Building a game player → Evaluation function

$$f(b) = \sum w_i \phi_i(b)$$

w_i: learned by playing games

$\phi_i(b)$: domain knowledge

Set of actions: for each a there is a reward $R(a)$

$R(a)$ might be random (in general, drawn from a distribution)

Ad placement: ϕ_i → words/content in ad

Find ad a s.t. we obtain max reward $R(a)$

a_i → try it n_i times: $R_1 \dots R_{n_i}$

Find $\vec{w}_i$ (weight or parameter vector) → array

s.t. $w_i \cdot \phi_i$ → correct-ish estimate of $R(a_i)$

$$\sum_j w_{i,j} \phi_{i,j}(a)$$

Minimize the error between our estimates and what we get from the environment

Idea 1

Error function:

$$\min_h \sum_{j=1}^{n_i} (h(j) - R_j)^2$$

R_j: actual data

all data

$h(j)$: prediction on j^{th} try

E.g. $h = \sum w \cdot \phi_i$ in our previous case.

$h(w)$

$$\min_w \sum_{j=1}^{n_i} (h(w, j) - R_j)^2$$

take derivative wrt w and set them to 0 to find w

In linear case: we can solve this in closed form (nice!)

Look for action a_i that maximizes reward (according to our estimates)

$\arg\max h(w_i)$

↳ best action

Exploration - exploitation dilemma:

- try out something new? or
- use the best guess you have so far?

A1 → payoff of 100
A2 → payoff of 10 | Big difference → A1

Regret: how much loss do I get from not doing the optimal thing

A1 → payoff of 100 → ?
A2 → payoff of 99
] A2 not much worse

A1 → payoff of 100
A2 → payoff of 99
] Both tried 10^6 times
Lots of data → more "greedy"

Randomized strategies for picking actions
↳ Best actions according to est est picked more
↳ a_i and a_j have close reward est → $p(a_i) \approx p(a_j)$
↳ Action that is best according to our est should see into prob → 1 as data → ∞

Boltzmann distribution
R^* - best est. so far
$p(a_i)$ prop to $(R^* - R_i)/\sigma_{mi}$
↑ "temperature"

Bandit problem → really used for ad placement

Delayed rewards: reward for some action is seen much later!

Suppose we are in some situation s (eg board position)

$s \xrightarrow{a} s'$

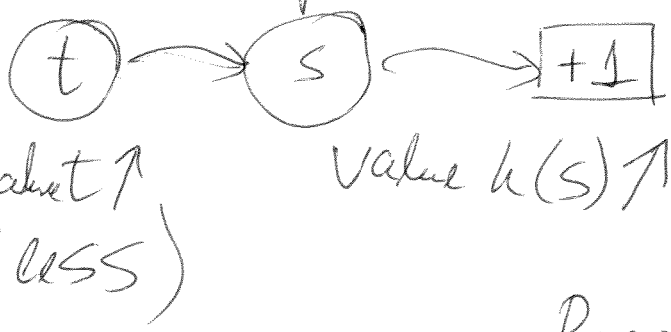
a: action; s': new situation (random)

$h(s)$ $\quad r + h(s') \rightarrow$ using our real fn

r: intermediate rewards (eg. capture a piece)

Train h to be similar to $r + h(s')$

↳ adjust its parameters



Propagate rewards **back** to previous situation

adjust h

Similar idea to how classical conditioning works.

Example applications

↳ Game playing

↳ Automatically flying helicopters

state/situation: set of pos/velocities + sensors

action: how much torque on rotor?

reward: +100 for reaching destination
-1000 for crashing
0 otherwise

=> Train! → simulations

Choosing actions → max preferred (innate bias of alg)

Robots → save

↳ Stock market!