Project 2: Rescorla-Wagner

For this project, you will use MATLAB to illustrate six predictions of the Rescorla-Wagner model for classical conditioning.

Background

The Rescorla-Wagner model and its variants are used today as computational models for learning and decision-making in humans and non-human animals. The core concept with Rescorla-Wagner, as with all computational models of behavior, is to do what physicists have done for centuries: use mathematical models to make predictions about real-world phenomena. As described here, the Rescorla-Wagner model makes predictions about the degree of psychological association, or "associative strength" between a stimulus which naturally produces a response, called the "unconditioned stimulus" or "US", and some other stimulus that is presented to the observer at the same time as the US, called the "conditioned stimulus" or "CS". Over time, if the US and CS are presented together, the CS will be "conditioned" (hence the name) to produce the same response as the US (which does not need conditioning, hence the name).

As a running example, consider Pavlov's classic conditioning experiment in which a dog is conditioned to salivate to the sound of a bell (the conditioned stimulus) when the sound of the bell occurs at the same time the dog receives food (the unconditioned stimulus). The Rescorla-Wagner model attempts to predict the associate strength V of the CS (bell) to the US (food); in this example, the amount of saliva the dog produces upon hearing the bell can be used as a proxy for the associative strength of the CS. When the sound of the bell is repeatedly paired with food, the associative strength V increases and the dog salivates more when it hears the bell because the bell predicts the arrival of food; likewise, when the sound of the bell is presented without food, the associative strength V decreases and the dog salivates less when it hears the bell because the sound of the bell no longer predicts food.

Each presentation of the CS, with or without the US, is called a "trial" and produces a change in V, denoted ΔV ("delta V"). When the CS (bell) is presented with the US (food), ΔV is positive, and when the CS is presented without the US, ΔV is negative. Defined in this way, we can write

$$V_{new} = V_{old} + \Delta V$$

to describe how associative strength changes across trials. After each trial, the associative strength (V_{new}) can be computed as the sum of the associative strength before the trial (V_{old}) and the effect of that trial on associative strength (ΔV).

Here is the Rescorla-Wagner rule, one of many rules by which V may be updated across trials:

$$\Delta V = \alpha(\lambda - V)$$

Besides ΔV and V, this model contains two parameters: λ ("lambda") and α ("alpha"). The parameter λ is called the "asymptote" and is the value of V that can potentially be reached given a hypothetical infinite number of trials. For instance, if the sound of the bell can be fully associated with food, such that the dog salivates just as much to the sound of the bell as it does to food, then λ=1. The parameter λ changes depending on the type of trial; if the CS is presented with the US, then λ is the maximum possible value of V, but if the CS is presented without the US, then λ is the minimum possible value of V (usually zero). The role of λ as the asymptote can be seen in the Rescorla-Wagner rule: as V approaches λ, the difference λ-V decreases to zero and ΔV approaches zero. The parameter α, which ranges from 0 to 1, is the

learning rate, which determines how quickly *V* changes across trials, and is determined by many factors including experimental design, animal, nature of the CS and US, and interactions among these three factors. The role of α as the learning rate can also be seen in the Rescorla-Wagner rule: higher values of α lead to higher values of ΔV, which result in faster learning.

The key quantities in the Rescorla-Wagner model are summarized below:

V = the current associative strength of the CS (bell).
ΔV = the change in associative strength of the CS during a trial
$V_{new} = V_{old} + \Delta V$ = the new value of V after a trial of Rescorla-Wagner updating
Rescorla-Wagner rule: $\Delta V = \alpha(\lambda - V)$
λ (lambda) = maximum associative strength of CS-US pair
α (alpha) = learning rate parameter

The key term is $(\lambda - V)$: The further the current associative strength V is from its asymptote λ, the faster learning occurs. This is the "principle" behind Rescorla-Wagner learning.

The Rescorla-Wagner model also predicts some results from compound conditioning, an experimental setup in which two CS's, *A* (e.g., a bell) and *B* (e.g., a light) are presented together. In this setup, there are two associative strengths, one for *A* (V_A) and one for *B* (V_B), which are summed to yield the associative strength of the compound CS *AB*, defined as the simultaneous presentation of *A* and *B*:

$$V_{AB} = V_A + V_B$$

Each CS also has its own learning rate: α_A for *A* and α_B for *B*. In compound conditioning, the two CS's are updated separately

$$V_{A,new} = V_{A,old} + \Delta V_A$$
$$V_{B,new} = V_{B,old} + \Delta V_B$$

using the same Rescorla-Wagner rule, but with V_{AB} determining the change in both *V*'s:

$$\Delta V_A = \alpha_A (\lambda - V_{AB})$$
$$\Delta V_B = \alpha_B (\lambda - V_{AB})$$

The key quantities in the Rescorla-Wagner model for a compound CS are described below:

V_A = the current associative strength of the CS *A* (bell)
V_B = the current associative strength of the CS *B* (light)
$V_{AB} = V_A + V_B$ = the current associative strength of *A* and *B* together
ΔV_A = the change in associative strength of the CS *A* during a trial
ΔV_B = the change in associative strength of the CS *B* during a trial
α_A = learning rate for A
α_B = learning rate for B

CONTINUED ON NEXT PAGE

Part 1: Functions

The first step in using the Rescorla-Wagner model is creating functions which implement the Rescorla-Wagner updating rule for a single CS and for a compound CS. The header line for these functions (first line of code) should be as follows:

Single CS: `function V=rwRule(V,alpha,lambda)`
Compound CS: `function [VA,VB]=rwABRule(VA, VB, alphaA, alphaB, lambda)`

The inputs to the functions are described above, and the outputs of the functions should be the associative strength(s) (`V`=V_{new}; `VA`=$V_{A,new}$; `VB`=$V_{B,new}$) after one trial of Rescorla-Wagner updating. If you keep these functions separate from the functions below (see below for alternatives), store each function as a separate .m file, where the name of the .m file corresponds to the name of the function, and each function should run without errors when given the described inputs.
NOTE: Running the function file itself (e.g., `rwRule.m`) will yield an error. This is fine.

Once you've made these functions, use them to build two functions (1.5 points each, 3 total):

1 CS: `function VVect=rw(nTrials,V,alpha,lambda)`
2 CS's: `function [VVectA,VVectB]=rwAB(nTrials,VA,VB,alphaA,alphaB,lambda)`

The new argument `nTrials` is the number of trials of Rescorla-Wagner updating to perform. In these functions, use a `for` loop to execute `nTrials` trials of Rescorla-Wagner. Store the `nTrials+1` values of V (1 and 2) or V_A and V_B (3-6), including the initial value and `nTrials` updated values in order, in the output vector(s) `VVect` (1 and 2) or `VVectA` and `VVectB` (3-6).

Organization of functions is up to you! **All get full credit!** Options are:

- Put all calculations in `rw.m` and `rwAB.m`
  - **NOTE**: You won't have functions named `rwRule` or `rwABRule`. This is fine.
- **RECOMMENDED**: Use separate function files `rwRule.m` and `rwABRule.m` for the updating rule; use these functions in `rw.m` and `rwAB.m` to run multiple trials
  - If you do this, please include those extra function files with your submission!
- Put `rwRule` and `rwABRule` as *local functions* in `rw.m` and `rwAB.m` (see end of W3D2 lecture)
  - **NOTE**: You won't be able to run `rwRule` or `rwABRule` from the console. This is fine.

Part 2: Main Script

For each prediction, listed below, you will produce the predicted value of associative strength V after each of 10 trials (for a total of 11 values of V including the starting value(s)) and graph the resulting values, reproducing the graph in the corresponding slide from W5D1 lecture. Store your script as a single .m file named `pj2_<YOUR STUDENT ID>.m` (replace `<YOUR STUDENT ID>` with your own 9-digit student ID number). Upload this file to CCLE along with the aforementioned functions as your project submission. Your entire .m file should run without errors. The script should begin with the command `clear` and, when it completes, all variables listed at the end of this document should be in the workspace.

Each numbered prediction (**1. Acquisition** to **6. Overexpectation**) from the lecture slides contains a set of starting values for the Rescorla-Wagner model. Use these values and the `rw` and `rwAB` functions to execute 10 trials of Rescorla-Wagner updating. Use the `rw` and `rwAB`

functions to generate the 11 values of V (1 and 2) or V_A and V_B (3-6), including the initial value and the 10 updated values in order, and store them in separate vectors named `VVect#` (1 and 2) or `VVectA#` and `VVectB#` (3-6), replacing `#` with the prediction number. Each such vector is worth 1 point (1 and 2) or 0.5 points (3-6) for a total of 6 points. See the end of this document for a complete list of required variables.

Each graph is worth 0.5 points for a total of 3 points. These graphs should match those from the W5D1 lecture slides exactly. You don't need to save the graphs, only include code in your main script that produces them. Thus, if you don't do the extra credit, your script should open 6 figure windows, each of which has one of the desired graphs.

Predictions of Rescorla-Wagner

1. **Acquisition**: In classical conditioning, acquisition is the learning of the conditioned response to the CS when it is paired with the US (V=0; α=.5; λ=1)
2. **Extinction**: In classical conditioning, extinction refers to the reduction, and ultimately elimination, of the response to the CS when it is no longer paired with the US (V=1; α=.5; λ=0)
3. **Compound conditioning**: In compound conditioning, two or more CSs are presented together (compound CS). They both obtain some associative strength, but only the compound CS reaches the maximum (V_A=V_B=0; α_A=α_B=.3; λ=1)
4. **Overshadowing**: If the learning rates for the components of a compound CS are different, then the component with the higher learning rate ends up with a higher associative strength than the other (V_A=V_B=0; α_A=.3; α_B=.1; λ=1)
5. **Blocking**: When one of the components of a compound CS begins already fully conditioned, and the other is not conditioned at all, their associative strengths stay constant (V_A=1; V_B=0; α_A=α_B=.3; λ=1)
6. **Overexpectation**: If two CS's are both fully conditioned, and then are presented together, the resulting response is stronger than that elicited by a single fully conditioned CS (V_A=V_B=1; α_A=α_B=.3; λ=1)

Extra credit: Probabilistic Reinforcement

Rescorla-Wagner was intended as a starting point for more realistic descriptions of situations and behavior. In one situation, whether an agent (say, a dog or human) engages with a stimulus is probabilistic, and the probability depends on the current association strength:

- One CS: $p = \varepsilon + \big((1 - 2\varepsilon) * V\big)$
- Two CS's: $p_A = \varepsilon + \big((1 - 2\varepsilon) * V_A\big)$; $p_B = \varepsilon + \big((1 - 2\varepsilon) * V_B\big)$

Here, the probability of encountering the stimulus is given by p; if the stimulus is encountered, the Rescorla-Wagner rule kicks in and the association strength is updated; if the stimulus is not encountered, nothing happens. I use terms like "encountering" and "abstaining", but the model is truly describing whether the Rescorla-Wagner rule kicks in or not, and this random component can represent a random *reality* or a random *reinforcement process in the brain*. For example, a dog can encode the association between a bell and the smell of meat on one trial but not another trial, determined by a random process such as described here, and thus different dogs may ultimately learn the association at different rates.

The extra term ε is the chance of trying a novel stimulus or of not encountering a conditioned stimulus; higher values indicate that the reinforcement process (within or outside of the brain) is more random, while lower values indicate that the reinforcement process is determined more by

the association strength(s). This term is also needed to avoid a situation where a stimulus has no association strength *and* the probability of reinforcement depends on association strength, such that reinforcement never occurs (i.e., $\varepsilon = 0$ always leads to no learning).

Extra Credit Functions

For the extra credit, edit copies of `rw.m` and `rwAB.m` to create two new functions called `pr.m` (.5 points) and `prAB.m` (1 point) in their own function files named `pr.m` and `prAB.m`, respectively, which has the same inputs as `rwAB` and, like `rwAB`, plus a new, final input `eps` corresponding to ε. The function `pr.m` should return a vector `VVect`, just like `rw.m`, while `prAB.m` should return two vectors `VVectA` and `VVectB` , just like `rwAB.m`. These vectors will be vectors of association strengths after applying the probabilistic reinforcement rule. Your functions will not store whether stimuli were actually encountered on each trial; this is fine.

Extra Credit Main Code

Reproduce Predictions 1 and 3 from the main project using the probabilistic reinforcement rule. **Use an epsilon value of $\boldsymbol{\varepsilon = 0.1}$**. This code should appear after the Rescorla-Wagner code in the same .m-file; thus, if you do the extra credit graphs in addition to the main project graphs, your main code will produce a total of 8 graphs, where the first six correspond to Rescorla-Wagner and the last two correspond to probabilistic reinforcement. Use the same starting values for the extra credit as you did for the main project. **Use 50 trials of conditioning for probabilistic reinforcement.**

The reinforcement process is random, so to properly represent its predictions for behavior, you will need to plot multiple `VVect`'s in the same graph for both Prediction 1 (.5 points) and Prediction 3 (1 point). Excluding white (which wouldn't show up), there are seven plotting colors you can refer to by name in MATLAB: `"yellow" "magenta" "cyan" "red" "green" "blue" "black"`. Therefore, you will plot the results of seven probabilistic Rescorla-Wagner simulation in each graph, one with each of these colors. You do not need to store the results of your simulation (`VVect`'s) anywhere in particular, you only need to use them to make the graphs. For Prediction 3, the two lines can be of the same line type or of different types as in the main project, but for each simulation (i.e., generation of a `VVect` or pair of `VVect`'s), the lines must be of the same color.

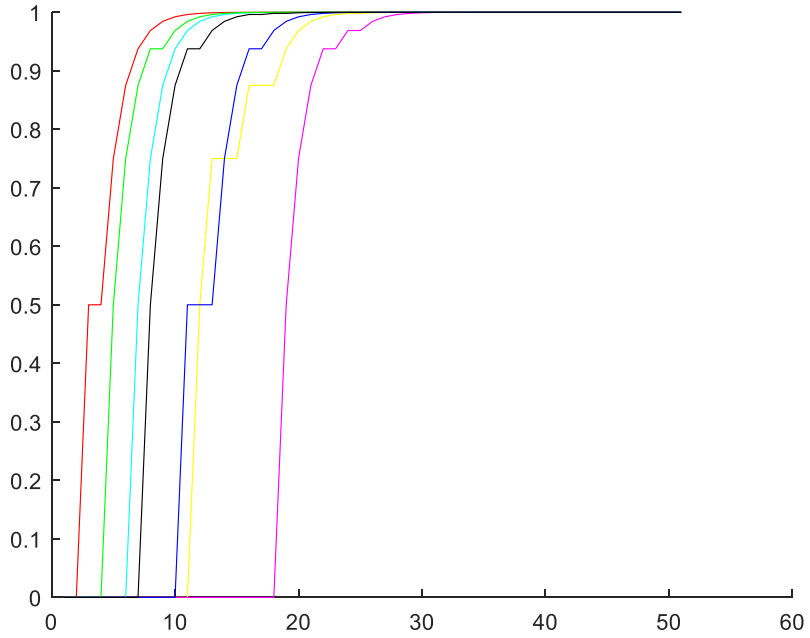
The next page provides an example graph for Prediction 1 and Prediction 3. Note that your graphs may look different from mine if different random numbers were generated (this is fine).

Observe two things: first, with Prediction 1, a random reinforcement process leads to different times to complete conditions for the different agents (e.g., dogs). This is one way to model between-individual variability, which we observe in nearly any study of behavior.

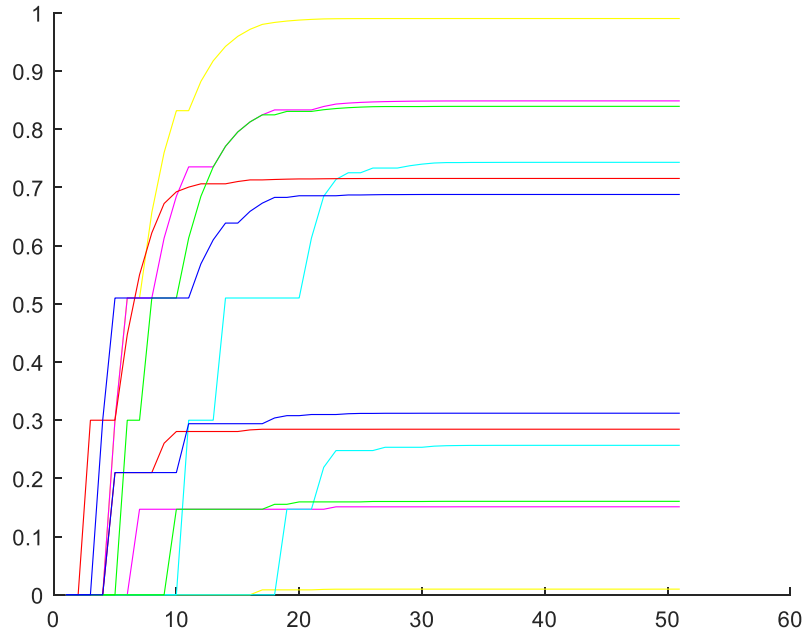
Second, with Prediction 3, even though the two stimuli have the same initial association strength and the same learning rate, their final association strengths depend on chance. Sometimes (e.g., the yellow lines), only one association is truly learned and the other CS is ignored. Other times (e.g., the blue lines), both associations are learned, but to differing degrees.

CONTINUED ON NEXT PAGE

PREDICTION 1



PREDICTION 3



Required Variables and Functions

Upon completion, your workspace should contain the following variables

`VVect1`	`VVectA3`	`VVectA4`	`VVectA5`	`VVectA6`
`VVect2`	`VVectB3`	`VVectB4`	`VVectB5`	`VVectB6`

and produce six figures in separate windows as described above (**eight with the extra credit included**). In addition, submit the following functions as separate .m files (bold = extra credit):

rw.m rwAB.m **pr.m** **prAB.m**