Homework 1 Solutions

1. Play with the code
a) Execute the 4 blocks of code in order and look at the graphical results.
b) Change the paradigm (1, 2, or 3), and the learning rate, and explore how things change.
c) Test the model on a different paradigm (i.e., cue-outcome schedule), or try something else creative.

Here's a paradigm that has been used to demonstrate what's called the pre-exposure effect. One cue is presented several times with no outcome. Then that cue and a novel cue are each presented with the outcome. Learning for the pre-exposed cue is slower than that for the novel cue.

```
s(1:10,1) = 1; s(1:10,2) = 0; %first 10 trials: cue 1 present, cue 2 absent
R(1:10) = 0; %first 10 trials: no reward
s(11:2:49,1) = 1; s(11:2:49,2) = 0; %odd trials: cue 1 present, cue 2 absent
s(11:2:49,1) = 0; s(11:2:49,2) = 1; %odd trials: cue 1 absent, cue 2 present
R(11:49) = 1; %reward present in all phase 2 trials
```

Simulating the model on this paradigm reveals no difference in learning between the two cues. This is because the pre-exposure phase has no impact on the model. The only memory in the model is its current weights; it doesn't know whether those weights are based on ignorance or abundance of experience. One solution that's been proposed is that the learning rate for the pre-exposed cue decreases during phase 1, in what's called learned inattention. Bayesian models can also explain the pre-exposure effect, by tracking not just a point estimate for each weight but also the uncertainty in that estimate.

2. A pathology with high learning rates
a) What happens if $\varepsilon > ½$? Why? (Hint: simulate the model and then look at the values of p.)
b) That was for 2 cues. In general, with *k* cues, how large can ε be before the same pathology appears? How could the model be modified to avoid this problem?

Consider a case where the same stimulus configuration is presented on consecutive trials. How much does learning on the first trial change the prediction on the next trial?

$$\Delta P = \sum_i S_i \cdot \Delta w_i = \sum_i S_i \cdot \varepsilon\delta S_i = \varepsilon\delta \cdot \Sigma_i S_i^2$$

The biggest that $\Sigma_i S_i^2$ can be is k. In that case, we have $\Delta P = \varepsilon k\delta$. So, if $\varepsilon k > 1$, meaning $\varepsilon > 1/k$, the change in prediction will be larger than the prediction error ($\Delta P > \delta$), and P will overshoot the observed outcome (R). That is, when $R = 0$, P' (the prediction on the next trial) will be negative, and when $R = 1$, P' will be greater than 1.

Having the model's prediction lie outside [0,1] might not be a problem in itself, but it causes problems for our response rule, $\Pr[r = 1] = P$. In particular, this means we get nonsensical likelihoods when we evaluate model fits. One solution for this would be to censor at 0 and 1, meaning a response rule of $\Pr[r = 1] = \min\{\max\{0,P\},1\}$.

If $\varepsilon > 2/k$, then we get an additional problem. In that case, $\Delta P > 2\delta$, which implies $|P'-R| > |P-R|$. That is, P gets further from R, and over time it diverges to ∞.

More generally, it seems undesirable for the effective learning rate to depend on k. One option would be to change the learning rule to assume the learning rate is divided across the different cues, rather than replicated on every cue: $\Delta w_i = (\varepsilon/\Sigma_j S_j^2)\cdot\delta S_i$. Then ΔP on repeated trials will always equal $\varepsilon\delta$, regardless of the number of cues present.

3. Separate learning rates
a) Modify the code to allow a separate learning rate for each cue.
b) Generate data by simulating the common-ε model, then fit it using both the common-ε and the separate-ε models. The latter will involve a joint search over ε_i for all *i* (I suggest limiting to 2 cues). How much better does the separate-ε model fit?
c) Write a loop around steps 3a and 3b, to generate a sampling distribution of the difference in loglikelihood between the two models. What can you observe about this distribution?

```
%setup
n = 50; %number of trials
k = 2; %number of cues
N = 1000; %number of datasets to generate and fit
e = .1; %learning rate for data-generating model
E = .01:.01:.5; %range of learning rates to be evaluated in fitting models
```

```
s = zeros(n,k); %stimulus matrix (trial x cue)
R = zeros(n,1); %outcome sequence
p = zeros(n,1); %prediction on each trial
r = zeros(n,1); %response on each trial
diff = zeros(1,N); %for tracking results: difference in fit between models

for i=1:N

    %create cue-outcome schedule (using 2-partial-cue paradigm)
    s = randi([0 1],n,k); %independent random cues
    R = rand(n,1) < s*[.5;.5]; %cues contribute additively

    %simulate common-e model to create test data
    w = zeros(k,n+1); %initialize weight vector
    for t=1:n %loop through trials
        p(t) = s(t,:)*w(:,t); %expected outcome
        r(t) = rand<p(t); %simulated response
        d = R(t) - p(t); %prediction error
        w(:,t+1) = w(:,t) + e*d*s(t,:)'; %learning update
    end

    %fit common-e model to the data
    Lcommon = zeros(length(E),1); %total log-likelihood for each model
    for m = 1:length(E) %loop through candidate models
        w = zeros(k,n+1); %initialize weight vector
        for t=1:n %loop through trials
            p(t) = s(t,:)*w(:,t); %expected outcome
            %add log-likelihood of current response given model's prediction:
            Lcommon(m) = Lcommon(m) + log(abs(1-p(t)-r(t)));
            d = R(t) - p(t); %prediction error
            w(:,t+1) = w(:,t) + d*E(m)*s(t,:)'; %learning update
        end
    end
    fitCommon = max(Lcommon); %loglikelihood of best-fitting common-e model

    %fit separate-e model to the data
    Lseparate = zeros(length(E),length(E)); %total log-likelihood for each model
    for e1 = 1:length(E) %loop through values for first learning rate
        for e2 = 1:length(E) %loop through values for second learning rate
            w = zeros(k,n+1); %initialize weight vector
            for t=1:n %loop through trials
                p(t) = s(t,:)*w(:,t); %expected outcome
                %add log-likelihood of current response given model's prediction:
                Lseparate(e1,e2) = Lseparate(e1,e2) + log(abs(1-p(t)-r(t)));
                d = R(t) - p(t); %prediction error
                w(:,t+1) = w(:,t) + d*[E(e1);E(e2)].*s(t,:)'; %learning update
            end
        end
    end
    fitSeparate = max(max(Lseparate)); %loglikelihood of best-fitting common-e model

    diff(i) = fitSeparate - fitCommon; %difference in fit between models
    disp(i)
    if fitSeparate<fitCommon,break,end
end
```