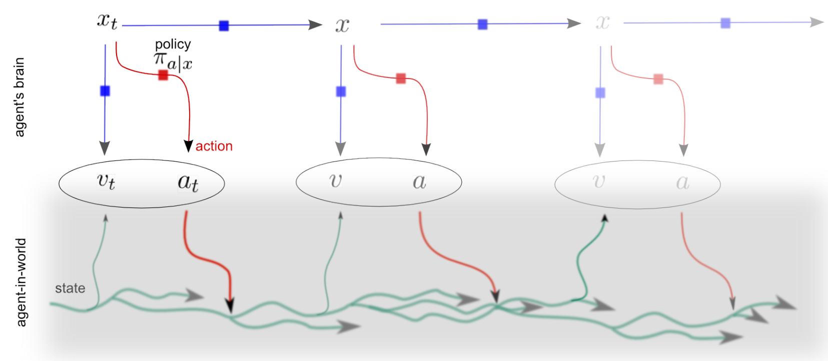
cons

- lose possibility of convergence to *global* optimum
- finding a good policy representation can be extremely difficult

---

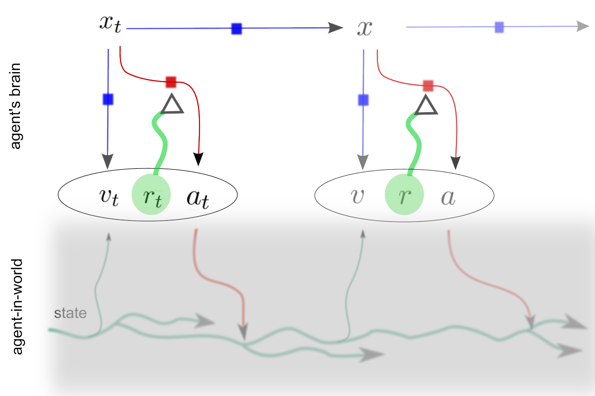
Reinforcement learners end up parameterising either the value function or the policy. The latter are more robust, but the problem is a long way from being solved: RL algorithms are very slow / require a *lot* of experience. At present they're only useful in worlds where we have a good model to practice on (like GAs and GP), because they're so wasteful of data.

For discussion: musing about real creatures...



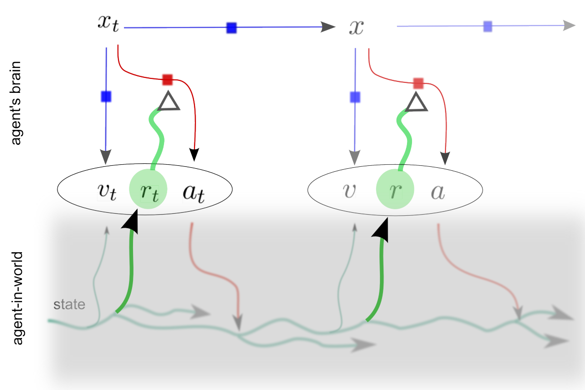
- a reactive agent
- evolution acts on the policy (and on representation x, it's priors, and learning algorithm)
- or could *learn* the policy in a supervised fashion (via EM).

For discussion: musing about real creatures...



- an agent learning from reinforcement signals
- we assume r comes from an inspired source (an engineer)

For discussion: evolved RL systems



- "reinforcement": anything relevant to **changing the policy**
- *i.e. anything correlated with long-term survival & reproduction*
- evolution links states to policy changes, where these lead to survival and reproduction
- *e.g.* pleasure and pain systems