Math Modeling, Week 2

1. Show that probability matching is a special case of Luce choice. That is, consider a task with two actions, A and B, exactly one of which is correct on each trial. Probability matching means making a prediction $P(A)$ for the probability that A will be correct, and choosing actions with probabilities $\Pr[a = A] = P(A)$ and $\Pr[a = B] = 1 - P(A)$. (This is what the simulation from last week did.) Assuming a reward of 1 for being correct and 0 for being incorrect, work out the expected rewards for both actions according to P(A), and then derive the action probabilities given by the Luce choice rule. If you're enjoying this, work out the action probabilities for softmax, and for Luce and softmax under different reward values for right/wrong (instead of 1/0).

Expected reward under action A

$\mathrm{E}[R|a=\mathrm{A}] = \Sigma_r \Pr[R=r|a=\mathrm{A}]\cdot r = \Pr[R=0|a=\mathrm{A}]\cdot 0 + \Pr[R=1|a=\mathrm{A}]\cdot 1 = (1-P(\mathrm{A}))\cdot 0 + P(\mathrm{A})\cdot 1 = P(\mathrm{A})$

Expected reward under action B

$\mathrm{E}[R|a=\mathrm{B}] = \Sigma_r \Pr[R=r|a=\mathrm{B}]\cdot r = \Pr[R=0|a=\mathrm{B}]\cdot 0 + \Pr[R=1|a=\mathrm{B}]\cdot 1 = P(\mathrm{A})\cdot 0 + (1-P(\mathrm{A}))\cdot 1 = 1-P(\mathrm{A})$

Luce choice

$\Pr[a=\mathrm{A}] = \mathrm{E}[R|a=\mathrm{A}]/(\mathrm{E}[R|a=\mathrm{A}]+\mathrm{E}[R|a=\mathrm{B}]) = P(\mathrm{A})/(P(\mathrm{A})+1-P(\mathrm{A})) = P(\mathrm{A})$

Therefore Luce choice produces probability-matching behavior.

Softmax

$$\Pr[a=\mathrm{A}] = \frac{e^{\mathrm{E}[R|a=\mathrm{A}]/T}}{e^{\mathrm{E}[R|a=\mathrm{A}]/T} + e^{\mathrm{E}[R|a=\mathrm{B}]/T}} = \frac{e^{P(\mathrm{A})/T}}{e^{P(\mathrm{A})/T} + e^{(1-P(\mathrm{A}))/T}} = \frac{1}{1 + e^{-\left(P(\mathrm{A})-\frac{1}{2}\right)\cdot 2/T}}$$

This is a logistic function of $P(\mathrm{A})$, meaning the logodds of the response is a linear function of $P(\mathrm{A})$ with intercept at ½ (i.e. $\Pr[a=\mathrm{A}]=½$ when $P(\mathrm{A})=½$) and slope $2/T$.

Generic reward values

If the rewards are r for correct and w for incorrect, then following the calculations above yields

$\mathrm{E}[R|a=\mathrm{A}] = w + (r-w)\cdot P(\mathrm{A})$, $\mathrm{E}[R|a=\mathrm{B}] = r + (w-r)\cdot P(\mathrm{A})$

Luce: $\Pr[a=\mathrm{A}] = \frac{w+(r-w)P(\mathrm{A})}{r+w}$

Softmax: $\Pr[a=\mathrm{A}] = \frac{1}{1+e^{-\left(P(\mathrm{A})-\frac{1}{2}\right)\cdot 2(r-w)/T}}$. This is a logistic function with slope $2(r-w)/T$.

2. Special cases of state-value learning ($\Delta V(s_t) = \varepsilon[R_t + \gamma V(s_{t+1}) - V(s_t)]$)
(a) What happens when $\gamma = 0$? How does the model compare to the simpler model from last week?

Future rewards are ignored, and the model learns only about immediate rewards. In that sense it corresponds to the simple RL model from Week 1 (except that there's separate learning for each state).

$\Delta V(s_t) = \varepsilon[R_t - V(s_t)]$

(b) What happens when there's only one state? Write a simplified version of the learning rule for that case. What does the value converge to, i.e. when is it in equilibrium?

If there's only one state, we can call it s and drop the t and t+1 subscripts:

$\Delta V(s) = \varepsilon[R_t + \gamma V(s) - V(s)] = \varepsilon[R_t - (1-\gamma)V(s)] = (1-\gamma)\varepsilon[R_t/(1-\gamma) - V(s)]$

Therefore we can think of $V(s)$ learning to approximate $R/(1-\gamma)$, with a learning rate $(1-\gamma)\varepsilon$.

(c) For the one-state case, define a new variable $W = (1-\gamma)V$. How does W behave?

It's important to realize we're not changing the model. We're just describing it with different parameters (W instead of V). Building on the derivation above, we have

$\Delta W = (1-\gamma)\cdot\Delta V = (1-\gamma)\varepsilon[R_t - (1-\gamma)V] = (1-\gamma)\varepsilon[R_t - W]$

Therefore W learns to approximate R (the immediate reward), with a learning rate $(1-\gamma)\varepsilon$.

3. Think of some ways to make the Q-learner smarter in the Gridworld task. If you can, implement one and try it out.

Simulated annealing

Reduce the temperature over time. This lets the agent explore more early on when knowledge is weak and exploit more later once it's expert at the task. One neat property of Q-learning is that it learns the action values for optimal (exploit-only) behavior, even while the agent is exploring. (This is because of the max operator for the next action, in the update rule for Q.) Thus under simulated annealing the agent is learning the same Q values the whole time, even though the response rule is changing.

State generalization

Nearby states are likely to have similar state and action values. Therefore, rather than updating the Q values only for the state that was experienced, the agent could update Q values for surrounding states as well (with a smaller learning rate). This generalization leads experience to be pooled across neighboring states, thus speeding learning.

Multistep backups

Rather than updating the Q values for just the most recent state, the agent could update for earlier states as well (using smaller learning rates). This strategy goes under the name *eligibility traces*, because past states have something like a memory trace that keeps them eligible for learning. Under this strategy, when a sequence of actions leads to prediction error, all actions in the sequence can be learned about, rather than just the final action.