Modeling Learning with R

Paul Kieffaber

2025-10-09

In this chapter we distinguish *natural-language models*—verbal theories stated in everyday words—from *formal models*—the same ideas written as equations or code that yield numerical predictions. A natural-language model might say: "People raise their expectation after rewards, especially when the outcome is surprising, and their expectations are reduced when outcomes are expected." This is intuitive but leaves magnitudes and rules unspecified. A formal model makes those commitments explicit. For example, a minimal learning model is the delta rule:

$$V_{t+1} = V_t + \alpha(O_t - V_t)$$

where:

- V_t is the current expectation about the outcome.
- $O_t \in \{0, 1\}$ is the actual outcome.
- $\alpha \in (0, 1)$ is the learning rate.

Why cognitive models?

Cognitive models are formal, quantitative theories of mental processes. They specify:

- Inputs: what information the person receives on each trial (stimuli, feedback).
- Latent states: internal variables that are not directly observed (e.g., beliefs, expectations).
- Parameters: psychologically meaningful constants (e.g., a learning rate).
- Update rules: how latent states change over time as new information arrives.
- Observation model: how latent states generate observable data (choices, reaction times, accuracy).
- Predictions: trial-by-trial forecasts that can be compared to data.

Modeling in this way is central to theory building in psychology because it forces clarity about mechanisms. Natural language theories often admit multiple interpretations; a cognitive model translates assumptions into equations that make testable, falsifiable predictions. This lets us:

- Disambiguate competing explanations. Two verbal theories can predict opposite learning curves once formalized.
- Quantify mechanisms. Parameters (e.g., a learning rate) become estimates we can compare across people, groups, or conditions.
- Accumulate evidence. Models can be fit, compared, and refined using principled criteria (likelihood, cross-validation), enabling cumulative science rather than one-off findings.
- Bridge levels of analysis. The same model can link behavior to neural signals (e.g., prediction errors), or to individual differences.
- Generalize. Once specified, a model can be simulated under new tasks or interventions to generate new hypotheses.

Modeling Learning with R

In this chapter, we investigate theories of learning in Psychology. Our goal is to implement two foundational models of learning, (1) the Rescorla-Wagner model and (2) a Bayesian learning model—using R. By simulating these models, we can develop a much deeper intuition for how they work, what psychological phenomena they explain, and how they differ in their assumptions about the nature of learning.

We will start with error-driven learning, where surprise is the engine of change, and then move to belief-updating, where the mind is modeled as a statistician collecting and weighing evidence.

Part 1: Error-Driven Learning: The Rescorla-Wagner Model

The Rescorla-Wagner (R-W) model is arguably the most influential formal model of associative learning. Its central premise is simple but powerful: learning only happens when an outcome is surprising. The greater the surprise, the greater the learning.

The model formalizes the relationship between a set of cues (Conditioned Stimuli, CS) and an outcome (Unconditioned Stimulus, US). It tracks the "associative strength" (V) for each cue, which represents how strongly that cue predicts the outcome.

The R-W model elegantly explains several learning phenomena:

1. Acquisition: As a cue is repeatedly paired with an outcome, the associative strength grows, forming a classic learning curve.
2. Extinction: If the cue is later presented without the outcome, the expectation is violated (a "negative" surprise), and the associative strength gradually decreases.
3. Blocking: This is a critical prediction that sets the R-W model apart from simpler theories. If one cue (CS1, a light) already perfectly predicts an outcome, and then a new cue (CS2, a bell) is presented alongside the first cue, the organism learns very little about the new cue. In terms of the R-M model, the outcome is not surprising, so no new learning occurs. The "blocking" effect is one of the most robust and important phenomena in learning theory, and it has been demonstrated consistently in both animals and humans since the 1960s.

The Math and its Meaning The engine of the model is the Rescorla-Wagner equation, which calculates the change in associative strength (ΔV) for a single cue on a single trial:

$$\Delta V = \alpha * \beta * (\lambda - \sum V)$$

Let's translate each term into its psychological meaning:

ΔV: The change in learning. This is what we want to calculate.

α (alpha): The salience of the cue. A loud bell (high α) will be learned about more quickly than a dim light (low α).

β (beta): The learning rate associated with the outcome. A highly desirable piece of food (high β) will drive faster learning than a less desirable one.

λ (lambda): The magnitude of the outcome that actually occurred. If food is present, λ might be 1; if absent, it's 0.

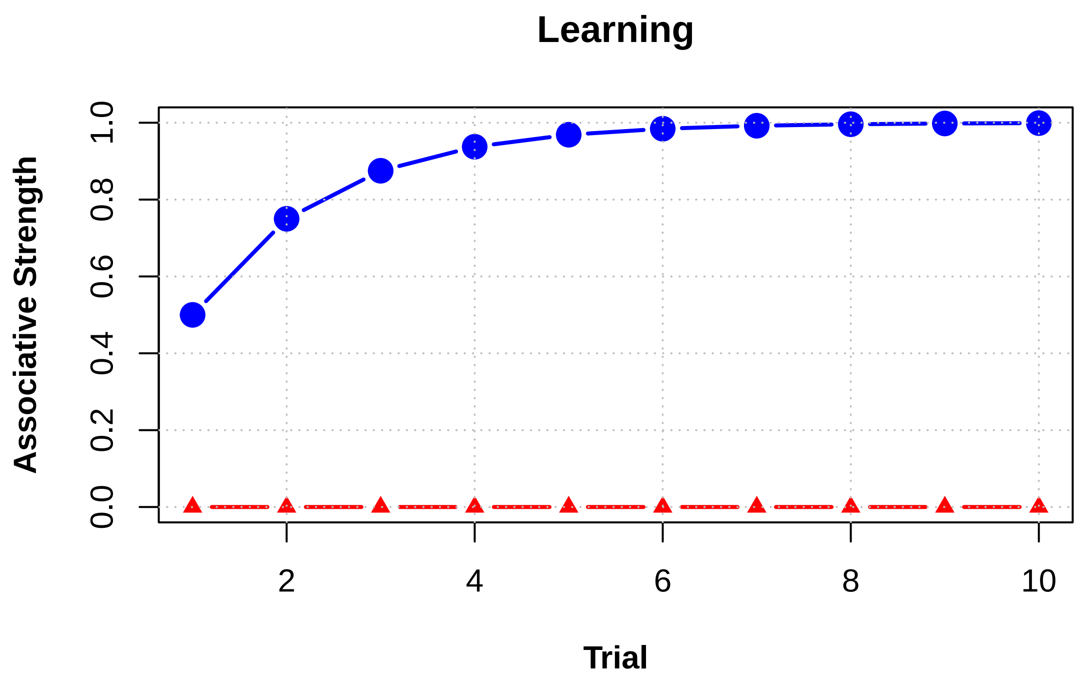
$\sum V$ (sigma V): The total expectation. This is the sum of the associative strengths of all cues present on a trial.

$(\lambda - \sum V)$: The Prediction Error or "Surprise". This is the heart of the model. It's the difference between what actually happened (λ) and what the organism expected to happen ($\sum V$).

Implementation in R: Simulating the "Blocking" Effect Let's simulate the classic blocking experiment. We will have two cues (a light and a bell) and one outcome (food).

Phase 1: Train the model by pairing the light with food for 10 trials.

```r
# --- Establish Model Parameters ---
alpha_light <- 0.5  # Salience of the light cue
alpha_bell <- 0.5   # Salience of the bell cue
beta <- 1.0         # Learning rate for the food outcome

# --- Experiment Setup ---
V_light <- 0        # Associative strength for the light
V_bell <- 0         # Associative strength for the bell

# 10 Trials of Light and Food present
Trials <- data.frame(
  Light = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
  Bell  = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  Food  = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
)

# Containers to Store results for plotting
light_strength_history <- c()
bell_strength_history <- c()

# --- Phase 1: Train the Light Alone ---
for (trial in 1:nrow(Trials)) {
  # Determine which stimuli are present
  light_present<-Trials$Light[trial]
  bell_present<-Trials$Bell[trial]
  # Determine whether food was present
  lambda<-Trials$Food[trial]
  # Expectation is the sum of strengths ONLY for PRESENT cues
  total_expectation <- (V_light * light_present) + (V_bell * bell_present)

  # Calculate the "surprise" or prediction error
  prediction_error <- lambda - total_expectation

  # Update light strength
  delta_V_light <- alpha_light * beta * prediction_error * light_present
  V_light <- V_light + delta_V_light
  # Update bell strength
  delta_V_bell <- alpha_bell * beta * prediction_error * bell_present
  V_bell <- V_bell + delta_V_bell

  # Store current strengths
  light_strength_history <- c(light_strength_history, V_light)
  bell_strength_history <- c(bell_strength_history, V_bell)
}

# --- Plot the Learning ---
plot(light_strength_history,
     type = "b",              # "b" for "both" line and points
     main = "Learning", # Title of the plot
     xlab = "Trial", # X-axis label
     ylab = "Associative Strength",# Y-axis label
     col = "blue",            # Color of the line and points
     lwd = 2,                 # Line width (thickness)
     pch = 19,                # Plotting character (19 is a solid circle)
     cex = 1.5,               # Character expansion (size of the points)
     font.lab = 2,             # Font for labels (2 is bold)
     ylim=c(0,1)
)
lines(bell_strength_history,
      type = "b",             # Also show both line and points
      col = "red",              # A different color for the second line
      lwd = 2,
      pch = 17                # A different point character (triangle)
)

# 2. Add the custom grid
grid(col = "gray", lty = "dotted")
```

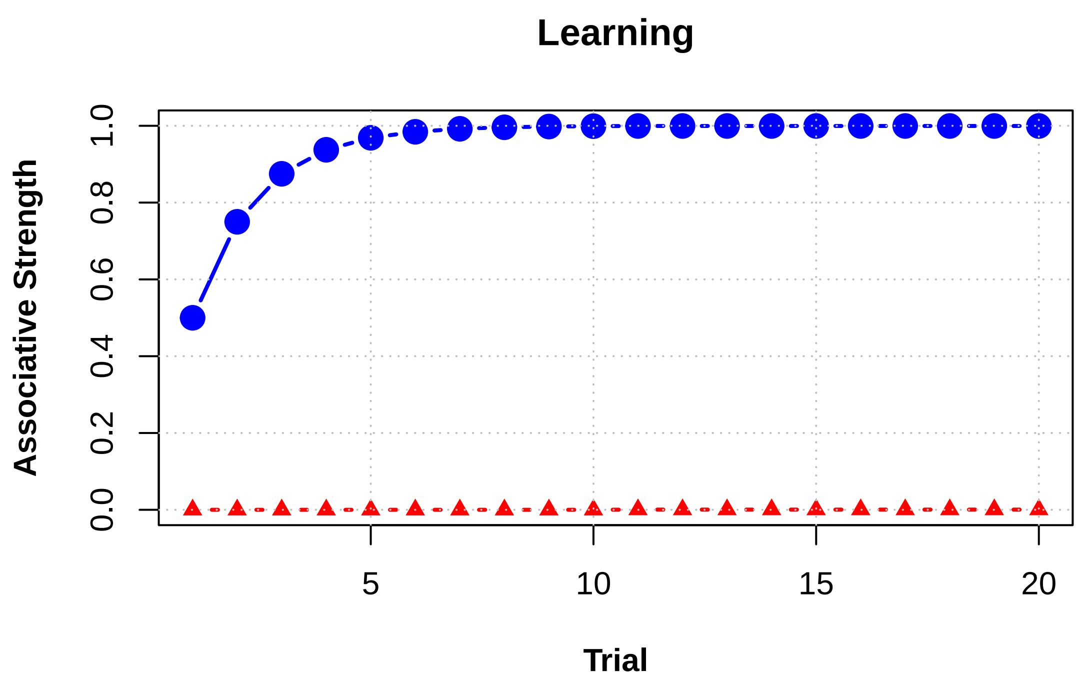


Phase 2: Add the Bell Stimulus

Now that the model has “learned” that the light predicts food, let’s try presenting a second stimulus -the bell- along with the light.

```
# Note that model parameters have already been established above

# 10 Trials of Bell, Light and Food present
Trials <- data.frame(
  Light = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
  Bell  = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
  Food  = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
)

# Containers to Store results for plotting
#light_strength_history <- c()
#bell_strength_history <- c()
```

```r
# --- Phase 1: Train the Light Alone ---
for (trial in 1:nrow(Trials)) {
  # Determine which stimuli are present
  light_present<-Trials$Light[trial]
  bell_present<-Trials$Bell[trial]
  # Determine whether food was present
  lambda<-Trials$Food[trial]
  # Expectation is the sum of strengths ONLY for PRESENT cues
  total_expectation <- (V_light * light_present) + (V_bell * bell_present)

  # Calculate the "surprise" or prediction error
  prediction_error <- lambda - total_expectation

  # Update light strength
  delta_V_light <- alpha_light * beta * prediction_error * light_present
  V_light <- V_light + delta_V_light
  # Update bell strength
  delta_V_bell <- alpha_bell * beta * prediction_error * bell_present
  V_bell <- V_bell + delta_V_bell

  # Store current strengths
  light_strength_history <- c(light_strength_history, V_light)
  bell_strength_history <- c(bell_strength_history, V_bell)
}

# --- Plot the Learning ---
plot(light_strength_history,
     type = "b",               # "b" for "both" line and points
     main = "Learning", # Title of the plot
     xlab = "Trial", # X-axis label
     ylab = "Associative Strength",# Y-axis label
     col = "blue",             # Color of the line and points
     lwd = 2,                  # Line width (thickness)
     pch = 19,                 # Plotting character (19 is a solid circle)
     cex = 1.5,                # Character expansion (size of the points)
     font.lab = 2,              # Font for labels (2 is bold)
     ylim=c(0,1)
)
lines(bell_strength_history,
      type = "b",              # Also show both line and points
      col = "red",               # A different color for the second line
      lwd = 2,
      pch = 17                 # A different point character (triangle)
)

# 2. Add the custom grid
grid(col = "gray", lty = "dotted")
```

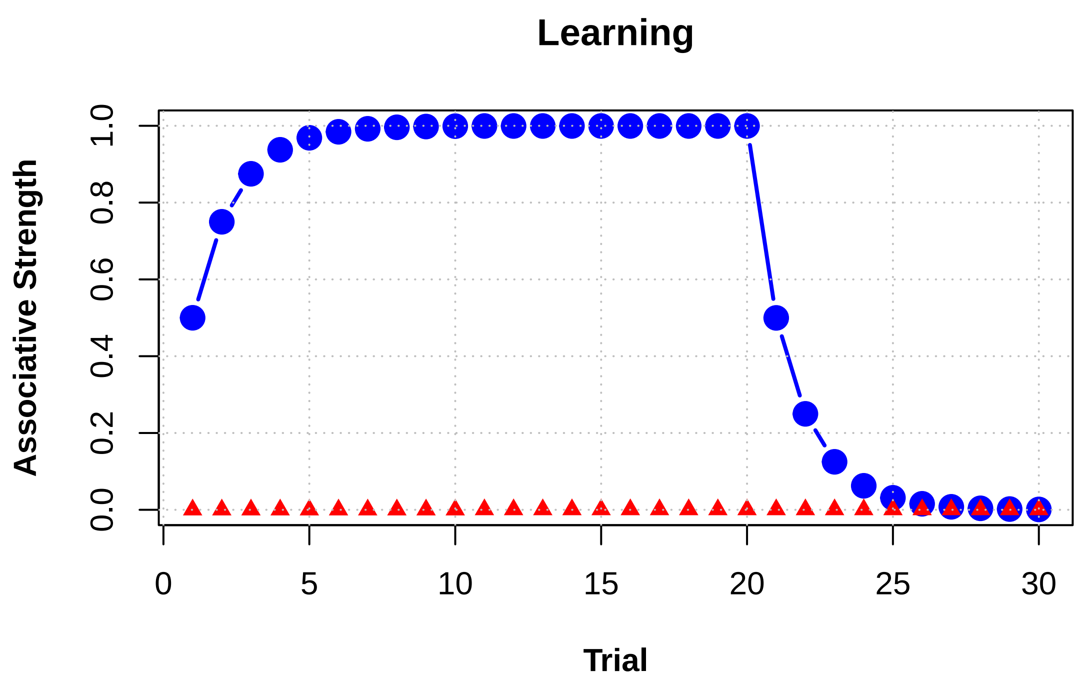


After ten more trials with both the light and the bell present, you can see that the association strength for the light stimulus reached asymptote because once it perfectly predicted the food, thus there was no prediction error and no further learning. The light already perfectly predicts the food, so the outcome is no longer surprising, and learning about the bell is blocked.

Extinguishing Learning

Now let's see what happens when the light no longer predicts the food. You will see that as the learning about the light is extinguished, the model rapidly "forgets" the association between the light and the food.

```r
# Note that model parameters have already been established above

# 10 Trials of Bell, Light and Food present
Trials <- data.frame(
  Light = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
  Bell  = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  Food  = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
)

# Containers to Store results for plotting
#light_strength_history <- c()
#bell_strength_history <- c()

# --- Phase 1: Train the Light Alone ---
for (trial in 1:nrow(Trials)) {
  # Determine which stimuli are present
  light_present<-Trials$Light[trial]
  bell_present<-Trials$Bell[trial]
  # Determine whether food was present
  lambda<-Trials$Food[trial]
  # Expectation is the sum of strengths ONLY for PRESENT cues
  total_expectation <- (V_light * light_present) + (V_bell * bell_present)
  # Calculate the "surprise" or prediction error
  prediction_error <- lambda - total_expectation

  # Update light strength
  delta_V_light <- alpha_light * beta * prediction_error * light_present
  V_light <- V_light + delta_V_light
  # Update bell strength
  delta_V_bell <- alpha_bell * beta * prediction_error * bell_present
  V_bell <- V_bell + delta_V_bell

  # Store current strengths
  light_strength_history <- c(light_strength_history, V_light)
  bell_strength_history <- c(bell_strength_history, V_bell)
}

# --- Plot the Learning ---
plot(light_strength_history,
     type = "b",              # "b" for "both" line and points
     main = "Learning", # Title of the plot
     xlab = "Trial", # X-axis label
     ylab = "Associative Strength",# Y-axis label
     col = "blue",            # Color of the line and points
     lwd = 2,                 # Line width (thickness)
     pch = 19,                # Plotting character (19 is a solid circle)
     cex = 1.5,               # Character expansion (size of the points)
     font.lab = 2,             # Font for labels (2 is bold)
     ylim=c(0,1)
)
lines(bell_strength_history,
      type = "b",             # Also show both line and points
      col = "red",              # A different color for the second line
      lwd = 2,
      pch = 17                # A different point character (triangle)
)

# 2. Add the custom grid
grid(col = "gray", lty = "dotted")
```



Belief Updating: An Introduction to Bayesian Learning

Bayesian models offer a fundamentally different perspective on learning. Instead of tracking a single value like associative strength, The Bayesian model tracks its degree of "belief" about the world in the form of a probability distribution. Learning in this context is the process of updating these beliefs when new evidence is collected.

The core of Bayesian inference lies in combining two sources of information:

1. Prior Belief: What you believe about the world before observing new data. This can be strong (based on years of experience) or weak (a complete guess).
2. Likelihood: How likely is your new observation, given your current belief? This is the new evidence.

These two are combined via Bayes' rule to produce a *Posterior Belief* –an updated belief that integrates your prior belief with the new evidence. A key advantage of this approach is that it models not just what we believe, but also our certainty in that belief.

Bayes' Rule: The Engine of Belief Updating

The mathematical engine that combines the prior and the likelihood is a simple but profound formula known as Bayes' Rule. In its proportional form, it states that the posterior belief is proportional to the prior belief multiplied by the likelihood.

The full formula is:

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

Where:

$P(H|E)$ - The Posterior: The probability of our Hypothesis (H) being true, given the Evidence (E) we just observed. This is what we want to calculate—our updated belief distribution.

$P(E|H)$ - The Likelihood: The probability of observing the Evidence if our Hypothesis were true. This is the part that connects our belief to the actual data.

$P(H)$ - The Prior: The probability of our Hypothesis being true before we saw any new evidence. This is our starting belief.

$P(E)$ - The Marginal Likelihood: The overall probability of observing the Evidence, averaged over all possible hypotheses. This is a normalizing constant that ensures the final posterior probabilities sum to 1.

Implementation in R: The Beta-Binomial Model

Because it is a little more complex than the R-W model, let's start with a simple example of Bayesian learning. The classic example of Bayesian learning is flipping a coin and trying to figure out if it's fair. We want to estimate its "bias," or the probability (θ) that it will land on heads. If you believe that probability is 0.5, then you believe that the coin is completely unbiased. However, if you believe that probability is 0.75, then you believe that the coin is biased toward landing on heads.

From the Formula to the Simple Rule: Conjugate Priors

Our belief about the coin's bias can be represented by a Beta probability distribution. This distribution is flexible and is defined by two shape parameters, a and b. A flat Beta(1, 1) represents a totally uncertain prior, while a sharply peaked Beta(100, 100) represents a very strong belief that the coin is fair.

The new evidence (the data from coin flips) can be described by a Binomial distribution, which gives the probability of getting k heads in n flips.

Fortunately, the Beta distribution and the Binomial distribution are what is known as a "conjugate pair." This means that when you combine a Beta prior distribution with Binomial data using the full Bayes' Rule formula, the complex multiplication and division simplifies dramatically. The math works out such that the posterior is guaranteed to be another Beta distribution. This allows us to bypass the complex formula and use an incredibly simple update rule:

- heads_posterior = heads_prior + number_of_heads
- tails_posterior = tails_prior + number_of_tails

This simple addition is the direct result of applying Bayes' Rule to this special pair of distributions. It's a powerful shortcut that makes Bayesian updating computationally efficient and intuitive. Let's see how this kind of Bayesian updtating can be implemented in R.

The Classic Biased Coin Example

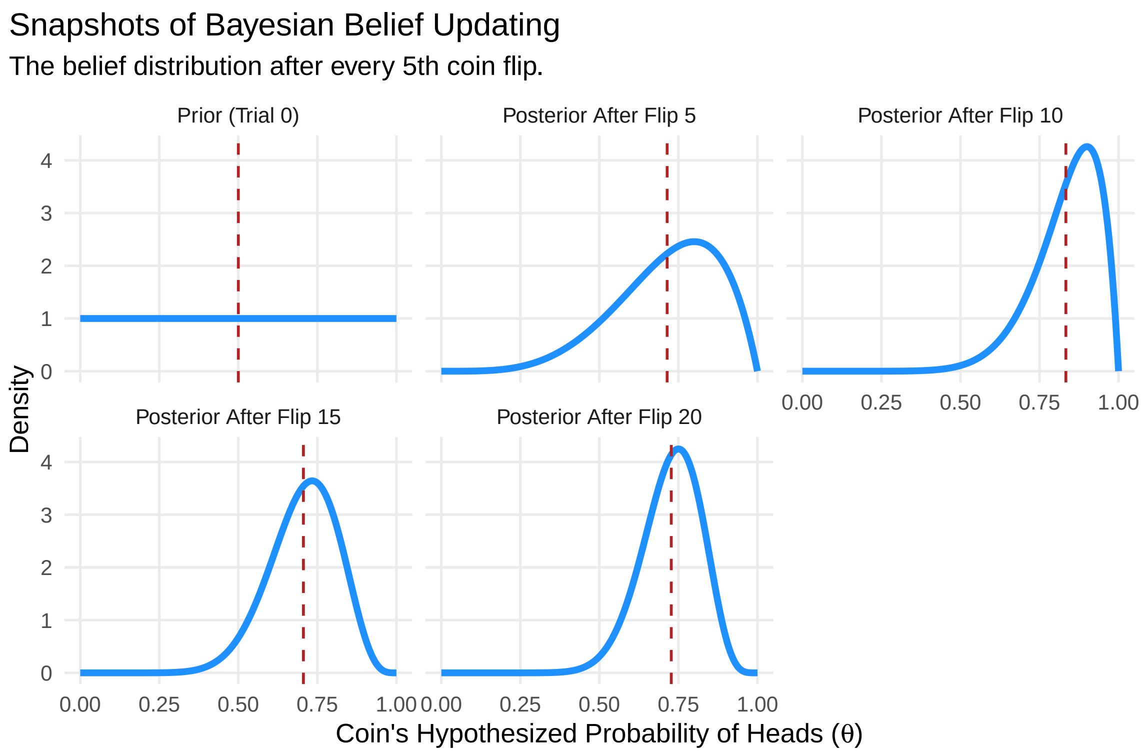
In order to demonstrate how beliefs are updated in the Bayesian model, we will present it with some data. Specifically, we will present the model with the results of 20 flips of the coin. For the purpose of illustration, we will make it so that the coin lands on heads 15 of the 20 times it was flipped.

- Think aboout it: How would your "beliefs" change if you had a coin that landed on heads 75% of the time

```r
# --- 1. Define the Prior Belief & Experiment ---
# We start with an uninformative prior, Beta(1,1).
initial_heads <- 1
initial_tails <- 1

# Create the same reproducible sequence of 20 flips with 15 heads.
set.seed(123) # for reproducibility
outcomes <- c(rep("Heads", 15), rep("Tails", 5))
coin_flips <- sample(outcomes) # Shuffle the outcomes

# --- 2. Loop Through Data and Record Belief History ---
# Generate the belief state after every single flip and store it.
history_list <- list()
```

```r
current_heads <- initial_heads
current_tails <- initial_tails

for (i in 1:length(coin_flips)) {
  outcome <- coin_flips[i]
  if (outcome == "Heads") {
    current_heads <- current_heads + 1
  } else {
    current_tails <- current_tails + 1
  }

  history_list[[i]] <- data.frame(
    Flip_Number = i,
    nHeads = current_heads,
    nTails = current_tails
  )
}
# Combine the list into one data frame
history_data <- bind_rows(history_list)

# --- 3. Prepare Data for Faceted Plot ---
# We want to show the state at the beginning (Prior) and after every 5th trial.

# First, create the data for the Prior distribution (Flip 0)
theta <- seq(0, 1, length.out = 200)
prior_data <- data.frame(
  Flip_Number = 0,
  nHeads = initial_heads,
  nTails = initial_tails
)

# Filter the history to get flips 5, 10, 15, and 20
snapshots_data <- history_data %>%
  filter(Flip_Number %in% c(5, 10, 15, 20))

# Combine the prior with the snapshots
facet_data_prep <- bind_rows(prior_data, snapshots_data)

# Now, create the full data needed for plotting each curve
# and add a clean label for each facet panel.
plot_data <- facet_data_prep %>%
  rowwise() %>%
  mutate(
    dist = list(data.frame(
      theta = theta,
      density = dbeta(theta, nHeads, nTails)
    )),
    Label = ifelse(Flip_Number == 0,
                   "Prior (Trial 0)",
                   paste0("Posterior After Flip ", Flip_Number))
  ) %>%
  unnest(dist)
#Force the reordering of $Label levels for plotting
```

```
plot_data$Label<-factor(plot_data$Label, levels=(c("Prior (Trial 0)", "Posterior After Flip 5", "Poster

# --- 4. Create the Faceted Plot ---
# Today's date is Thursday, October 9, 2025.
ggplot(plot_data, aes(x = theta, y = density)) +
  geom_line(color = "dodgerblue", linewidth = 1.2) +
  # Add a vertical line to show the mean (our "best guess") of the distribution (a/a+b=mean of beta dis
  geom_vline(aes(xintercept = nHeads / (nHeads + nTails)), color = "firebrick", linetype = "dashed") +
  # This is the key function to create the grid of plots
  facet_wrap(~Label, ncol = 3) +
  labs(
    title = "Snapshots of Bayesian Belief Updating",
    subtitle = "The belief distribution after every 5th coin flip.",
    x = expression(paste("Coin's Hypothesized Probability of Heads (",theta,")")),
    y = "Density"
  ) +
  theme_minimal(base_size = 10) +
  theme(plot.title.position = "plot", # Align title to the left
        panel.grid.minor = element_blank())
```



The resulting plots illustrate how the posterior beliefs change over time. Initially, the "Prior" is a flat line, indicating the belief that any bias is equally likely. After 5 trials, the "posterior" has begun to develop a peak, indicating a developing belief that the probability of getting heads when flipping the coin may be higher but the distribution is wide, indicating a lot of uncertainty about this belief. However, after 20 trials, the posterior is a sharp peak centered around 0.75 (15 heads / 20 flips). In other words, the model learned from the data, updating its belief over time and becoming much more certain about the coin's true bias.

Bayesian Classical Conditioning

Now that we've had a chance to understand the basic premise of Bayesian belief updating, let's consider the same classical conditioning situation that was learned by the Rescorla-Wagner model in which a light and/or bell is paird with food. In the previous example we only needed to model the belief about whether or not the coin flip would be heads. This time, we need to model two beliefs, (1) the probability that food will appear following the light and (2) the probability that food will appear following the bell.

This model will be exposed to the exact same experimental setup as the Rescorla-Wagner model. Namely, three blocks of ten trials. The first ten "acquisition" trials are when the light is paired with the food. In the second ten "compound" trials, the bell is added and food is paired with presentation of both stimuli (i.e., light+bell). Finally, in the last ten "extinction" trials, only the light is presented, it is not paired with the presentation of food.

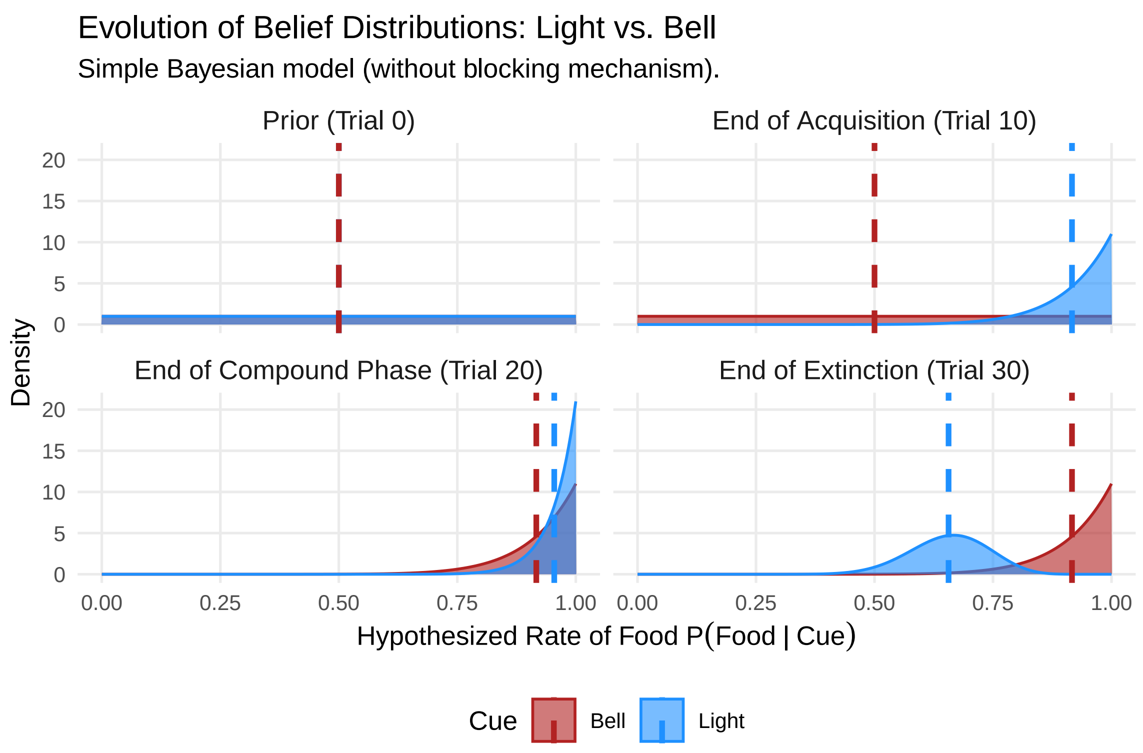
Experimental Setup

```
# --- 1. Define the Priors (The "Prior" beliefs State) ---
# The model starts with a completely open mind (uninformative prior).
# We use a Beta(1,1) distribution, which is a flat line representing total uncertainty.
# We need a separate belief for each cue.
# 'a' represents "counts" of Food, 'b' represents "counts" of No Food.
a_light <- 1
b_light <- 1

a_bell <- 1
b_bell <- 1

# --- 2. Design the Experiment (The "Data") ---
# We use the EXACT same experimental plan as the Rescorla-Wagner model for direct comparison but with a
phase1 <- data.frame(Trial = 1:10, Phase = "1. Acquisition", Light = 1, Bell = 0, Food = 1)
phase2 <- data.frame(Trial = 11:20, Phase = "2. Blocking", Light = 1, Bell = 1, Food = 1)
phase3 <- data.frame(Trial = 21:30, Phase = "3. Extinction", Light = 1, Bell = 0, Food = 0)
experiment_plan <- bind_rows(phase1, phase2, phase3)
```

Running the simulation in R

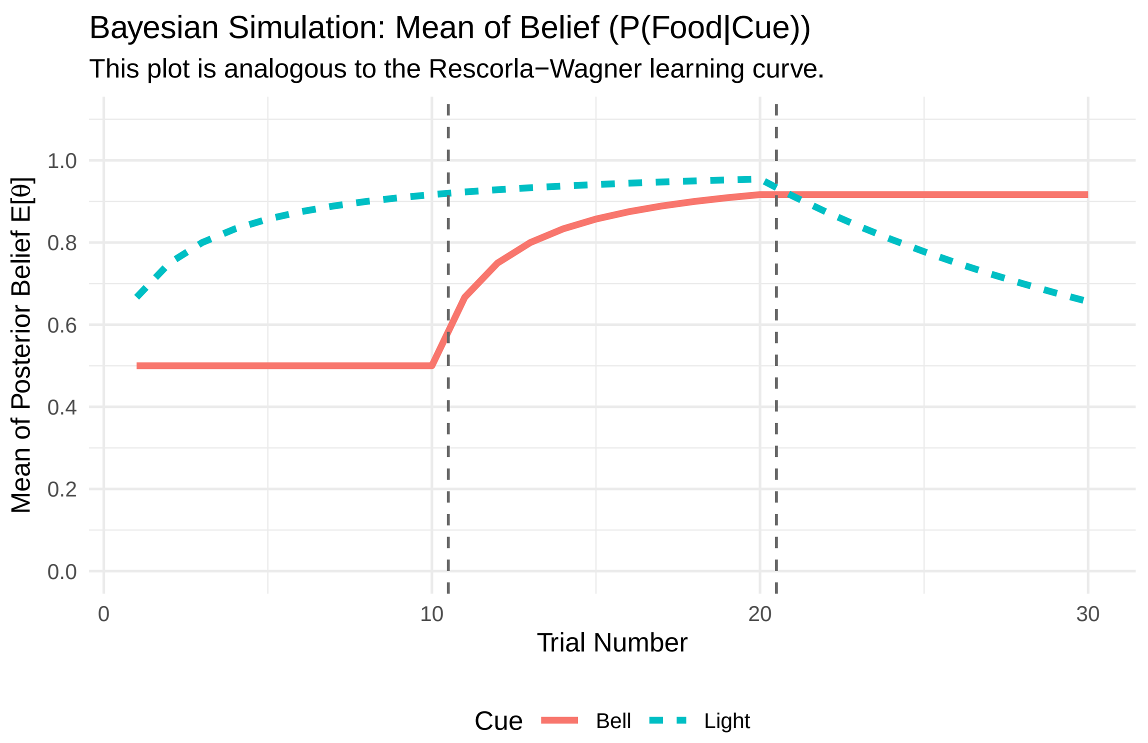
```
# --- 3. Run the Simulation (The "Update" Process) ---
# We loop through each trial and update the parameters of our Beta distributions.
history <- data.frame()

for (i in 1:nrow(experiment_plan)) {
  current_trial <- experiment_plan[i, ]
  light_present <- current_trial$Light
  bell_present <- current_trial$Bell
  food_present <- current_trial$Food

  # Bayesian update depends on which cues are present
  if (light_present == 1 && bell_present == 0) { # Light alone
    a_light <- a_light + food_present
    b_light <- b_light + (1 - food_present)
  } else if (light_present == 0 && bell_present == 1) { # Bell alone (not in this experiment)
    a_bell <- a_bell + food_present
    b_bell <- b_bell + (1 - food_present)
  } else if (light_present == 1 && bell_present == 1) { # Compound trial (Light + Bell)
    a_light <- a_light + food_present
    b_light <- b_light + (1 - food_present)
```

```r
    a_bell <- a_bell + food_present
    b_bell <- b_bell + (1 - food_present)
  }

  # Store the state (both the parameters and the mean of the belief)
  trial_history <- data.frame(
    Trial = current_trial$Trial,
    Phase = current_trial$Phase,
    a_light = a_light, b_light = b_light, mean_light = a_light / (a_light + b_light),
    a_bell = a_bell, b_bell = b_bell, mean_bell = a_bell / (a_bell + b_bell)
  )
  history <- bind_rows(history, trial_history)
}

# ---  Visualize the Full Belief Distributions  ---

# We need to define the prior state for Trial 0
initial_a <- 1
initial_b <- 1

# First, get the state at the end of each phase (Trials 10, 20, 30)
snapshots_df <- history %>%
  filter(Trial %in% c(10, 20, 30))

# Next, create the data for the Prior state (Trial 0)
prior_df <- tibble(
  Trial = 0,
  Phase = "0. Prior",
  a_light = initial_a, b_light = initial_b,
  a_bell = initial_a, b_bell = initial_b
)

# --- THIS DATA PREP SECTION IS THE FIX ---

# Combine the prior with the snapshots
plot_prep_df <- bind_rows(prior_df, snapshots_df)

# STEP 1: Reshape the data from "wide" to "long" format FIRST.
# This creates our clean 'Cue', 'a', and 'b' columns which will be preserved.
tidy_beliefs_df <- plot_prep_df %>%
  pivot_longer(
    cols = c(a_light, b_light, a_bell, b_bell),
    names_to = c(".value", "Cue"), # .value creates 'a' and 'b' columns
    names_sep = "_"                # The character separating value from cue
  ) %>%
  mutate(Cue = recode(Cue, "light" = "Light", "bell" = "Bell"))

# STEP 2: Now, generate the density curves from this clean data frame.
theta <- seq(0, 1, length.out = 200)
dist_plot_data <- tidy_beliefs_df %>%
  rowwise() %>%
  mutate(
    dist = list(data.frame(theta = theta, density = dbeta(theta, a, b)))
  ) %>%
  unnest(dist) %>%
  # Create clean labels for the facets and order them
  mutate(
    Label = case_when(
      Trial == 0  ~ "Prior (Trial 0)",
      Trial == 10 ~ "End of Acquisition (Trial 10)",
      Trial == 20 ~ "End of Compound Phase (Trial 20)",
      Trial == 30 ~ "End of Extinction (Trial 30)"
    ),
    Label = factor(Label, levels = c("Prior (Trial 0)", "End of Acquisition (Trial 10)", "End of Compou
  )


# --- The ggplot call is now simpler and will work correctly ---

ggplot(dist_plot_data, aes(x = theta, y = density, fill = Cue, color = Cue)) +
  geom_area(alpha = 0.6, position = "identity") +

  # This now works because 'a' and 'b' exist in dist_plot_data
  geom_vline(aes(xintercept = a / (a + b), color = Cue), linetype = "dashed", linewidth=1) +

  facet_wrap(~ Label, ncol = 2) +
  scale_fill_manual(values = c("Light" = "dodgerblue", "Bell" = "firebrick")) +
  scale_color_manual(values = c("Light" = "dodgerblue", "Bell" = "firebrick")) +
  labs(
    title = "Evolution of Belief Distributions: Light vs. Bell",
    subtitle = "Simple Bayesian model (without blocking mechanism).",
    x = expression(paste("Hypothesized Rate of Food ", P(Food ~ "|" ~ Cue))),
    y = "Density"
  ) +
  theme_minimal(base_size = 10) +
  theme(legend.position = "bottom",
        panel.grid.minor = element_blank(),
        strip.text.x = element_text(size=10))
```



This initial output illustrates the Prior distribution indicating the belief that the probability that food all probabilities were equally likely

```
# ---  Visualize the Learning Process ---

# Plot 1: The Learning Curve (Comparable to Rescorla-Wagner)
# Here we plot the MEAN of the belief distribution, which is analogous to V.
mean_plot_data <- history %>%
  select(Trial, Phase, Light = mean_light, Bell = mean_bell) %>%
  pivot_longer(cols = c("Light", "Bell"), names_to = "Cue", values_to = "Strength")

ggplot(mean_plot_data, aes(x=Trial, y=Strength, color=Cue, linetype=Cue)) +
  geom_line(linewidth=1.2) +
  geom_vline(xintercept = c(10.5, 20.5), linetype="dashed", color="gray40") +
  scale_y_continuous(limits = c(0, 1.1), breaks = seq(0, 1, 0.2)) +
  labs(title="Bayesian Simulation: Mean of Belief (P(Food|Cue))",
       subtitle="This plot is analogous to the Rescorla-Wagner learning curve.",
       x="Trial Number",
       y=y <- expression(paste("Mean of Posterior Belief ", E, "[", theta, "]"))) +
  theme_minimal(base_size = 10) +
  theme(legend.position = "bottom")
```



Comparing the Models

The Rescorla-Wagner model is an error-driven model. Learning is proportional to "surprise." In a blocking trial (Light+Bell -> Food): The well-trained Light already causes the model to expect food (Total Expectation is high). When the bell is combined with the light and food still appears, there is no surprise (Prediction Error is near zero). When the light continues to appear, but it is no longer paired with food, rapid extinction occurs as the model "forgets" the association was ever there.

The simple Bayesian model treats learning as independent evidence gathering. It asks two separate questions:

- What is the probability of food given the Light? P(Food | Light)
- What is the probability of food given the Bell? P(Food | Bell)

In a blocking trial (Light+Bell -> Food): The model observes two events simultaneously:

- "The Light was present, and food occurred." -> This is evidence, so it strengthens the belief in P(Food | Light).
- "The Bell was present, and food occurred." -> This is also treated as evidence, so it strengthens the belief in P(Food | Bell).

Thus, the model gives credit to both cues. It has no built-in way to understand that the Light's presence explains away the food, making the Bell's presence uninformative. This leads to the most important distinction. The Bayesian model is a historian. It remembers every single trial. Its final belief after extinction reflects the entire history of the experiment—both the initial period where the light predicted food and the later period where it did not. The final mean belief a / (a + b) is a rational balance of all evidence. This is why extinction can sometimes seem slower or less complete in a Bayesian model; the prior evidence still has weight.

The Rescorla-Wagner model is more forgetful. It only knows its current associative strength and the current error. The negative error signal directly overwrites the positive strength, effectively erasing the past learning.

Head-to-Head Comparison: The Illusory Correlation Task

Now, instead of modeling classical conditioning, let's apply both models to a classic cognitive bias task to see how their different architectures lead to different insights.

The Phenomenon: In an illusory correlation task, people see a series of behaviors performed by members of a majority group (Group A) and a minority group (Group B). The ratio of desirable to undesirable behaviors is identical for both groups, but because the minority group is smaller and undesirable behaviors are less frequent, the co-occurrence of these two rare events becomes highly salient. This is thought to lead people to form a biased judgment, overestimating the rate of undesirable behavior for the minority group.

Importantly, the illusory correlation isn't just about minority group biases, the illusory correlation has been shown to explain everything from diagnoses based on responses to the Rorschach test to the formation of personal superstitions. For example, an athlete might wear a specific pair of socks-a distinctive behavior-before a major victory, a rare outcome. The brain forges a powerful link between these two unusual events, creating a "lucky charm" out of pure coincidence. This same mechanism underlies many beliefs in the effectiveness of pseudoscientific remedies, where a person experiencing a fluctuating symptom feels better after trying an unusual treatment and incorrectly attributes their recovery to the treatment rather than to chance. At its core, the phenomenon is driven by a cognitive shortcut: our minds are highly sensitive to the joint occurrence of two rare or distinctive events, making that pairing feel more significant and frequent than it truly is. Once this initial, illusory link is formed, it is often cemented by confirmation bias, as we then tend to notice and remember subsequent instances that support the perceived correlation while ignoring the many times the pattern does not hold. In this way, illusory correlation is a fundamental feature of human cognition that helps explain how we construct patterns and causal theories—both real and imagined—to make sense of a complex world.

Step 1: Create the Stimulus Data

Let's create a simple dataset that represents two groups of people-Group A and Group B. The dataset will also contain a behavior describing each person. The true proportion of negative behaviors is the same for two groups (approx 20%), but Group A is much larger than Group B. Instead of listing the actual behavior in our sample dataset, we will just include a factor describing whether the behavior was positive or negative.

```r
# --- Create the Illusory Correlation Dataset ---
set.seed(42) # For reproducible shuffling

group_a_pos <- 80
group_a_neg <- 20
group_b_pos <- 20
group_b_neg <- 5

# Create vectors of observations
observations <- data.frame(
  group = c(rep("A", group_a_pos + group_a_neg), rep("B", group_b_pos + group_b_neg)),
  behavior = c(rep("positive", group_a_pos), rep("negative", group_a_neg),
               rep("positive", group_b_pos), rep("negative", group_b_neg))
)

# Shuffle the observations to simulate a random presentation order
shuffled_trials <- observations[sample(nrow(observations)), ]

# Add a lambda (outcome) value for R-W model (1 for negative, 0 for positive)
shuffled_trials$lambda <- ifelse(shuffled_trials$behavior == "negative", 1, 0)

# Check the true proportions
print(paste("Group A Negative Proportion:", group_a_neg / (group_a_pos + group_a_neg)))
```

```
## [1] "Group A Negative Proportion: 0.2"
```

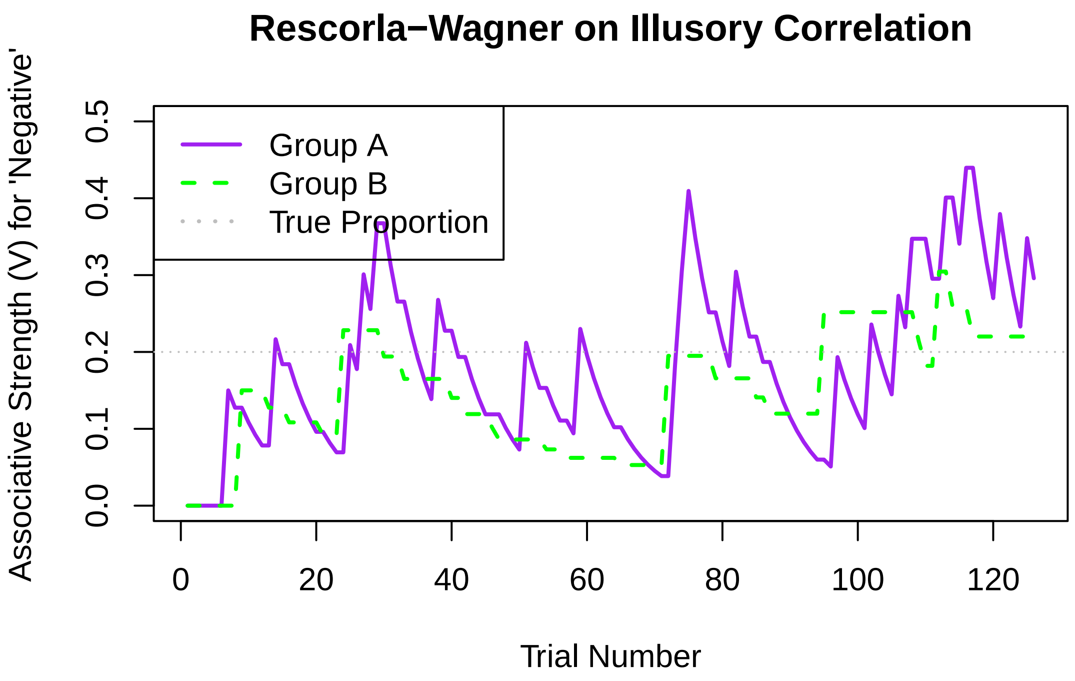
```
print(paste("Group B Negative Proportion:", group_b_neg / (group_b_pos + group_b_neg)))
```

```
## [1] "Group B Negative Proportion: 0.2"
```

Step 2: Simulate the Rescorla-Wagner Model

The R-W model will learn an association strength between each group and “negative behavior.”

```
# --- R-W Simulation ---
V_A <- 0 #Initial association between Group A and negative behaviors
V_B <- 0 #Initiol association between Group B and negative behaviors
alpha <- 0.3 #Salience of behaviors
beta <- 0.5 # Learning rate

# Store history containers
history_V_A <- c(0)
history_V_B <- c(0)

# Loop over trials
for (i in 1:nrow(shuffled_trials)) {
  trial <- shuffled_trials[i, ]

  if (trial$group == "A") {
    prediction_error <- trial$lambda - V_A
    delta_V <- alpha * beta * prediction_error
    V_A <- V_A + delta_V
  } else { # Group B trial
    prediction_error <- trial$lambda - V_B
    delta_V <- alpha * beta * prediction_error
    V_B <- V_B + delta_V
  }

  history_V_A <- c(history_V_A, V_A)
  history_V_B <- c(history_V_B, V_B)
}

# --- Plot R-W Results ---
plot(history_V_A, type="l", col="purple", lwd=2, ylim=c(0, 0.5),
     xlab="Trial Number", ylab="Associative Strength (V) for 'Negative'",
     main="Rescorla-Wagner on Illusory Correlation")
lines(history_V_B, type="l", col="green", lwd=2, lty=2)
abline(h = group_a_neg / (group_a_pos + group_a_neg), lty=3, col="gray")
legend("topleft", legend=c("Group A", "Group B", "True Proportion"),
       col=c("purple", "green", "gray"), lty=c(1, 2, 3), lwd=2)
```



Step 3: Simulate the Bayesian Beta-Binomial Model

The Bayesian model will update its belief distribution for each group's rate of negative behavior.

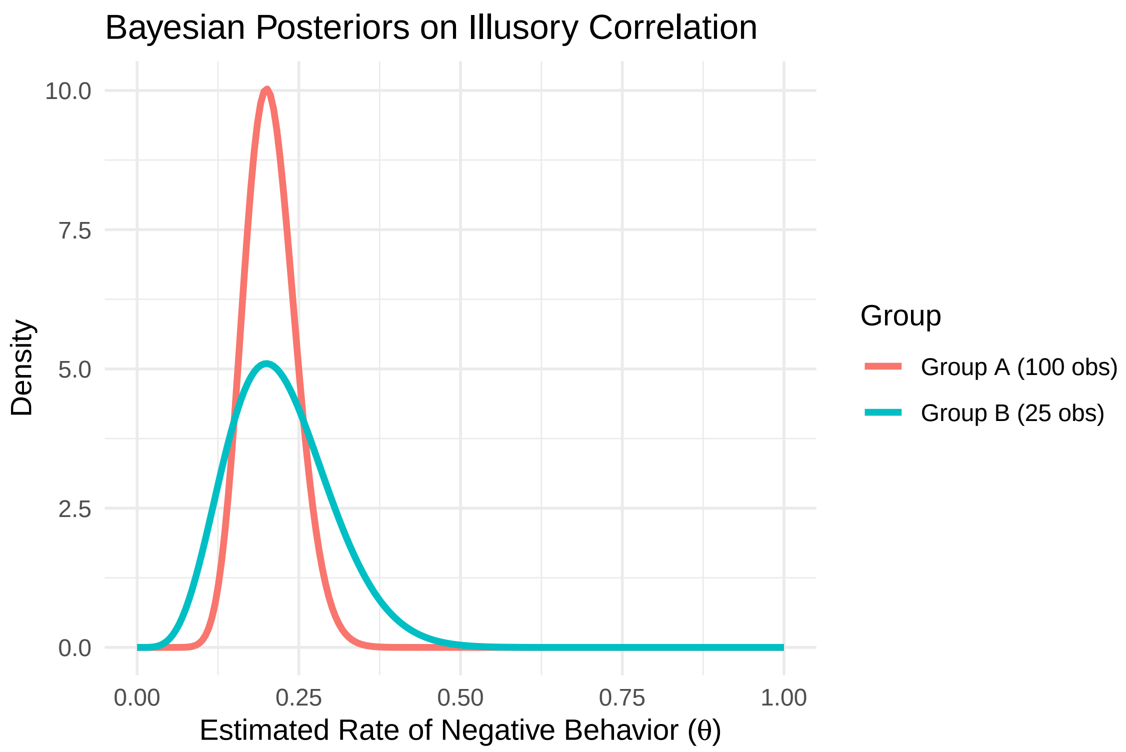
```
# --- Bayesian Simulation ---
# Prior beliefs for both groups (uninformative)
prior_a_A <- 1; prior_b_A <- 1
prior_a_B <- 1; prior_b_B <- 1

# --- Run through the trials ---
# We can just count the final outcomes for simplicity
final_counts <- aggregate(lambda ~ group, data=shuffled_trials, FUN=sum)
total_counts <- as.data.frame(table(shuffled_trials$group))
names(final_counts) <- c("group", "neg_behaviors")
final_counts$pos_behaviors <- total_counts$Freq - final_counts$neg_behaviors
show(final_counts)
```

```
##   group neg_behaviors pos_behaviors
## 1     A            20            80
## 2     B             5            20
```

```
# Update posteriors based on all data
posterior_a_A <- prior_a_A + final_counts$neg_behaviors[final_counts$group == "A"]
posterior_b_A <- prior_b_A + final_counts$pos_behaviors[final_counts$group == "A"]

posterior_a_B <- prior_a_B + final_counts$neg_behaviors[final_counts$group == "B"]
posterior_b_B <- prior_b_B + final_counts$pos_behaviors[final_counts$group == "B"]

# --- Plot Bayesian Results ---
theta <- seq(0, 1, length.out=200)
density_A <- dbeta(theta, posterior_a_A, posterior_b_A)
density_B <- dbeta(theta, posterior_a_B, posterior_b_B)

plot_data_bayes <- data.frame(
```

```r
  theta = rep(theta, 2),
  density = c(density_A, density_B),
  Group = rep(c(paste0("Group A (", total_counts$Freq[1], " obs)"),
                paste0("Group B (", total_counts$Freq[2], " obs)")),
              each=200)
)

ggplot(plot_data_bayes, aes(x=theta, y=density, color=Group)) +
  geom_line(size=1.2) +
  labs(title="Bayesian Posteriors on Illusory Correlation",
       x=expression(paste("Estimated Rate of Negative Behavior (", theta,")")),
       y="Density") +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Interpretation of the Model's Predictions

Like the R-W model, the peaks (means) of both posterior distributions are centered very close to the true rate of 20%. Both models are "rational" in that they correctly estimate the central tendency of the data.

Look at the width of the two distributions. The curve for Group B is noticeably shorter and wider than the curve for Group A. This visually represents the model's uncertainty. Because it has seen fewer examples from the minority Group B, its belief about Group B's true nature is less precise.

Explaining the Bias: This difference in uncertainty provides a potential explanation for the illusory correlation bias. The Rescorla-Wagner model simply reports the learned statistic. The Bayesian model suggests that while the best guess for both groups is the same, the learner is far less confident about their judgment of the minority group. This lack of certainty could make the judgment more susceptible to being swayed by other factors, such as the high salience of rare events, leading to the biased human conclusion that R-W cannot explain.

That being said, neither model predicts the kind of behavior that is typically observed in human participants. That is, both models will, on average, predict the same rate of negative behaviors in both groups of people.

- Think about it… what is missing?

Modeling Social Bias - How Priors Can Create Illusory Correlations

In the previous section, we showed how a rational Bayesian model's uncertainty could create the conditions for a bias to form. But what if the bias is already present before learning begins? The Bayesian framework is perfect for testing this idea by using an *informative* prior.

Let's hypothesize that a learner doesn't start with a blank slate. They may have a pre-existing societal bias that minority groups are more associated with negative outcomes. We can model this as a prior belief that is already skewed.

Step 1: Defining a Biased Prior

Instead of an uninformative Beta(1, 1) prior for Group B, let's use a Beta(3, 1) prior.

What this means: This prior is equivalent to having already observed 3 "pseudo-examples" of negative behavior and only 1 "pseudo-example" of positive behavior before the experiment even starts. It's a belief, however weak, that the rate of negative behavior for Group B is likely to be high (peaking at a rate of (3-1)/(3+1-2) = 1, but with high uncertainty). We will keep the prior for the majority Group A as Beta(1, 1) (unbiased).

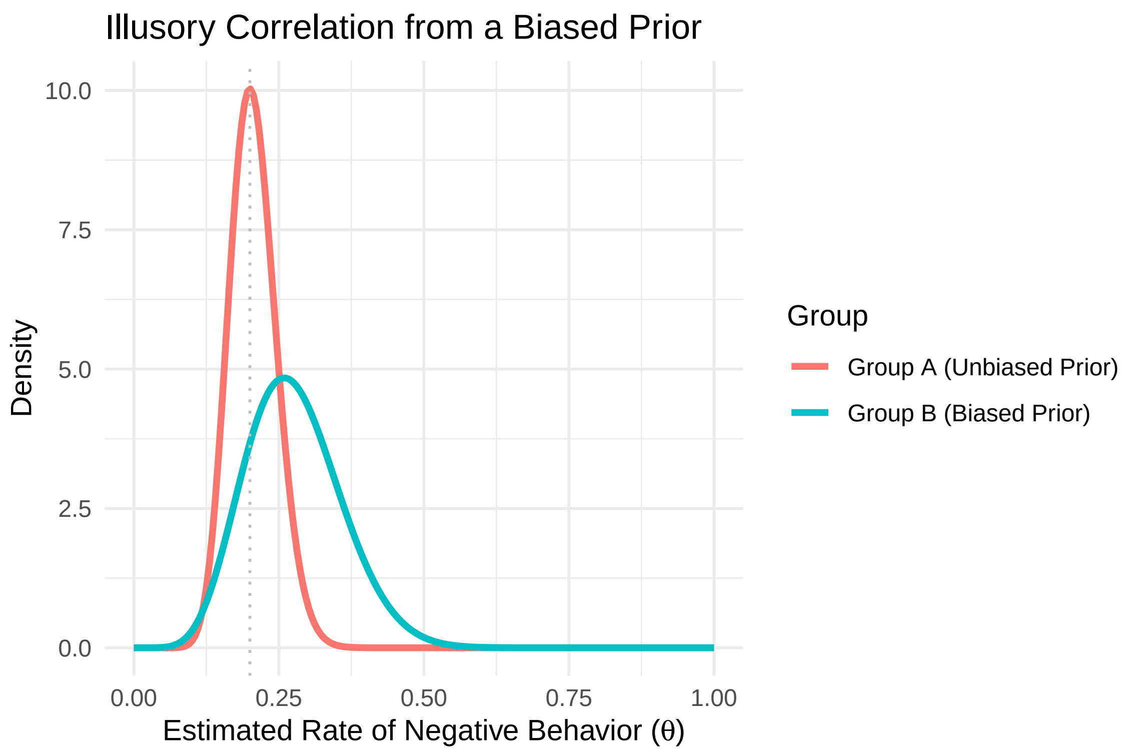
Step 2: Rerunning the Simulation with the Biased Prior

We use the exact same data as before. The only thing we change is the starting belief for Group B.

```r
# --- Biased Bayesian Simulation ---
# Prior for Group A is uninformative
prior_a_A_biased <- 1; prior_b_A_biased <- 1

# Prior for Group B is BIASED towards negative behaviors
prior_a_B_biased <- 3; prior_b_B_biased <- 1

# --- Recalculate Posteriors ---
# Posterior for A is the same as before
posterior_a_A_biased <- prior_a_A_biased + final_counts$neg_behaviors[final_counts$group == "A"]
posterior_b_A_biased <- prior_b_A_biased + final_counts$pos_behaviors[final_counts$group == "A"]

# Posterior for B now incorporates the biased prior
posterior_a_B_biased <- prior_a_B_biased + final_counts$neg_behaviors[final_counts$group == "B"]
posterior_b_B_biased <- prior_b_B_biased + final_counts$pos_behaviors[final_counts$group == "B"]

# --- Plot the Biased Results ---
density_A_biased <- dbeta(theta, posterior_a_A_biased, posterior_b_A_biased)
density_B_biased <- dbeta(theta, posterior_a_B_biased, posterior_b_B_biased)

plot_data_bayes_biased <- data.frame(
  theta = rep(theta, 2),
  density = c(density_A_biased, density_B_biased),
  Group = rep(c("Group A (Unbiased Prior)", "Group B (Biased Prior)"), each=200)
)

# Calculate the peak (mode) of the Group B posterior
mode_B_biased <- (posterior_a_B_biased - 1) / (posterior_a_B_biased + posterior_b_B_biased - 2)
print(paste("Peak of Biased Group B Posterior:", round(mode_B_biased, 3)))
```

```
## [1] "Peak of Biased Group B Posterior: 0.259"
```

```
ggplot(plot_data_bayes_biased, aes(x=theta, y=density, color=Group)) +
  geom_line(size=1.2) +
  geom_vline(xintercept = group_a_neg / (group_a_pos + group_a_neg), lty=3, col="gray") +
  labs(title="Illusory Correlation from a Biased Prior",
       x=expression(paste("Estimated Rate of Negative Behavior (", theta,")")),
       y="Density") +
  theme_minimal()
```



The new plot tells a very different story.

The posterior for Group A is unchanged, peaking at the true rate of 20%.

However, the posterior for Group B is now visibly shifted to the right. Its peak is no longer at 0.2; the code output shows it's now around 0.25 or higher.

Even though the model was presented with the exact same evidence, the biased prior "pulled" the final belief towards it. The model's final conclusion is that the rate of negative behavior for the minority group is indeed higher than for the majority group. The model has formed an illusory correlation.

This powerful demonstration shows that the Bayesian framework can not only represent rational learning under uncertainty but can also be used to precisely formalize and test hypotheses about how pre-existing cognitive biases shape our interpretation of evidence. It transforms a vague idea ("prior beliefs matter") into a specific, quantifiable prediction.

Modeling A General Bias - Uniqeness

While modeling prior beliefs can be useful in the context of questions about social biases, we need to also address the fact that the illusory correlation phenomenon is robust and can be observed even when individual learners have no prior experience with the experimental stimuli. Recall that the prevailing theory about illusory correlation is that learning is different in the context of rare events and this is what causes the resulting enhancement of the perceived relationship between low-frequency (e.g., negative) behaviors and the smaller group (e.g., Group B). However, both of the models that were just articulated use the same updating (i.e., learning) process on all trials. This makes them good at modeling rational behavior, but not so good at modeling human behavior.

- Think about it… Look back at the code for each model and think about how you might implement the irrational behavior of humans.

In order to make either model demonstrate the illusion, we must introduce an asymmetry in the learning that reflects the underlying psychological theory. The theory states that the co-occurrence of two rare events (Minority Group + Negative Behavior) is highly distinctive or salient, and is therefore overweighted in learning.

1. ***The Rescorla-Wagner model***

Simple Approach

This approach assumes the minority group is inherently more distinctive, so we pay more attention to them in general. We can model this with a permanently higher alpha (salience) for the minority group.

The Logic:

a. Majority Group (A) is common, so it has a standard salience: alpha_A = 0.3
b. Minority Group (B) is rare and distinctive, so it has a higher salience: alpha_B = 0.5

Your loop would then check which group is present and use the appropriate alpha. This is a good first approximation, but it doesn't fully capture that the combination of the minority group and the negative behavior is what's special.

More Accurate Approach

This is a more precise way to model the "doubly distinctive" theory. Instead of one group being more salient all the time, we say that the learning process itself is amplified on the specific trials where the rare group and rare behavior co-occur.

The Logic:

a. On most trials, the learning rate is standard (alpha*beta).
b. On the rare, salient trials (Minority Group B + Negative Behavior), we temporarily boost the learning rate for that single update.

This method more directly targets the specific cognitive mechanism thought to cause the illusion.

Implementation in R

Experimental Setup:

The experimental setup will be identical to that used earlier.

```r
# --- Create the Illusory Correlation Dataset ---
set.seed(42) # For reproducible shuffling

group_a_pos <- 80
group_a_neg <- 20
group_b_pos <- 20
group_b_neg <- 5
```

```r
# Create vectors of observations
observations <- data.frame(
  group = c(rep("A", group_a_pos + group_a_neg), rep("B", group_b_pos + group_b_neg)),
  behavior = c(rep("positive", group_a_pos), rep("negative", group_a_neg),
               rep("positive", group_b_pos), rep("negative", group_b_neg))
)

# Shuffle the observations to simulate a random presentation order
shuffled_trials <- observations[sample(nrow(observations)), ]

# Add a lambda (outcome) value for R-W model (1 for negative, 0 for positive)
shuffled_trials$lambda <- ifelse(shuffled_trials$behavior == "negative", 1, 0)

# Check the true proportions
print(paste("Group A Negative Proportion:", group_a_neg / (group_a_pos + group_a_neg)))
```

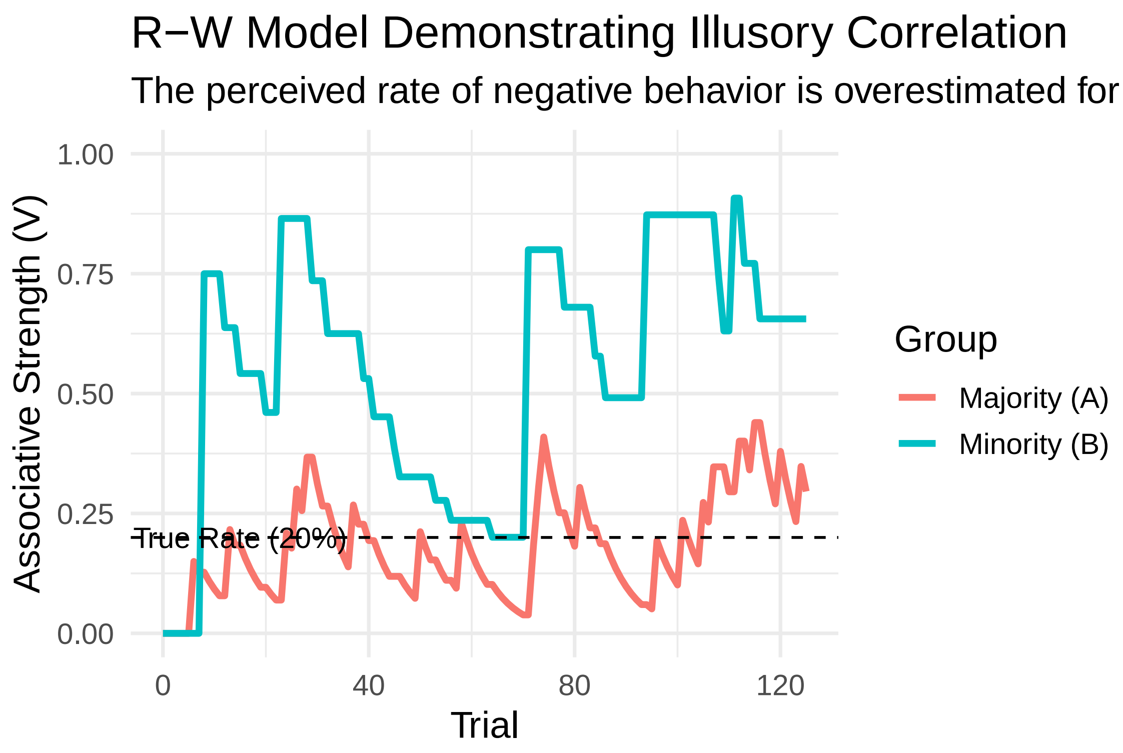
```
## [1] "Group A Negative Proportion: 0.2"
```

```r
print(paste("Group B Negative Proportion:", group_b_neg / (group_b_pos + group_b_neg)))
```

```
## [1] "Group B Negative Proportion: 0.2"
```

Implement the new model:

```r
# --- 2. R-W Model Parameters ---
V_A <- 0 # Initial association between Group A and negative behaviors
V_B <- 0 # Initial association between Group B and negative behaviors

alpha <- 0.3 # Base salience of the groups
beta <- 0.5  # Base learning rate for behaviors
salience_boost <- 5 # How much to amplify learning on "doubly distinctive" trials

# Containers to store history for plotting (start with initial values)
history_V_A <- c(V_A)
history_V_B <- c(V_B)

# --- 3. Run the Simulation Loop with Dynamic Learning ---
for (i in 1:nrow(shuffled_trials)) {
  trial <- shuffled_trials[i, ]

  # Set the base learning rate
  effective_learning_rate <- alpha * beta

  # On the special, "doubly distinctive" trials, boost the learning rate
  if (trial$group == "B" && trial$lambda == 1) {
    effective_learning_rate <- effective_learning_rate * salience_boost
  }

  # Update the associative strength for the presented group
  if (trial$group == "A") {
    prediction_error <- trial$lambda - V_A
    delta_V <- effective_learning_rate * prediction_error
    V_A <- V_A + delta_V
  } else { # Group B trial
    prediction_error <- trial$lambda - V_B
    delta_V <- effective_learning_rate * prediction_error
    V_B <- V_B + delta_V
  }

  # Store the updated strengths
  history_V_A <- c(history_V_A, V_A)
  history_V_B <- c(history_V_B, V_B)
}

# --- 4. Visualize the Results ---
plot_data <- tibble(
  Trial = 0:nrow(shuffled_trials),
  V_A = history_V_A,
  V_B = history_V_B
) %>%
  pivot_longer(cols = c(V_A, V_B), names_to = "Group", values_to = "Strength") %>%
  mutate(Group = recode(Group, "V_A" = "Majority (A)", "V_B" = "Minority (B)"))

# The final plot will show V_B is higher than V_A
ggplot(plot_data, aes(x = Trial, y = Strength, color = Group)) +
  geom_line(size = 1.2) +
  # Add a line for the TRUE probability
  geom_hline(yintercept = 1/5, linetype = "dashed", color = "black") +
  annotate("text", x = 15, y = 0.2, label = "True Rate (20%)", color = "black") +
  scale_y_continuous(limits = c(0, 1)) +
  labs(
    title = "R-W Model Demonstrating Illusory Correlation",
    subtitle = "The perceived rate of negative behavior is overestimated for the minority group.",
    x = "Trial",
    y = "Associative Strength (V)"
  ) +
  theme_minimal(base_size = 14)
```



As you can see from the plot above, this Rescorla-Wagner model is capable of demonstrating more human-like behavior in that learning is exaggerated on trials with double-distinctiveness, resulting in a biased association between group B and negative behaviors. Importantly, however, this model requires the salience boost to be very high or that the model is exposed to very large samples.

- Think about it: Why does this model require the salience boost to be very high OR require a large number of trials?

1. ***The Bayesian model***

Modifying the original Baysian model is conceptually very similar to how we modified the Rescorla-Wagner model. We will move the model from being a purely rational evidence-gatherer to a more human-like one by making it overweight rare, distinctive events.

A simple Bayesian model treats every observation as exactly "1" piece of new evidence. To show the illusory correlation, we will modify this rule by introducing bias. We will tell the model that the "doubly distinctive" event (the co-occurrence of the rare Minority Group and the rare Negative Behavior) is highly salient and should be counted as more than just 1 piece of evidence.

- The Logic:

a. We will introduce a salience_boost parameter (e.g., 1.8). This can be interpreted to mean that one observation of a minority member performing a negative act is as impactful as seeing it happen 1.8 times.

Here is how the learning rule changes:

- Standard Rule:

If Negative Behavior: a <- a + 1

If Positive Behavior: b <- b + 1

- Our New Biased Rule:

If Majority Group + Negative Behavior: a_groupA <- a_groupA + 1

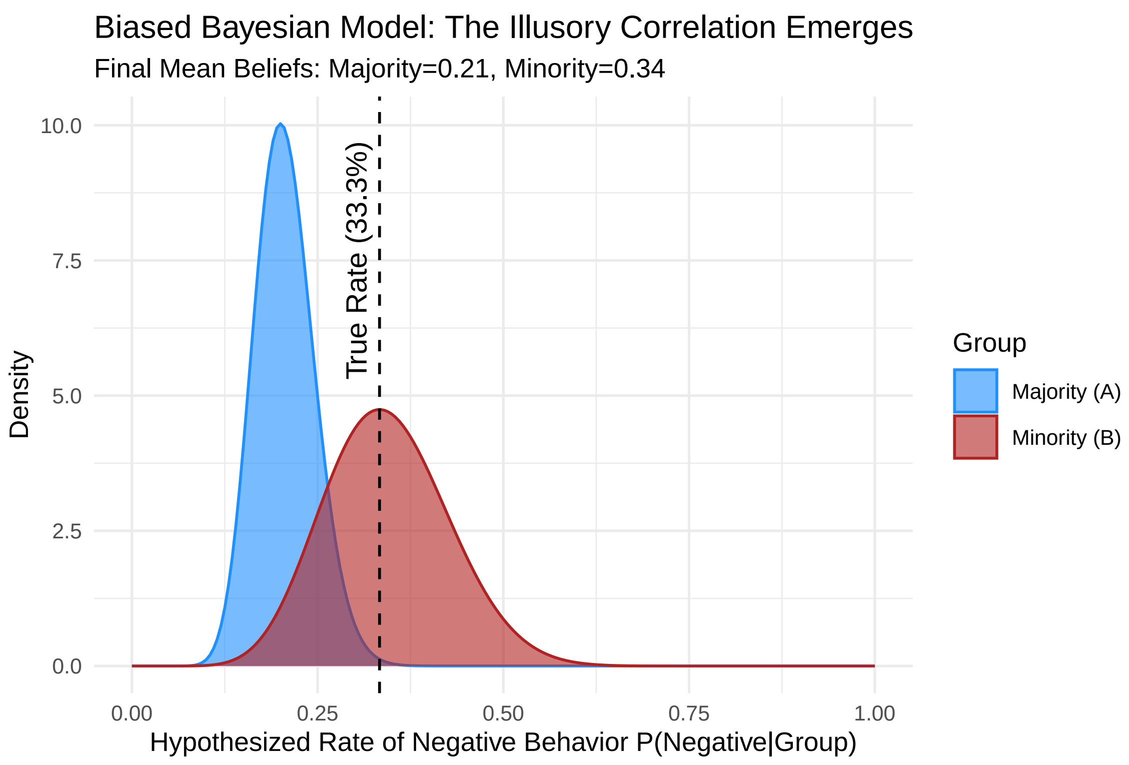
If Minority Group + Negative Behavior: a_groupB <- a_groupB + salience_boost

If Positive Behavior: b <- b + 1 (Positive behaviors are common and not distinctive, so they are not boosted).

Implementation in R

```r
# --- 2. Bayesian Model Parameters ---
# Start with uninformative priors (Beta(1,1)) for P(Negative|Group)
negative_groupA <- 1; positive_groupA <- 1 # Belief for Majority Group A
negative_groupB <- 1; positive_groupB <- 1 # Belief for Minority Group B

# --- THIS IS THE KEY ADDITION ---
# The psychological bias parameter
salience_boost <- 2 # How much to "overweight" a salient observation

# --- 3. Run the "Biased" Simulation Loop ---
for (i in 1:nrow(shuffled_trials)) {
  trial <- shuffled_trials[i, ]

  is_negative <- trial$behavior == "negative" #1 if negative / 0 if positive

  if (trial$group == "A") {
    # Majority group updates are always standard
    negative_groupA <- negative_groupA + is_negative
    positive_groupA <- positive_groupA + !is_negative
  } else { # Minority Group B
    # For the minority group, the update depends on the behavior
    if (is_negative) {
      # This is the "doubly distinctive" event, so we apply the boost
      negative_groupB <- negative_groupB + salience_boost
    } else {
      # Positive behaviors are not distinctive, so the update is standard
      positive_groupB <- positive_groupB + 1
    }
  }
}

# --- 4. Visualize the Final Biased Beliefs ---
theta <- seq(0, 1, 0.005)

# Create a data frame with the final posterior densities for both groups
final_beliefs <- tibble(
  theta = theta,
  Group_A_density = dbeta(theta, negative_groupA, positive_groupA),
  Group_B_density = dbeta(theta, negative_groupB, positive_groupB)
) %>%
  pivot_longer(
    cols = -theta,
    names_to = "Group",
    values_to = "Density"
  ) %>%
  mutate(Group = recode(Group, "Group_A_density" = "Majority (A)", "Group_B_density" = "Minority (B)"))

# Calculate the final means of the beliefs
mean_A <- negative_groupA / (negative_groupA + positive_groupA)
```

```r
mean_B <- negative_groupB / (negative_groupB + positive_groupB)

# Plot the final distributions
ggplot(final_beliefs, aes(x = theta, y = Density, color = Group, fill = Group)) +
  geom_area(alpha = 0.6, position = "identity") +
  geom_vline(xintercept = 1/3, linetype = "dashed", color = "black") +
  annotate("text", x = 0.33, y = 7.5, label = "True Rate (33.3%)", color = "black", angle = 90, vjust =
  scale_fill_manual(values = c("Majority (A)" = "dodgerblue", "Minority (B)" = "firebrick")) +
  scale_color_manual(values = c("Majority (A)" = "dodgerblue", "Minority (B)" = "firebrick")) +
  labs(
    title = "Biased Bayesian Model: The Illusory Correlation Emerges",
    subtitle = paste0("Final Mean Beliefs: Majority=", round(mean_A, 2), ", Minority=", round(mean_B, 2
    x = "Hypothesized Rate of Negative Behavior P(Negative|Group)",
    y = "Density"
  ) +
  theme_minimal(base_size = 10)
```



Examing the predictions of the biased Bayesian model, you can see that, in addition to the increased uncertainty (wider probability distribution) about group B, the increased salience of the double-distinctive events (e.g., group B + negative behavior) has led the model to form an illusory correlation. That is, its posterior "belief" after training is that group B has an increased probability of being associated with negative behaviors despite the fact that negative behaviors are equally proportional in the two groups.

From Simulation to Data: Fitting Models to Human Behavior

The ultimate test of a model is not just whether it can reproduce a phenomenon in principle, but whether it can explain the specific, trial-by-trial choices of a real person. This process is called model fitting. The goal when fitting a model is to find the parameter values that make the participant's actual choices seem most probable.

Let's use the Biased Bayesian model from the previous section and fit it to data from an illusory correlation experiment.

The Task:

A participant sees one of two planets: "Planet Zorg" (frequent, 80% of trials) or "Planet Xylos" (infrequent, 20% of trials) on each trial, then they predict if the inhabitants are "positive" or "negative". After making their prediction, they are shown a word (a positive or negative word) that describes the people on the planet. The word is negative 20% of the time for both planets.

The Data

First, we need to collect some data and record the participant's trial-by-trial data. For present puposes we'll simulate a sample data frame.

```r
# --- Create Sample Participant Data ---
set.seed(101) # For a new reproducible participant
n_trials <- 150 #number of simulated trials
planet_Zorg_freq <- 0.8 #Frequency for common planet
neg_feedback_prob <- 0.2 # Probability of negative feedback

# Create the trial sequence
planets <- sample(c("Zorg", "Xylos"), n_trials, replace=TRUE, prob=c(planet_Zorg_freq, 1-planet_Zorg_fr
feedback <- rbinom(n_trials, 1, neg_feedback_prob) # 1=negative, 0=positive

# Simulate some plausible choices (a real dataset would have actual choices)
# Let's assume the participant learns and shows a slight bias
sim_choices <- rbinom(n_trials, 1, 0.6)

participant_data <- data.frame(
  trial = 1:n_trials,
  planet = planets,
  choice = sim_choices, # This would be the participant's actual button press
  feedback = feedback # This is the outcome/lambda
)

head(participant_data)
```

```
##   trial planet choice feedback
## 1     1   Zorg      1        0
## 2     2   Zorg      1        0
## 3     3   Zorg      1        0
## 4     4   Zorg      1        1
## 5     5   Zorg      1        0
## 6     6   Zorg      1        0
```

Connecting the Model to Participant Behavior

Recall that our Bayesian model's primary output is a full probability distribution—the Beta(a, b)—which represents its entire belief state, including its uncertainty. However, a participant in an experiment provides a discrete, binary choice (e.g., predicting a planet will be 'positive' or 'negative'). To bridge this gap between the model's rich internal belief and the observable behavior, we need a decision function or "choice rule."

A common and psychologically plausible choice rule is the softmax function. It provides a principled way to translate the model's internal beliefs into the probability of making a specific choice. In the case we are particularly interested the probability of the model choosing 'negative' for a given planet, which can be calculated as:

$$P(ChooseNegative) = \frac{1}{1 + e^{(-(Belief_{planet} - 0.5) * \tau)}}$$

Where:

- $Belief_{planet}$ - is the mean of the posterior belief (θ): **This is the crucial link to our Bayesian model. It is calculated as a / (a + b) for the group being judged. It represents the model's current "best guess" for the probability of a behavior being negative for that group.**
- τ (tau) — is the "Inverse Temperature" of the softmax function: **This is a free parameter that determines the sensitivity of choices to the model's internal belief.** You can think of it as a "confidence" or "consistency" dial. A high τ makes choices highly sensitive to the belief. The model will almost always choose the option with the higher value (deterministic, exploitative behavior). A low τ makes choices less sensitive to the belief, making them more random and noisy (exploratory behavior).
- 0.5 — is the decision threshold: We compare the model's belief to 0.5 because it represents the point of indifference. If the model believes the probability of negative behavior is greater than 0.5, it leans towards a 'negative' choice; if less than 0.5, it leans towards 'positive'.

Set Up The Objective Function (Finding the Best-Fitting Parameters)

The core of fitting our model to a participant's actual behavior is the objective function. Think of it as a "scoring" mechanism. Its goal is to find the specific set of parameter values that make our model's moment-to-moment predictions best align with the participant's actual choices.

This function works by taking a candidate set of parameters, running a simulation of our model trial-by-trial through the participant's data, and calculating the total log-likelihood of the participant's choices given the model's predictions. A higher log-likelihood signifies a better fit.

For our biased Bayesian model, this function needs to find the optimal values for two key psychological parameters:

1) salience_boost: This parameter directly captures the magnitude of the illusory correlation bias. A value greater than 1 indicates that the participant overweighs the evidence of a minority member performing a negative action.
2) tau (τ): The softmax inverse temperature. This parameter captures the participant's decision consistency, or how sensitively their choices reflect their internal beliefs.

We will create an objective function called "calculate_neg_log_likelihood_bayesian" that implements this process. The function takes a vector of parameter values (salience_boost, tau) and the participant's data as input. It returns a single number—the negative of the total log-likelihood. The reason returns the negative log-likelihood is because standard optimization functions in R are designed to find a minimum value. Minimizing the negative log-likelihood is equialent to maximizing the log-likelihood.

```r
# --- The Objective Function for the Biased Bayesian Model ---
# This function calculates how well a specific set of parameters (salience_boost, tau)
# explains a participant's sequence of choices.

calculate_neg_log_likelihood_bayesian <- function(params, data) {
  # --- Step 1: Extract Parameters to be Fitted ---
  salience_boost <- params[1] # The key bias parameter we want to find
  tau <- params[2]            # The softmax inverse temperature (choice sensitivity)

  # --- Step 2: Initialize the Model's "Mind" ---
  # Beliefs are Beta distributions, starting as Beta(1,1) (uninformative prior)
  a_A <- 1; b_A <- 1 # Beliefs for Majority Group A
  a_B <- 1; b_B <- 1 # Beliefs for Minority Group B
  total_log_likelihood <- 0

  # --- Step 3: Loop Through Each Trial of the Participant's Data ---
  for (i in 1:nrow(data)) {
    trial_data <- data[i, ]

    # === Part A: Make a Prediction Based on Current Belief ===
    # Get the model's current "best guess" for the probability of negative behavior
    if (trial_data$planet == "Zorg") { #Common planet
      current_belief_mean <- a_A / (a_A + b_A)
    } else { # Minority planet
      current_belief_mean <- a_B / (a_B + b_B)
    }

    # Use the Softmax rule to convert this belief into a choice probability
    prob_choose_neg <- 1 / (1 + exp(- (current_belief_mean - 0.5) * tau))

    # === Part B: Calculate the Likelihood of the Participant's ACTUAL Choice ===
    # Check if the model's predicted probability matches what the participant did
    if (trial_data$choice == 1) {
      likelihood <- prob_choose_neg
    } else {
      likelihood <- 1 - prob_choose_neg
    }

    # Add the log of the likelihood to the total (add a tiny number to prevent log(0))
    total_log_likelihood <- total_log_likelihood + log(likelihood + 1e-10)

    # === Part C: Update the Model's Belief Based on Feedback ===
    # This is the biased Bayesian learning rule, the engine of our model
    is_negative_feedback <- trial_data$feedback == 1

    if (trial_data$planet == "Zorg") {
      # Majority group updates are always standard (evidence count is +1)
      a_A <- a_A + is_negative_feedback
      b_A <- b_A + !is_negative_feedback
    } else { # Minority planet Xylos
      if (is_negative_feedback) {
        # This is the "doubly distinctive" event, so we apply the BIAS
        a_B <- a_B + salience_boost
      } else {
        # Positive behaviors are not distinctive, so the update is standard
        b_B <- b_B + 1
      }
    }
  }

  # Return the NEGATIVE log-likelihood, because optim() minimizes by default
  return(-total_log_likelihood)
}
```

Optimization

Now we use R's optim() function to find the params (salience_boost and tau) that minimize the negative log-likelihood (which is the same as maximizing the likelihood).

```r
# --- Fit the model ---
# Provide some starting guesses for the parameters
start_params <- c(salience_boost=2, tau = 3)

# Use optim() to find the best-fitting parameters
fit_results <- optim(
  par = start_params,                # Starting guesses
  fn = calculate_neg_log_likelihood_bayesian, # The function to minimize
  data = participant_data,           # The data to pass to the function
  lower = c(0.01, 0.1),              # Set plausible lower bounds for parameters
  upper = c(5, 20),                  # Set plausible upper bounds
  method = "L-BFGS-B"                # A good optimization algorithm for bounded problems
)

# --- View the results ---
best_params <- fit_results$par
names(best_params) <- c("salience_boost", "tau")
print("Best-fitting parameters:")
```

```
## [1] "Best-fitting parameters:"
```

```r
print(best_params)
```

```
## salience_boost            tau
##      5.0000000      0.2027225
```

Assignment

For this assignment, you will be asked to think about the assumptions that are built into the examples we covered and to modify one of the models to address that assumption. Your task is to create a new R-markdown file, using a combination of markdown and code blocks to address each of the items below.

1) We examined how biased priors could be used to model social biases in the Bayesian learning model. How would you implement pre-existing biases in the rational Rescorla-Wagner model? Implement your proposed change and run the model. Explain how the R-W and Bayesian differ in their response to the addition of pre-existing biases.
2) There is at least one major assumption built into both the R-W and Bayesian models of illusory correlation. This assumption has to do with prior "knowledge" that is programmed into the models. Analyze the code and describe the assumption and how it is implemented in the model.
3) Describe (in words) how you could fix the identified assumption to make the models more plausible. (*For example: I would change the α parameter so that...*)
4) Copy the code for either the R-W or Bayesian model and implement the change(s) you described in #2. Be sure to add comments to each line of code that you write or change, explaining what your addition/change does.
5) Run both the original code and your new code, producing a graph of the trial-by-trial learning for each. Write a few sentences explaining how the two models differ (if at all) in terms of their predicted learning patterns.