Computational Modeling of Behavior

Mark Steyvers

Contents

1. Introduction

1.1 The Purpose of Computational Modeling

In the fields of cognitive science and psychology, the overarching aim is to understand the inner workings of the mind to accurately describe, predict, and explain human behavior. Understanding the connection between the mind and behavior is crucial, as the mental processes—thoughts, memories, and decision-making mechanisms—serve as the driving forces that shape observable actions. In other words, your thought patterns and cognitive processes directly influence your actions and reactions to external stimuli, giving researchers a valuable window into the mind through the study of behavior.

To advance this understanding, computational modeling has become increasingly vital in recent years. This approach allows researchers to analyze a complex system, such as the human mind, by creating a simpler, computer-implemented version. This streamlined model focuses on core characteristics, while omitting extraneous elements, offering insights into the mind's complexity in a more manageable way.

Computational models serve as approximations of mental processes. They simulate internal mental activities to generate specific behaviors that are of interest to researchers. These models can be quite useful; they can be applied to various psychological experiments or real-world scenarios. By running these models in conditions that mimic those of specific experiments or real-world environments, researchers can gain valuable data to describe, predict, and explain human behavior. Thus, computational models serve as valuable tools for exploring the intricate connection between the mind and behavior.

1.2 Advantages of Computational Modeling

Precision & Clarity

Computational models add rigor and precision to verbally stated theories. By requiring a theory to be implemented as a computer program, they force researchers to be explicit about their assumptions. Every variable must be clearly defined, operationalized, and set within the model, eliminating room for vagueness or ambiguity. This process ensures that all elements of a theory can be tested and

validated against empirical data. The relationships between variables have to be fully articulated, often uncovering hidden assumptions or gaps in understanding. This makes computational models an invaluable tool for enhancing the scientific robustness of theoretical frameworks.

Explanation

Computational models offer a powerful approach to understanding complex behaviors by "fitting" them to empirical data. This fitting process involves adjusting various parameters in the model so that its output closely matches observed behaviors. Once the model has been fit, the adjusted parameter values become more than just numerical coefficients; they serve as a computational explanation for the observed behaviors.

One area where this can be particularly insightful is in analyzing the results from behavioral experiments. Suppose a study involves multiple conditions, such as condition A and condition B, and produces different outcomes under each. By adjusting the parameters in a computational model to fit the data across conditions, researchers can isolate the variables or mechanisms that led to the divergent results. The adjusted parameters may reveal, for example, that a specific cognitive process is more active in condition A compared to condition B. This provides a quantifiable and precise way to understand the differences between conditions, lending empirical support to theoretical claims.

Similarly, computational models can be used to explore differences between individuals. By fitting a model to the behavioral data of individual participants, researchers can identify which parameters differ significantly across people. These parameter values can serve as computational markers for traits or cognitive processes that vary between individuals. For instance, one person might be more risk-averse than another, and this difference would manifest in the corresponding parameters of a decision-making model.

Exploration

Computational models offer a unique advantage in their ability for theoretical exploration. They allow researchers to simulate various "what-if" scenarios and predict outcomes in conditions that are difficult or even impossible to test empirically. For example, neural networks can serve as a testing ground for understanding the role of specific components in a system. By "lesioning" parts of the network, we can simulate the effects of damage to certain brain regions. This can offer insights into the function of those regions without the ethical concerns of human experimentation. As another example, computational models can also expedite educational research by simulating different learning schedules and methods. Running simulations is far quicker than conducting long-term empirical studies, allowing researchers to identify promising strategies that can be tested empirically.

1.3 Choosing between types of models: accuracy and interpretability

In behavioral and cognitive sciences, researchers face a plethora of model choices, each offering distinct advantages and limitations. A constant dilemma in selecting an appropriate model is balancing predictive accuracy with ease of interpretation (see Figure 1.1). Some models excel in forecasting outcomes but are opaque in revealing how they arrived at such predictions. In contrast, models that are transparent may lack the predictive robustness of their more complex counterparts. Box 1.3.1 describes three types of computational models to highlight this trade off.

In this book, we will focus mainly on models such as cognitive models. These models offer a balanced approach between accuracy and interpretability. These models allow us to better understand human behavior and provide some glimpses into the underlying mechanisms. However, the choice of model should be dictated by the research objectives and the specific needs of the study. Whether the objective is to achieve high predictive accuracy or to understand the underpinnings of behavior, each model has something unique to offer.

Box 1.3.1: Types of Models

Deep Neural Networks
Deep neural networks, a specialized subset of machine learning algorithms, are well known for their predictive accuracy. Initially designed for tasks like image and speech recognition, their application has extended to the field of cognitive science and psychology as models of human cognition and behavior. Some research suggests that the layered architecture of neural networks resembles the hierarchical organization of the human brain, providing a computational counterpart to neurobiological processes (Kriegeskorte; 2015; Battleday et al.; 2021).

However, deep neural networks are often criticized for their "black box" nature, which makes their inner workings hard to interpret. Recently, efforts have been made to improve their interpretability through techniques like feature visualization and sensitivity analysis. While they are not fully transparent, these advances mean that deep neural networks are becoming increasingly useful for studies that require both high accuracy and some degree of interpretability.

Cognitive Models
Cognitive process models aim to simulate the mental operations leading to observed behavior. They offer a compromise between accuracy and interpretability. Not as predictive as deep neural networks, these models shine in elucidating the "why" and "how" behind actions and decisions. Derived from psychological theories, these models are designed to represent cognitive functions like memory, attention, or decision-making. The parameters are interpretable within the context of these functions, enabling researchers to gain insights into various aspects of cognitive processes under different experimental conditions.

Linear Models
Linear and logistic regression models are the simplest yet most interpretable in the modeling toolkit. They provide an initial framework for understanding the relationships among variables. Their coefficients indicate the change in the outcome variable per unit change in predictor variables, offering a direct interpretive advantage. However, their simplicity is a double-edged sword. While they are transparent, they often fall short in capturing the complex, non-linear relationships that frequently influence human behavior. In addition, the fact that they are interpretable does not mean that provide adequate explanations. Linear models are not able to capture the underlying cognitive processes that govern behavior.

1.4 Approach in this book

The remainder of the book will present a number of computational models to explain simple behaviors. The primary goal is to provide students with hands-on experience in terms of the implementation, running, and evaluation of computational models. To make the models more tractable and amenable for simulation, some of the "bells and whistles" from some key models have been simplified or removed.

Data and Code

The data for the exercises and example code can be found here: `https://www.dropbox.com/sh/h06mg6e6r59u1od/AACZcWeux_EZOSArvS-59RrIa?dl=0`

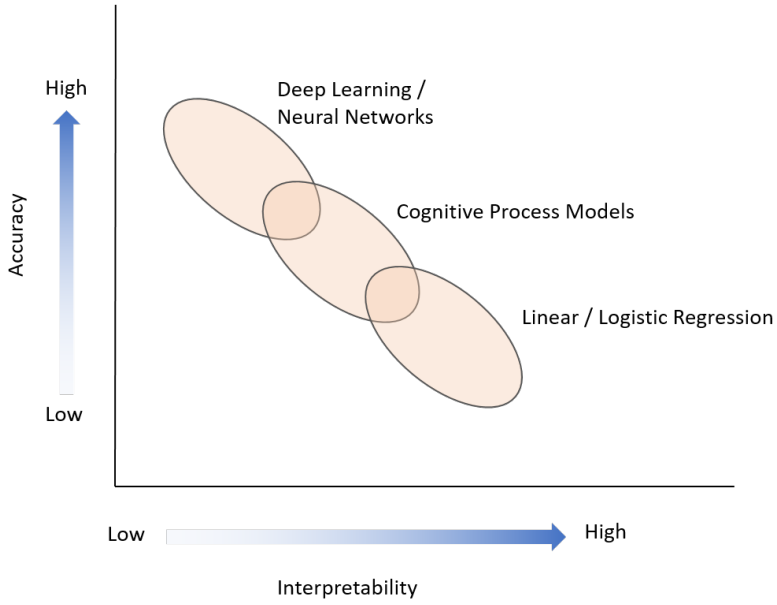

Figure 1.1: Trade-off between accuracy and interpretability between various types of models

Exercise 1.1 "All models are wrong, but some are useful" is a variant of a phrase by the statistician George Box[a]. What do you think it means and how does this phrase apply to the computational modeling of behavior? ■

[a] Box, George E. P.; Norman R. Draper (1987). Empirical Model-Building and Response Surfaces, p. 424, Wiley.

Exercise 1.2 How do the objectives for computational modeling differ between AI researchers and cognitive science researchers (typically)? ■

Exercise 1.3 What are "counterfactuals," and how can computational models be used to explore them? ■

Further Reading

Battleday, R. M., Peterson, J. C. and Griffiths, T. L. (2021). From convolutional neural networks to models of higher-level cognition (and back again), *Annals of the New York Academy of Sciences* **1505**(1): 55–78.

Calder, M., Craig, C., Culley, D., de Cani, R., Donnelly, C. A., Douglas, R., Edmonds, B., Gascoigne, J., Gilbert, N., Hargrove, C. et al. (2018). Computational modelling for decision-making: where, why, what, who and how, *Royal Society open science* **5**(6): 172096.

Fum, D., Del Missier, F. and Stocco, A. (2007). The cognitive modeling of human behavior: Why a model is (sometimes) better than 10,000 words, *Cognitive Systems Research* **8**(3): 135–142.

Guest, O. and Martin, A. E. (2020). How computational modeling can force theory building in psychological science.

Kriegeskorte, N. (2015). Deep neural networks: a new framework for modeling biological vision and brain information processing, *Annual review of vision science* **1**: 417–446.

Wilson, R. C. and Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data, *Elife* **8**: e49547.

2. Modeling Response Times

Behavioral experiments often measure how quickly a person can respond under different experimental conditions. Faster average response times in one condition may suggest an advantage in information processing. For instance, in memory tests, quick response times can offer clues about the speed of information retrieval. Similarly, in tasks requiring perceptual decision-making, quick responses may indicate efficient information gathering from visual cues.

Two fundamental aspects of response times need consideration. First is the element of variability. Even under identical stimulus and experimental conditions, response times among participants are likely to differ. A computational model for response times should account for this randomness. Second is the speed-accuracy tradeoff: Faster responses can often mean reduced accuracy. This principle is evident in real-world situations like game shows, where hasty answers may be incorrect. In addition, in many lab-based studies, researchers can adjust instructions to focus on either speed or accuracy, leading to a decrease in accuracy or an increase in response times, respectively.

Figure 2.1 displays sample response time data from a single user engaged in the "Ebb and Flow" game on Lumosity's cognitive training platform. This rapid task-switching game rewards both speed and accuracy. Interestingly, the data reveals no speed-accuracy tradeoff during training. Instead, the user maintains high accuracy rates (averaging 96% correct) through both training phases while becoming faster by about 70ms. Moreover, errors are slightly faster than correct responses. How can we account for this unique combination of accuracy and speed?

This chapter delves into computational models based on noisy evidence accumulator models. The core concept behind these models is that making a decision involves accumulating evidence over time, and this process is inherently subject to variability or "noise." To introduce the foundational concepts, we'll first simulate a basic random walk model.

2.1 Random Walk

To begin, we'll explore the basic random walk model as a straightforward example of evidence accumulation. In this model, an imaginary "walker" moves either forward or backward based on incoming evidence. This continues until the walker reaches a predetermined decision point or threshold.

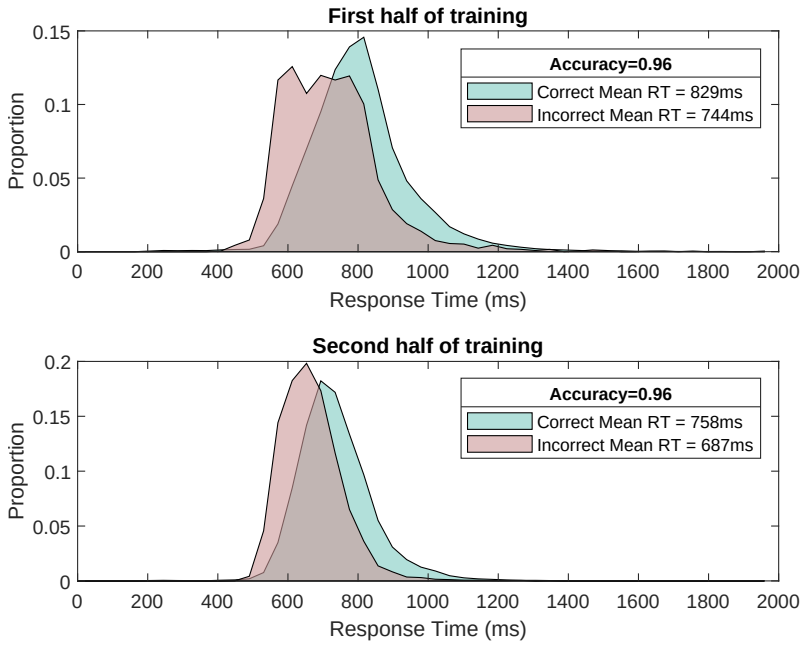

Figure 2.1: Response time distributions for correct (green) and error responses (red) for a a single participant training on task switching on the Lumosity platform. The results are split by the first and second half of training. Data is taken from the user with ID=2 from the file `data_sim7001.csv` from the OSF repository `https://osf.io/sxr5f/`

Imagine the walker starts at a neutral position. Before each step, a fair coin is flipped. A "heads" result moves the walker one step forward (+1), while "tails" sends the walker one step backward (-1). Each flip of the coin dictates the next move, making this a one-dimensional random walk constrained to integer steps. Figure 2.2a illustrates sample paths this walker could take.

In a variant of this model, steps are not uniform but are instead determined by random draws from a Gaussian distribution with a mean of 0 and a standard deviation of 1. This variant is visualized in Figure 2.2b.

So, what can we predict about the walker's position after t steps? While each individual step is random, certain patterns become evident when the process is repeated multiple times. To explore these patterns, we'll use Monte Carlo simulations (see Box 2.1.1). These simulations aim to predict the outcomes of random processes like our random walk. By running the walk multiple times, we can then compute average outcomes or visualize the distribution of all possible outcomes.

2.1.1 Exercises

Exercise 2.1 Implement the random walk algorithm with discrete steps and simulate this algorithm for a number of restarts from the same position. For simplicity, let's assume that the initial position is 0 and each step is either a +1 or -1 step (with equal probability). Graph the position at each step and superimpose the separate random walks on the same graph. Your graph should look like Figure 2.2a (note that the visual details of the graph, including line color are not important here). ■

Box 2.1.1: Monte Carlo Simulations

The term *Monte Carlo* is named after the famed casino in Monaco, reflecting the method's reliance on randomness and chance. In essence, a Monte Carlo simulation is a computational algorithm that provides numerical results by repeatedly simulating random samples from known distributions. It is often used in situations where analytical solutions are hard to obtain or too complex to be easily understood. The technique has a wide range of applications, from finance and engineering to artificial intelligence and environmental science.

To illustrate, let's consider a simple example: estimating the value of π. Imagine we have a circle inscribed within a square. The ratio of the area of the circle to the area of the square is $\frac{\pi}{4}$. To approximate π, we could randomly toss darts at the square and then count the number that land inside the circle. The ratio of darts that hit the circle to the total number of darts tossed should approximate $\frac{\pi}{4}$. Multiply that ratio by 4, and you get an estimate of π. The more darts you throw, the closer you should get to the actual value of π.

In much the same way, Monte Carlo simulations can be used to model more complex systems and processes, like our random walk model. By repeatedly simulating these systems, we can explore various outcomes and make more informed predictions.

Exercise 2.2 Modify your program to seed the random number generator at the start such that the random walks have the same trajectories on every run of your program (i.e., always producing the same figure in the end). How do you seed the random number generator in your programming language? Can you think of scenarios when is it useful to ensure that your simulation generates the same random sequence every time you restart it? ■

Exercise 2.3 Suppose we repeat the random walk process infinitely many times. What is the most likely position after each time step? In other words, if we had to guess the location, what would our best guess be? More formally, this question is about the *expectation* of the position at any step.

- [a] > 0
- [b] < 0
- [c] 0
- [d] Impossible to determine, as this depends on number of steps (t) taken

■

Exercise 2.4 Modify your program to a Gaussian random walk. At each iteration of this random walk, draw a random number drawn from a standard normal distribution (i.e., with mean 0 and standard deviation of 1). This random number is the step size. Visualize the results of a few Gaussian random walks to produce a graph like Figure 2.2b ■

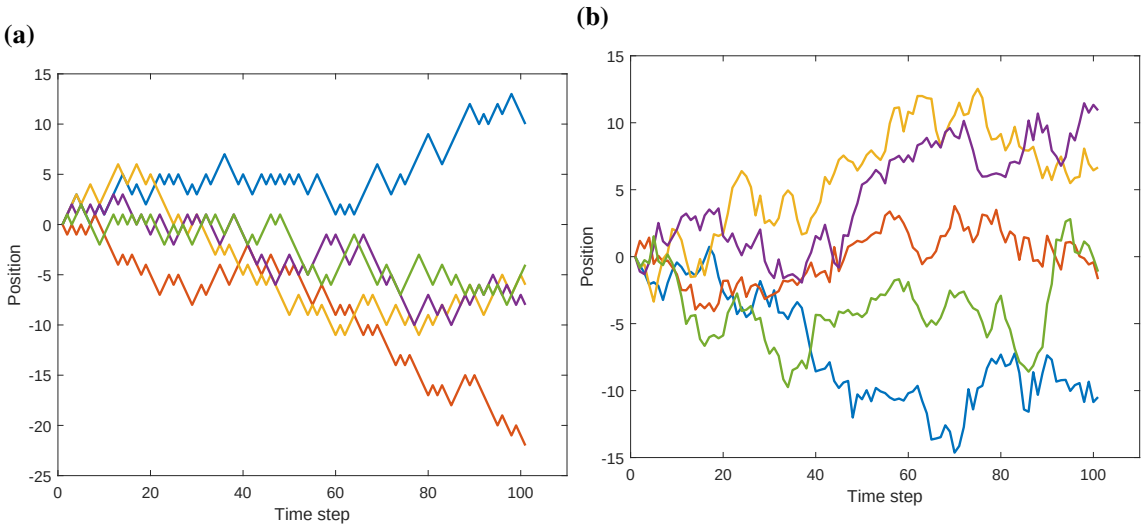

Figure 2.2: Examples of integer random walks (a) and Gaussian random walks (b).

Exercise 2.5 If we generate an infinite number of Gaussian random walks, what is the expected standard deviation of the position as a function of t? Hint: the sum of two random variables x and y that are drawn from independent normal distributions is also normally distributed. For example[a], if $x \sim N(\mu_x, \sigma_x)$ and $y \sim N(\mu_y, \sigma_y)$, then $x+y \sim N(\mu_x+\mu_y, \sqrt{\sigma_x^2+\sigma_y^2})$

a t
b $\sqrt{t}$
c t^2
d 1

[a] in the following notation, $x \sim N(\mu_x, \sigma_x)$, should be read as "x is distributed as a normal distribution with mean μ_x and standard deviation σ_x

2.2 Drift Diffusion Model

Building on the random walk model, we introduce the drift diffusion model as a more specialized framework commonly employed to analyze both response time and accuracy in two-alternative forced-choice tasks—tasks that are frequently featured in psychological experiments. In this model, the position of the random walk is interpreted as the cumulative evidence favoring one of two possible responses.

A high value is evidence for one response and a low value is evidence for the other response. A decision is made when sufficient evidence has been accumulated for one of the alternatives. This is modeled by setting two thresholds (barriers) on the random walk that represent the two response alternatives. If the random walk reaches one of response thresholds, the random walk terminates and a decision is made associated with the response threshold. In addition, the time at which the threshold is reached determines the response time.

Figure 2.3 shows some example response trajectories. The trajectories shown in blue and orange end up hitting the blue and orange thresholds respectively. The circles indicate the time at which the threshold was reached. Note that the random variation in the accumulation process leads to variation in response times. If many large steps are taken in one direction (say the upper

threshold), the upper threshold will be reached quickly leading to a fast response time. On the other hand, it is also possible that many of the random steps cancel each other out for a while causing the walk to terminate after many steps. Note that some of the trajectories might terminate outside the window in the figure (but all walks in this process will terminate at some point).

2.2.1 Simulating the basic drift diffusion model

There are a a number of methods available to simulate the drift diffusion model. Here, we will focus on a simple method based on the random walk we introduced earlier which operates in discrete time[1]. The algorithm for the basic drift diffusion model is illustrated with pseudocode in Algorithm 2.1 and is illustrated conceptually in Figure 2.4. The variable x represents the amount of evidence for one of two alternatives. The basic algorithm starts with initial evidence value $x = z$ and accumulates evidence over time. At each step, a random step ε is taken where ε is sampled from a normal distribution with mean v and standard deviation σ. The parameter v is the *mean drift rate*. This key parameter determines the overall direction of x. If the drift rate is positive, there is a drift towards the upper boundary. On the other hand, if the drift rate is negative, there is a drift towards the lower boundary. There are two response thresholds: the upper threshold at a and a lower threshold at 0. If at any point in time the accumulated evidence x reaches the lower or upper threshold, the process stops and a choice is made. The time taken to reach one of the thresholds is the response time.

Note that in this algorithm, we assume that each individual step of the random walk corresponds to a small increment in time (Δt). We will assume that $\Delta t = 1$ and that time is expressed in milliseconds. Therefore, if on a particular trial the process terminates after 700 steps, 700ms has elapsed. Setting Δt to other values can make the simulations more or less granular and also affects the overall computation time (i.e., time to run your simulation, not the response time predicted by the model).

Also note that the model assumes that time starts at t_0. This time represents the amount of time that has already passed before the process of evidence accumulation starts and/or any processing time after the threshold is reached such as motor response time. This *non-decision time* t_0 therefore reflects any additional processing time that is not related to evidence accumulation.

The two possible responses in this process can represent the two types of responses in a two-alternative forced choice task (e.g., “recognize the face”, “do not not recognize the face” in a

[1] the discrete time simulation is an approximation of the diffusion process which operates in continuous time

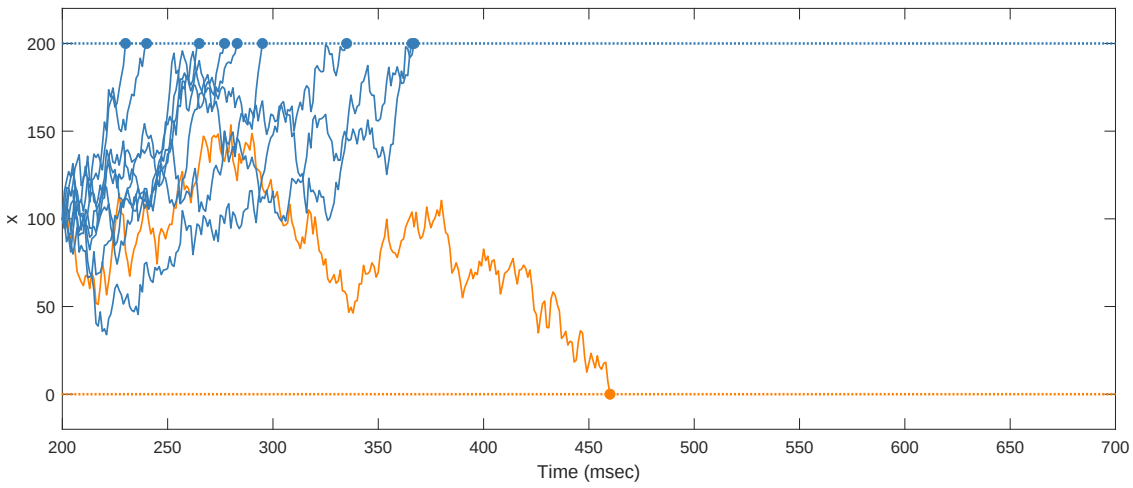

Figure 2.3: Examples trajectories of a drift-diffusion model

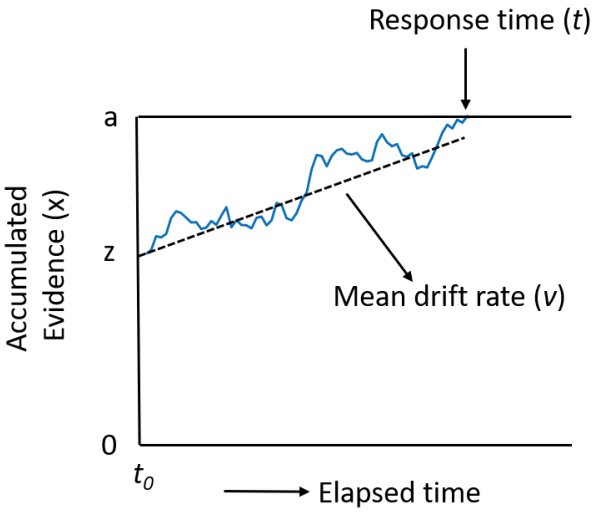

Figure 2.4: Illustration of the basic drift diffusion model

memory experiment). Another way to code the responses is in terms of correct responses (upper threshold) and incorrect responses (lower threshold). We will assume the latter coding in the exercises below.

2.2.2 Exercises

For the exercises, implement the basic drift diffusion model from Algorithm 2.1. Simulate the basic drift diffusion model for 1000 trials with the following parameters: $v = 0.5$, $\sigma = 7$, $z = 100$, $a = 200$, $\Delta t = 1$, $t_0 = 200$. On each trial, you simulate a different trajectory of evidence accumulation that results in some response time and some outcome y. You can assume that reaching the upper threshold represents a correct decision ($y = 1$), and the lower threshold an incorrect decision ($y = 0$)[2]

Exercise 2.6 What is the accuracy of this simulated choice process[a]? In other words, what is the probability that the outcome is the correct decision ($y = 1$)?

a $0.65 - 0.75$
b $0.75 - 0.85$
c $0.85 - 0.95$
d $0.95 - 1.00$

[a] With a 1000 simulated trials, you should be able to estimate accuracy sufficiently well to identify the correct answer

[2] Keep in mind that once a random walk hits one of the thresholds, the random walk should terminate. Therefore, in your code, make sure that at this point, the random walk does not proceed. Otherwise, it is theoretically possible that the random walk hits the other threshold at a later time.

Algorithm 2.1 Simulating a single trial from the basic drift diffusion model. Note that in the pseudocode below, the arrow symbol "←" means "is assigned to" and can be translated to "=" in many programming languages

INPUT: v (mean drift rate), σ (standard deviation of drift rate within a trial), z (starting point), a (upper threshold), and Δt (change in time at each step), t_0 (non-decision response time)

OUTPUT: y (choice), t (response time)

function DRIFTDIFFUSIONMODEL(v,σ,z,a,Δt, t_0)
  $x \leftarrow z$ ▷ Initialize evidence x to starting position z
  $t \leftarrow t_0$ ▷ Initialize elapsed time to t_0
  while $x > 0$ **and** $x < a$ **do** ▷ Continue the random walk if boundaries have not been reached
    $\varepsilon \sim N(v, \sigma)$ ▷ Sample a step size ε from a Normal distribution
    $x \leftarrow x + \varepsilon$ ▷ Update the evidence with the step size
    $t \leftarrow t + \Delta t$ ▷ Update the elapsed time
  end while
  if $x \leq 0$ **then**
    $y \leftarrow 0$ ▷ Code the choice as 0 if the lower bound was reached
  else if $x \geq a$ **then**
    $y \leftarrow 1$ ▷ Code the choice as 1 if the upper bound was reached
  end if
  return (y,t) ▷ Return the choice and response time
end function

Exercise 2.7 What is average response time of this simulated choice process? Keep in mind that each step in the random walk takes $\Delta t = 1$ milliseconds (ms) and that you need to add t_0 to the elapsed time of the random walk to determine the response time (i.e., line 3 of Algorithm 2.1).

a $100 - 200$ms
b $200 - 300$ms
c $300 - 400$ms
d $400 - 500$ms
e $500 - 600$ms
f $600 - 700$ms

■

Exercise 2.8 What model parameter does *not* influence accuracy?

a t_0
b v
c z
d a

■

Exercise 2.9 Some individuals are more cautious than others and prefer accurate decisions over fast decisions. In the drift diffusion model, this is modeled by changes in the upper threshold a. Suppose we always start x halfway between the lower and upper threshold (i.e., $z = a/2$) and we raise the upper threshold (a) say from 200 to 300 simulating a more cautious choice process. What happens with the predicted response time and accuracy?

a response time decreases; accuracy decreases
b response time decreases; accuracy increases
c response time increases; accuracy decreases
d response time increases; accuracy increases

■

Exercise 2.10 Visualize the distribution of response times as illustrated in Figure 2.5. This figure was created on the basis of 10,000 simulated trials which will make the distribution look more smooth. If the computation time for 10,000 trials is too long, a figure based on 1000 trials will suffice. ■

2.2.3 Model extension: fast and slow errors**

In the basic drift diffusion model, response times for correct decisions are as fast as response times for incorrect decisions (assuming a starting position that does not bias any of the choice alternatives, i.e. $z = a/2$). However, there are situations where error responses tend to be fast or when errors tend to be especially slow. The drift diffusion model can be extended to produce differences in the response times for correct and incorrect responses. The full drift diffusion model incorporates the following two ideas:

1. Introduce variability in the starting point z across trials. The idea is that on some trials, the starting point is near one of the boundaries. For example, if z on a particular trial starts near the lower boundary (incorrect decision), it is more likely that the lower boundary is reached quickly, leading to fast errors.

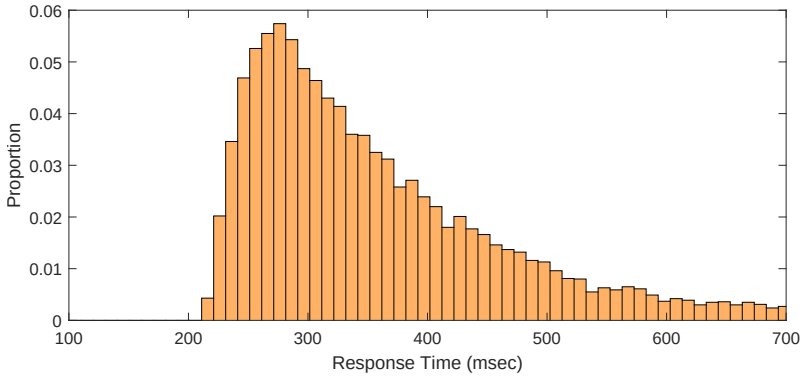

Figure 2.5: Distribution of response times

2. Introduce variability in the drift rate v across trials. If the mean drift rate v on one trial is slow and fast on another trial, this will lead to slow errors.

The algorithm for the full drift diffusion algorithm is shown in Algorithm 2.2.

2.2.4 Exercises (extra credit)

Exercise 2.11 For this exercise, implement the full drift diffusion model from Algorithm 2.2. Simulate the drift diffusion model for 1000 trials with the following parameters: $v = 0.5$, $\sigma = 7$, $z = 100$, $a = 200$, $\Delta t = 1$, $t_0 = 200$, $s = 140$, $\eta = 0$. With this parameter setting, there is *variability in starting point* but no across-trial variability in mean drift rate. You can assume that reaching the upper threshold represents a correct decision ($y = 1$), and the lower threshold an incorrect decision ($y = 0$). What is the predicted average response time for correct and incorrect decisions?

a Correct: 325-375ms; Incorrect: 275-325ms
b Correct: 375-425ms; Incorrect: 350-450ms
c Correct: 275-325ms; Incorrect: 325-375ms
d Correct: 275-325ms; Incorrect: 375-425ms

■

Exercise 2.12 Simulate the model as in the previous exercise but with the following change in parameters: $s = 0$, $\eta = 0.5$. With this parameter setting, there is *across-trial variability in mean drift rate* but no variability in starting point. What is the average response time for the correct and incorrect decisions?

a Correct: 325-375ms; Incorrect: 275-325ms
b Correct: 325-375ms; Incorrect: 350-450ms
c Correct: 275-325ms; Incorrect: 325-375ms
d Correct: 275-325ms; Incorrect: 375-425ms

■

Potential data sets to explore

- **Lumosity Task Switching Data** (`https://osf.io/sxr5f/`). This OSF repository contains the data and code from: *Steyvers, M, Hawkins, G., Karayanidis, F. and Brown, S. (2019). The Dynamics of Task Switching: Modeling Practice and Age Effects in Large-Scale Data. Proceedings of the National Academy of Sciences*. The data set includes the gameplay data

Algorithm 2.2 Simulating a single trial from the full drift diffusion model. Note that lines 1-5, and 7 have changed relative to the algorithm of the basic drift diffusion model.

INPUT: v (mean drift rate), η (standard deviation of drift rate across trials) σ (standard deviation of drift rate within a trial), z (starting point), a (upper threshold), and Δt (change in time at each step), t_0 (non-decision response time), s (variability in starting point)

OUTPUT: y (choice), t (response time)

1: **function** DRIFTDIFFUSIONMODEL(v,η,σ,z,a,Δt,t_0,s)
2: $r \sim \text{Uniform}(0,1)$ ▷ Sample a random number uniformly between 0 and 1
3: $x \leftarrow z + s \times (r - 0.5)$ ▷ Initialize evidence x between $z - 0.5s$ and $z + 0.5s$
4: $t \leftarrow t_0$ ▷ Initialize elapsed time to t_0
5: $v_{step} \sim N(v, \eta)$ ▷ Sample a mean drift for this trial
6: **repeat**
7: $\varepsilon \sim N(v_{step}, \sigma)$ ▷ Sample a step size ε from a Normal distribution
8: $x \leftarrow x + \varepsilon$ ▷ Update the evidence with the step size
9: $t \leftarrow t + \Delta t$ ▷ Update the elapsed time
10: **until** $x \leq 0$ **or** $x \geq a$ ▷ Stop when the lower boundary (0) or upper boundary (a) has been reached
11: **if** $x \leq 0$ **then**
12: $y \leftarrow 0$ ▷ Code the choice as 0 if the lower bound was reached
13: **else if** $x \geq a$ **then**
14: $y \leftarrow 1$ ▷ Code the choice as 1 if the upper bound was reached
15: **end if**
16: **return** (y,t) ▷ Return the choice and response time
17: **end function**

for a sample of users who play Ebb and Flow, a task switching game on Lumosity. This is a game designed to test the ability to switch between different tasks. The raw data is described at the individual trial level (i.e., individual decisions within a particular gameplay event) and include response time, accuracy, as well as identifiers that describe the particular stimulus display and type of condition associated with the trial.

Further Reading

Ratcliff, R. and Smith, P. (2015). Modeling simple decisions and applications using a diffusion model, *The Oxford Handbook of Computational and Mathematical Psychology*.

3. Modeling Performance Improvements

Typically, practicing a certain skill leads to performance improvements. One becomes faster and/or more accurate in performing a task. In his pioneering work, Ebbinghaus (1885) showed how repeated practice on memorizing lists of nonsense syllables led to successive improvements. When he repeated the same task over and over again, he became better at memorizing the lists, and was less likely to forget the items. Similarly, complex skills such as mathematical problem solving or driving all show a similar progression of performance over time: initial practice leads to rapid performance improvements and later phases of practice show more gradual improvements.

In addition, a lack of practice might lead to a loss of skill, at least temporarily. For example, anybody who hasn't driven a car for a few years, or hasn't solved certain types of mathematical problems for a while, will see a degradation in initial performance when performing these tasks again. However, with some additional practice, performance might be back at previously achieved levels. This effect is known as the warmup effect.

Many theories in psychology have been proposed to explain this progression of performance as a function of practice and retention with descriptive models such as exponential and power law learning functions (Newell and Rosenbloom; 1981; Evans et al.; 2018; Heathcote et al.; 2000) as well as cognitive architectures such as SOAR (Laird et al.; 1987) and ACT-R (Anderson and Lebiere; 2014). In this chapter, we will focus on two different modeling approaches. First, we will introduce some descriptive models for skill acquisition. The reason to start with these models is simplicity. It allows us to get started with some simple cognitive models and get some experience with fitting models to data. Second, we will introduce a model proposed by Anderson et al. (1999) based on the ACT-R framework that can explain not only the effect of practice but also the effect of delay between practice.

For example data, we will turn to data from Lumosity, an online cognitive "brain-training" platform. Lumosity provides a number of different games for users that are intended to tap memory, attention, flexibility, speeded processing, and problem solving. Many of these games are based on well-known tasks from cognitive psychology. Millions of people play these games over extended periods of time, providing a very rich platform on which to study skill acquisition (Steyvers and Benjamin; 2019; Steyvers and Schafer; 2020; Steyvers et al.; 2019; Kumar et al.; 2022).

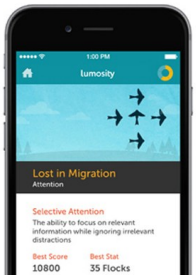

Figure 3.1: The Lumosity game "Lost in Migration"

To illustrate this model, we apply the model to Lumosity data from the game "Lost in Migration" as shown in Figure 3.1. This is a selective attention game inspired by the Eriksen flanker task (Eriksen and Eriksen; 1974). The goal is to respond to the direction of the target (a bird) and ignore the direction of distractors that flank the target. During each trial, the target and distractors are arranged in different spatial layouts. Users have use the arrow keys to indicate which direction the target is pointing; the layout and orientation of the distractors varies from trial to trial.

Each gameplay event has a fixed duration of 45 seconds. At the end of each gameplay event, users are provided feedback on a number of performance metrics, such as mean response time per trial and mean accuracy. For our purpose, we assess performance on the basis of the total number of correct trials completed within the 45 second time period. All of the following illustrations and modeling will use this performance score.

Typically, performance on this task improves over successive gameplays. Figure 3.2 shows practice curves for three illustrative users (the data repository for this book includes data from 247 users). Note the rapid improvement early in training followed by diminishing returns. Let's see how we can capture the trends in the data, starting with descriptive models.

3.1 Descriptive models

The goal of descriptive models for skill acquisition is simply to characterize the mathematical relationship between performance and amount of practice. These descriptive models do not provide much of an explanation of *why* performance improves in terms of underlying changes in cognitive representations and processes. However, the simplicity of these models is appealing and provides a good starting point for modeling. In addition, because these models are mathematically precise, and can be simulated using computer code, we can use these models to make predictions about future behavior.

One simple model for describing learning curves is the *exponential model*:

$$y = a - (a - u)e^{-c \times t} \tag{3.1}$$

In this model, y represents the predicted performance, and t is a representation of the amount of practice in terms of the number of times a task has been practiced. We will use a convention that the start of practice will be defined as $t = 0$.

There are three *parameters* in this model: the asymptote parameter a that represents the highest level of performance that can be reached after extended practice, the intercept parameter u that represents the baseline performance at the start of practice (when t=0) and the learning rate parameter c, which captures the rate at which learning progresses towards the asymptote.

What can we do with this descriptive model? The exponential model for learning is a very simple model that describes how performance changes as a function of practice. There are three "unknowns" in this model corresponding to the three parameters. If we know what these parameter

values are, we can make very precise predictions about how learning proceeds over time. For example, the value of the asymptote parameter will directly tell what the expected performance is after extended practice. An important part of computational modeling is to *fit* the model to data in order to get parameter estimates that provide good descriptions of the data. In the next section, we will go into one simple method of fitting based on *least-squares*.

3.1.1 Least-squares model fitting

The red lines in Figure 3.2 shows some examples of the predictions of the exponential learning model. Note that in this application, we have assumed that gameplay is used to represent t and the score is used to represent y. But how did we get the model to line up with the observed data? In other words, how did we find the parameters a, u and c for each individual user that best fits the data?

There are many approaches to model fitting and many of these approaches are outside the scope of this book. For our purposes, we will focus on the simplest approach for fitting models based on *least-squares*. The basic idea is that we want to minimize the deviations between what the model predicts and what is actually observed. One way to define the *goodness-of-fit* is the squared error that penalizes large deviations. Specifically, in least-squares model fitting, the goal is to find parameter values that minimize the mean squared error (MSE)[1]:

$$\text{MSE} = \frac{1}{N}\sum_{i}^{N}(y_{observed,i} - y_{predicted,i})^2 \tag{3.2}$$

In this definition, we are assuming that the data we want to fit has N observations. In the Lumosity data, N varies from user to user depending on how active they are in playing games. The model makes predictions $y_{predicted}$ and the observed data is $y_{predicted}$. The index i is used to refer to individual observations in the data (e.g., the individual circles in Figure 3.2).

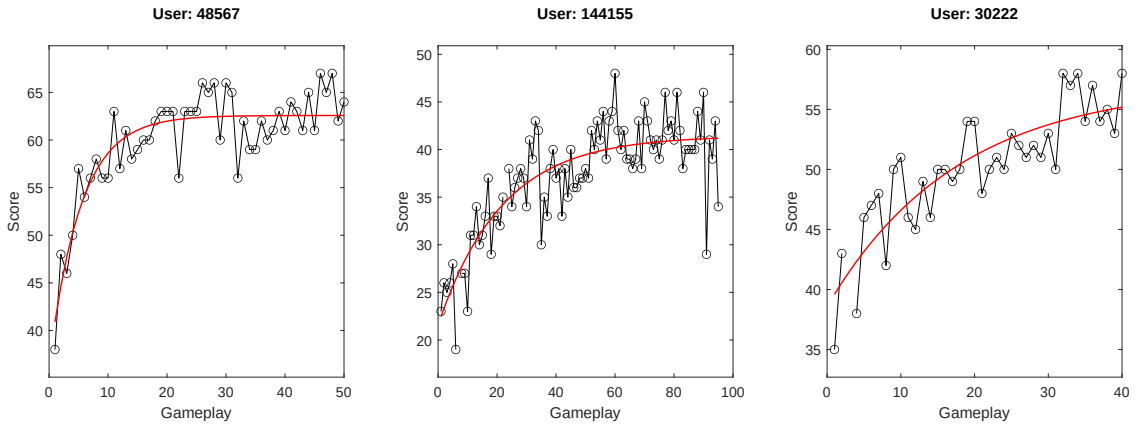

Figure 3.2: Practice curves for three illustrative Lumosity users (black line) with best fitting curve from the exponential model (red line). The user ids are arbitrary identifiers to represent individual Lumosity users. Note that the exponential model is fit separately for each user.

What is happening exactly when we use this least-squares fitting procedure in order to fit a model to data? In the fitting procedure, one starts with an initial guess about the parameter values. These initial values don't necessarily need to be good guesses, as long as the parameter values

[1]The MSE is closely related to other evaluation metrics such as the root mean squared error ($RMSE = \sqrt{MSE}$). Switching from MSE to RMSE might change the estimates but does not change which parameter estimates achieve the global minimum

are valid for the model. For example, the learning parameter c can only take positive values and therefore, the initial value for c has to be positive.

With the starting values, the fitting procedure can calculate the initial MSE. Next, the fitting procedure starts to look around the vicinity of the starting point to see if there is some new set of parameter values that will lead to smaller MSE values, which indicates a better fit. Once a new set of parameter values is chosen, the procedure again looks around this new set to see if there is some way to make a new "move" to minimize the MSE even further. This process is repeated until there is no way to further improve the MSE.

This iterative process is visualized in Figure 3.3 where the exponential model is applied to the data from the leftmost user (48567) in Figure 3.2. To simplify the visualization, we have assumed that the c parameter is already optimized and the goal is to find best fitting parameter values for the a and u parameters. The surface shows how the MSE depends on the values for these two parameters. The bowl-shaped surface shows that there is a set of parameter values a and u that minimizes the MSE. The least-squares fitting procedure starts at the point indicated in the figure (this was the initial guess) and iteratively improves the MSE by "sliding" down the surface (a useful analogy is a ball rolling down this surface and coming to rest of the bottom of the bowl). More technically, this procedure follows the gradient of the surface and always tries to find the direction of the steepest descent. Most (but not all) data fitting procedures are based on this idea of *gradient descent*.

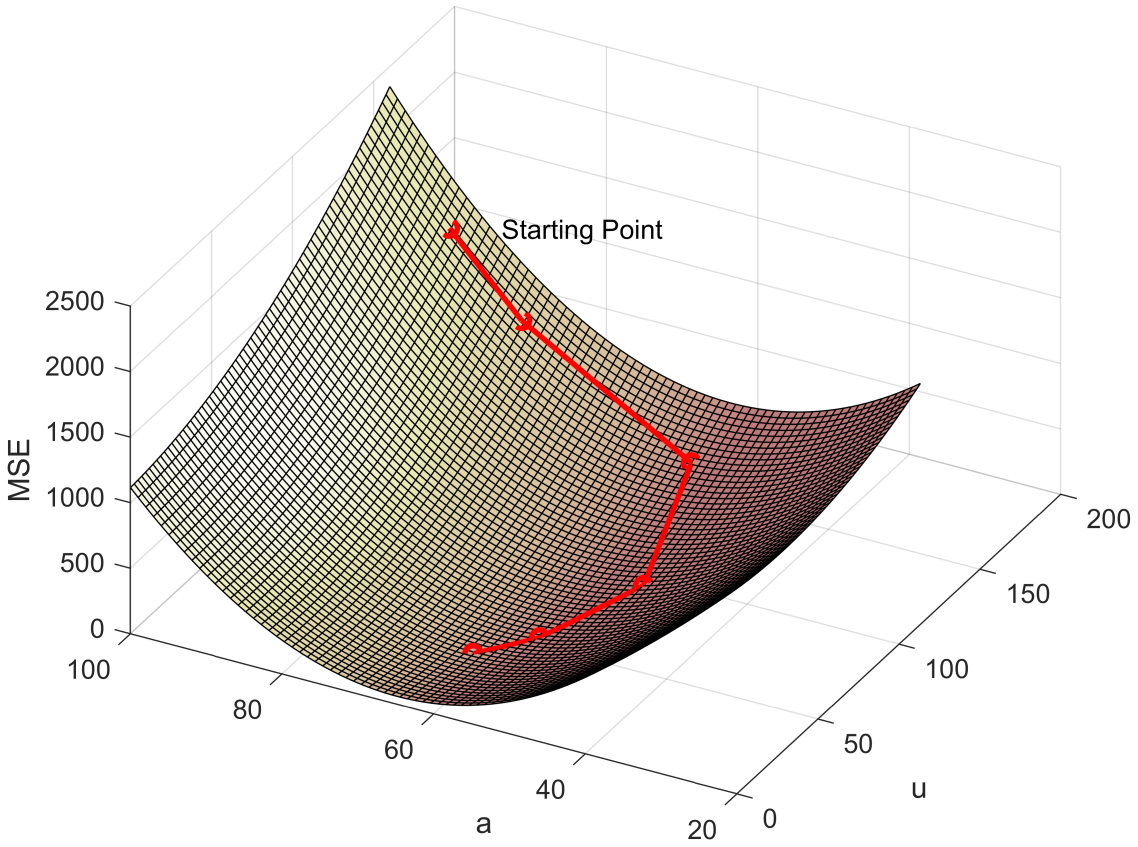

Figure 3.3: Illustration of the process of finding the best fitting parameters through gradient descent. The line shows the progression of parameter values found by gradient descent. The last point shows the final estimates identified by the procedure where the MSE can no longer be improved.

3.1.2 Exercises

Exercise 3.1 Apply a least squares fitting procedure applied to the data from user 30222 (corresponding to the file `LumositySample_user30222.csv`)[a]. You can use the column `gameplay` for the amount of practice t and column `score` for the performance y. Create a figure that shows the predicted scores from the exponential model (based on the best fitting model) as well as the observed data for user 30222. ■

[a] Note that this file has some missing observations for the scores (coded as `NaN` in the file). You can ignore these missing observations in the calculation of the mean squared error (or simply remove these data cases)

Exercise 3.2 For the exponential learning model in the previous exercise, what is the predicted asymptotic performance level? In other words, what does the model predict for the score after an infinite amount of practice?

- [a] < 55.25
- [b] 55.25-55.75
- [c] 55.75-56.25
- [d] 56.25-56.75
- [e] 56.75-57.25
- [f] 57.25-57.75
- [g] > 57.75

■

Exercise 3.3 At the end of training, user 30222 has practiced for 40 gameplays. How many *additional* gameplays would be required according to the best fitting exponential model to achieve a score of at least 70?

- [a] 1-10
- [b] 11-100
- [c] 101-1000
- [d] 1001-10,000
- [e] > 10,000
- [f] Impossible, this score will never be reached

■

Exercise 3.4 Now let's consider a slightly different descriptive model based on a power-law: $y = a - (a-u)t^{-c}$. Apply the power-law model to the data from user 30222. What is the predicted asymptotic performance level for this model?

- [a] < 55.25
- [b] 55.25-55.75
- [c] 55.75-56.25
- [d] 56.25-56.75
- [e] 56.75-57.25
- [f] 57.25-57.75
- [g] > 57.75

■

Exercise 3.5 Which model provides the best fit in terms of MSE?

a Exponential learning model

b Power-law learning model

■

Exercise 3.6 ** Modify your least-squares fitting code to constrain the possible parameter values. Specifically, enforce constraints such that parameters a, u, and c are all positive. ■

Exercise 3.7 ** In most cognitive modeling applications, it is desirable to fit the model to individual subject data. In this case, each individual would be associated with its own set of parameters. However, if there are only few observations for each individual, it might be challenging to get good estimates for parameters such as the learning rate parameter c. One solution is to share some parameters across individuals. For example, all individuals have the same parameter c but each individual has their own parameters a and u. Apply this modeling approach to the data from users 48567, 144155 and 30222 and modify your least-squares fitting code accordingly. Note that you will now have $(3 \times 2) + 1 = 7$ parameters to fit. ■

Exercise 3.8 ** Power-laws can be used to analyze not only performance improvements in humans but also AI models. One widely used AI algorithm is the Large Language Model (LLM), which is trained on an extensive corpus of text documents. The primary objective of an LLM is to accurately predict the subsequent word in a given text sequence. Due to the considerable computational resources required for training, it's crucial to understand how performance gains in these models correlate with both model size, assessed by the number of parameters (N), and the volume of training data, represented by the number of text tokens (D).
In a specific study by Hoffmann et al. (2022), the predictive error (L) of a particular LLM on a given dataset was described by the power-law equation: $L = 1.61 + (406.4)N^{-0.34} + (410.7)D^{-0.28}$. The constant 1.61 indicates the inherent variability in the dataset; perfect prediction is unattainable even with an ideal model. The terms that include N and D reveal that increases in either variable result in a decline in prediction error, signifying improved performance.
Assuming researchers wish to practically implement this scaling law, let's consider they have already trained a model with specific values: $N = 10^9$ and $D = 10^9$. They are now evaluating their next steps. What option leads to the most improvements in prediction error according to this scaling law:

a Increase the model's size (N) twofold, while maintaining a constant data size (D).

b Double the data size (D), with the model size (N) remaining the same.

c Insufficient information to make a decision.

■

3.2 AFD model

The descriptive models such as the exponential learning model seemed to provide a decent fit of the observed data, but as you have noticed in Figure 3.2, there are a few data points where the model predictions deviate substantially from the observed data. One limitation of the exponential model is that performance is only a function of one factor, the amount of practice. However, other factors such as the timing and spacing of practice also influences performance. Take a look at Figure

3.4. This shows the data from the same set of Lumosity users we have seen before. However, we now take a somewhat different look at the data by plotting the score as a function of continuous time (bottom row). Note that many of the circles (observed data) line up vertically. These are practice events that are occurring within a particular session where users practice the game several times in a row (separated by a few minutes or less). One obvious pattern is the "zig-zag" in the performance scores. Performance increases within a session but after extended periods without training, performance decreases again at the start of a new session.

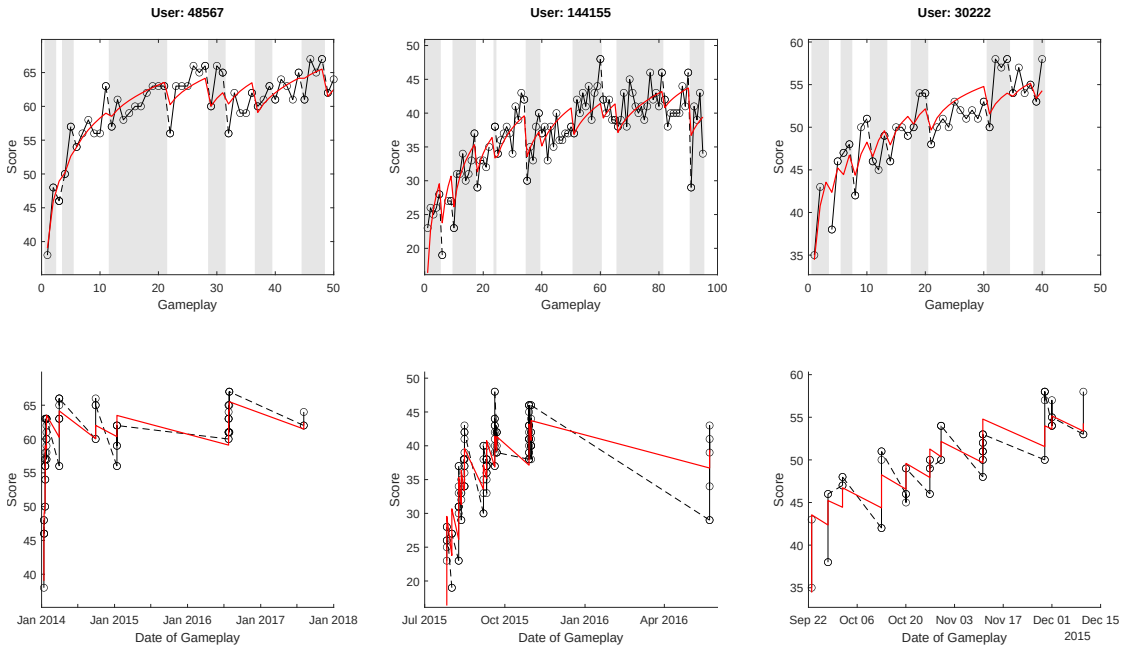

Figure 3.4: Practice curves for three illustrative Lumosity users (black line) with best fitting curves from the AFD model (red line). The top row shows performance as a function of gameplay. The shaded areas illustrate changes in sessions. The bottom row shows performance as a function of the date of gameplay.

We will now turn to the AFD model (named after the authors of the paper, Anderson, Fincham, and Douglass, 1999) to see if it can provide a better model fit, as well as a better explanation for the observed data. In this model, the assumption is that each individual practice event (e.g. an individual gameplay on Lumosity) creates a separate memory trace, that decays in strength over time. For example, after three practice events, there would be three memory traces, where the first memory trace has decayed the most. The accumulated strength across all memory traces is then based on a sum over all memory traces as follows:

$$strength_i = \sum_{j}^{n} (t_{i,j})^{-c} \tag{3.3}$$

where $strength_i$ represents the total strength at the (current) i-th practice event, and $t_{i,j}$ is the time that has passed between the j-th practice event and the current i-th practice event. The summation is over the n times the event has been practiced *before* the start of the current practice. Note that the term $(t_{i,j})^{-c}$ is a power-law decay with rate parameter c. Therefore, each memory trace contributes less to the overall strength with the passage of time. On the other hand, with more practice, there are memory traces to sum over which can increase the total accumulated strength. In contrast to the exponential model, this model predicts not only the effect of the amount of practice but also the

length of the retention interval. Practice events that are closely spaced in time are predicted to have a strong effect on the accumulated strength.

Let's see how this model works based on a simple example. Suppose a person practices three times, at times t=1, 2, and 4 days (let's assume for now that the units are expressed in days). Also, let's suppose $c = 0.1$. What is the accumulated strength at the start of the second practice event. This would be $strength_2 = (2-1)^{-.1} = 1$. At the start of the third practice event, we have $strength_3 = (4-1)^{-.1} + (4-2)^{-.1} = 1.83$.

To complete the model, we need two additional assumptions. First, we need to map the accumulated strength, which is a latent characteristic, to a performance score. For the AFD model, we assume the following transformation[2]:

$$y = u + b \times \log(1 + strength) \tag{3.4}$$

where u is an intercept parameter and b is a scaling parameter for the range in performance scores[3]

Another addition to the model is needed to explain the passage of time at longer time scales. In the Lumosity data, users sometimes practice several times in a session such that the practice events within a session are separated by several minutes. However, once a session is over, it might take days, weeks, or sometimes years before a user starts the next session. The model in Eq. 3.3 cannot handle these two different time scales very well. Anderson et al. (1999) proposed that the passage of time across sessions is handled differently. Specifically, the time elapsed $t_{i,j}$ between trials i and j is as follows:

$$t_{i,j} = \begin{cases} i - j, & \text{if } s_i = s_j. \\ i - j + (d_i - d_j)h, & \text{otherwise.} \end{cases} \tag{3.5}$$

In this definition, s_i is the session index associated with the i-th trial, $(d_i - d_j)$ is the number of days elapsed between the sessions for practice trials i and j, and h is a scaling parameter.

Therefore, within the same session (if $s_i = s_j$), traces age as a function of the number of the practice events (e.g. gameplays) that have elapsed. Between sessions, traces age not only as a function of the number of practice events that have occurred but also as a function of $(d_i - d_j)$, the number of days elapsed between sessions. With this definition, it is possible to set h such that the passage of time between sessions is much smaller than the time elapsed in terms of the total number of trials that could "fit" between sessions. Anderson et al. refer to this as the a "slowed-clock model"

Listing 3.1 shows pseudocode for to simulate the AFD model. The red lines in Figure 3.4 shows model fits for the three Lumosity users (with separate parameters for each user). Note how the model captures the qualitative "zig-zag" pattern in the data. Therefore, it seems to do a better job at fitting the data than the exponential data.

[2]the original AFD model was applied to response time data where practice leads to lower scores; the equation was modified in order to explain the Lumosity scores

[3]Note that in this model, there is no asymptote parameter. Theoretically, there is no upper bound of the performance score y.

Algorithm 3.1 Simulating the AFD model.

INPUT: u (baseline parameter), b (scaling parameter), c (rate parameter), h (scaling parameter), s (vector with session indices), d (vector with total days elapsed since start of practice)

OUTPUT: $y = (y_1, ..., y_n)$ (vector with predicted responses for n practice events)

1: **function** AFDMODEL(u,b,h,c,s,d)
2: **for** i=$1,\ldots,n$ **do** ▷ Loop over all n practice events
3: $strength \leftarrow 0$ ▷ Initialize strength to zero
4: **for** j=$1,\ldots,(i-1)$ **do** ▷ loop over all practice events before the i-th
5: **if** $s_i = s_j$ **then**
6: $t \leftarrow (i-j)$ ▷ Within a session, elapsed time is based on number of practice events elapsed
7: **else**
8: $t \leftarrow (i-j) + (d_i - d_j)h$ ▷ Between sessions, add an additional factor based on number of days elapsed between sessions
9: **end if**
10: $strength \leftarrow strength + t^{-c}$ ▷ Increment the total strength
11: **end for**
12: $y_i \leftarrow u + b \times \log(strength + 1)$ ▷ Map the total strength to a predicted score for the i-th practice event
13: **end for**
14: **return** y ▷ Return the vector of predicted scores
15: **end function**

3.2.1 Exercises

Exercise 3.9 What does the AFD model predict for the performance y after an infinitely long period without practice?

a 0
b u
c b
d $u + b\log(-d)$
e cannot be determined from the information provided

■

Exercise 3.10 Create code to apply a least squares fitting procedure for the AFD model to the data from user 30222 (corresponding to the file `LumosityData_user30222.csv`). [a]. For the practice indices i and j, you can use the data from column `gameplay`. For the observed score y, use the column `score`. For the session index s, use the column `session`. Finally, for the elapsed number of days since the start of training, use the column `dayselapsed`. If you need help with the code, there is some skeleton code (`afdskeleton`) in Python and Matlab on the code repository. ■

[a] Note that this file has some missing observations for the scores (coded as `NaN` in the file). Be careful about this missing data case. You can remove the missing data case as long as the elapsed time in terms of $(i - j)$ is not affected by the removal of this data case.

Exercise 3.11 Fit the AFD model to the data from user 30222. Create a figure that shows the predicted scores from the AFD model (based on the best fitting model) as well as the observed data. The horizontal axis should represent the number of gameplays. There is no need to embellish your graph as in Figure 3.4 with additional shaded areas to represent changes in sessions. ■

Exercise 3.12 What are the best fitting values that you found for parameters c, u, b, and h? What is the MSE? ■

Exercise 3.13 One of the implausible characteristics of the AFD model, as it is described in the text, is the amount of computation required to determine the strength after extended practice. In the Lumosity data, it is not uncommon for users to have thousands of practice events. Theoretically, to determine the accumulated strength at the 1001-th practice event, one has to sum over 1000 memory traces. Can you think of a solution to fix this model such that the number of memory traces to sum over does not grow linearly with the number of practice events?

■

3.3 Model Comparison based on Generalization Tests

We have now seen two models that can be applied to the same data. Which model is the best? What criteria do we use to compare models? Many approaches have been developed for model comparison and an in-depth treatment of this topic is outside the scope of this chapter. Here, we will introduce a simple and practical procedure based on *generalization tests*.

One possible approach to compare models, in the context of least squares model fitting, is to

compare the sum or mean of the squared deviations. In this approach, we select the model that produces the best fitting curve and the lowest squared deviations overall. This is a reasonable place to start model evaluation. The AFD model was able to capture some trends in the data (e.g. the "zig-zag") that the exponential model was unable to reproduce and this difference is reflected in the model fit based on MSE. Therefore, model fit to the observed data provides some useful information about the model.

However, the problem with this approach is that some models would do very well on this metric even though they would be terrible models for skill acquisition. For example, suppose we have n observed scores, and we introduce a model that has n parameters. In this model, the i-th parameter encodes the score for the i-th practice event. Applying this model to the data would lead to a perfect model fit — the sum or mean squared error would be zero. However, no researcher would take this model seriously. It does not really explain anything as the model corresponds to a lookup table and the ability of the model to fit the data is derived from the large number of parameters. Therefore, models that are extremely flexible can lead to good model fit of the observed data, and this flexibility is not always a good indicator of model adequacy.

The idea of a generalization test is that we test the ability of a model not to account for the data that is used to fit parameters, but its ability to account for *unseen* data. In other words, we want to assess the ability of the model to generalize. In the general setup of a generalization test, part of the data is used to fit the model. This is called the *training* data. The remaining part of the data, not used for model fitting (and in that sense, the data is "unseen") is used to evaluate the model. This is also known as the *validation* or *test* data[4]. We then evaluate the squared deviations only for the validation data. The best model is one with the lowest error on the validation data.

But how do we select the validation data? Figure 3.5 shows two possible approaches for some toy data set with 20 data points. One approach, illustrated in the left panel, is to randomly split the data into two partitions say of equal length, corresponding to a training set of 10 data points (filled circles) and a validation set of the remaining 10 data points (open circles). The red line show the best fitting exponential model. In this approach, we test the model's ability to generalize to data that is similar to the data used for model fitting. We will refer to this as an *interpolation* approach.

Another possibility is to test the ability of the model to generalize to data that is quite different from what is has seen. We will call this an *extrapolation* approach (Busemeyer and Wang; 2000). For example, in the skill acquisition modeling examples, we might care about the ability of the model to correctly predict the future performance of an individual. In the right panel of Figure 3.5, the 10 most recent gameplays were withheld from the model and used as validation data. As you can see, this is a challenging test for the model. It captures the trend for the training data (filled circles) but does not particularly generalize well. Of course, many models would be challenged by this particular generalization test. When this test is performed on a large data set with many individuals and many data points used for evaluation, it can become more clear which model is better at generalizing.

Keep in mind that extrapolation can be done in many different ways. For example, one can test the ability of a model to generalize to new individuals, to new experimental conditions, etc. An appealing feature of this generalization test is that it maps to real-world situations where a model is used to predict truly unseen data, for example events that haven't happened yet or experiments that haven't been conducted yet. As the saying goes, "It's tough to make predictions, especially about the future"[5]

[4] there is a distinction between validation and test data but it is not relevant for the current discussion

[5] variants of this saying have been attributed to to Neils Bohr, Yogi Berra and Mark Twain

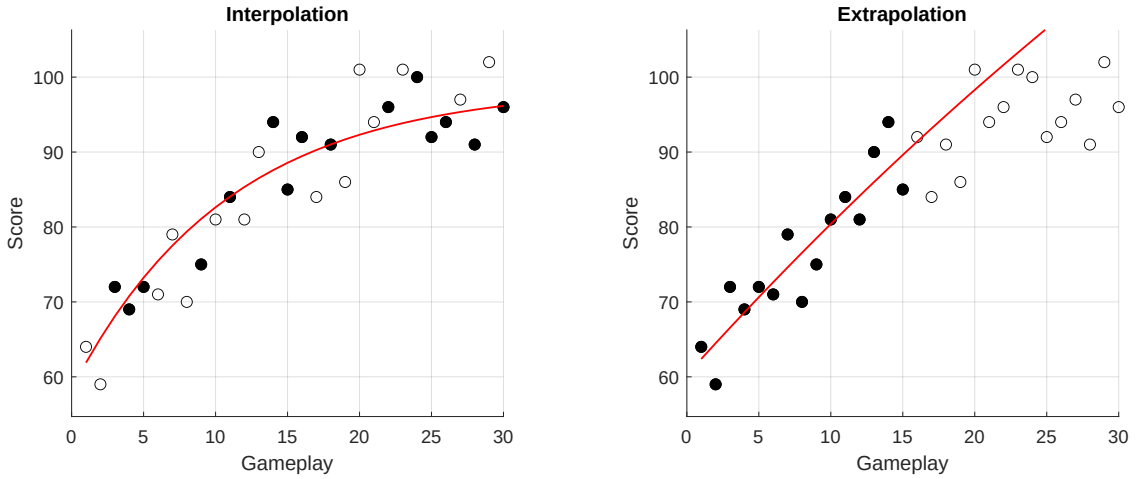

Figure 3.5: Two examples of generalization tests: interpolation and extrapolation. Filled circles show the training data used to fit an exponential learning model. Open circles represent the validation data — the data used to test the ability of the model to generalize to unseen data. The red line shows predictions from the exponential model. The data for this example can be found in `exampledata_Intrapolation.csv` and `exampledata_Extrapolation.csv`. Note that the data is exactly the same for the left and right panel. The only difference is the partitioning of the data into training and validation data

3.3.1 Cross-validation

One particular variant of a generalization test is *cross-validation*. This is a form of an interpolation test that is often applied to smaller data sets. The basic idea is that the available data is partitioned not just once into a training and validation set, but multiple times. Specifically, in *K-fold cross-validation*, the data is randomly partitioned into K partitions. In the first step, the model is trained on the data from all partitions except the first one, which is used for validation. In the second step, the model is trained on the data from all partitions except the second one, which is used for validation. This procedure is repeated until all partitions ("folds") have been used for validation. The evaluation metric is then the model fit on the validation data across all K folds. A typical value for K encountered in the literature is 10 leading to 10-fold cross-validation. Although cross-validation is efficient with data as all data points are used for validation, it is expensive as the model needs to be refit K times.

3.3.2 Exercises

Exercise 3.14 Create code to replicate the results from Figure 3.5. The data for this figure can be found in `exampledata_Intrapolation.csv` and `exampledata_Extrapolation.csv`. The column `istrain` indicates whether the datapoint should be used for model fitting. Report the best fitting parameter values for the exponential model in both cases. ■

Exercise 3.15 Use the code from the previous exercise to evaluate the MSE for the validation data. What MSE values do you obtain for the interpolation and extrapolation data sets? What explains the difference? ■

Exercise 3.16 ** Create code to apply 10-fold cross-validation to the data from Figure 3.5. What is the MSE averaged across the 10 folds (applied only to the validation data)? ■

Further Reading

Anderson, J. R., Fincham, J. M. and Douglass, S. (1999). Practice and retention: A unifying analysis., *Journal of Experimental Psychology: Learning, Memory, and Cognition* **25**(5): 1120.

Anderson, J. R. and Lebiere, C. J. (2014). *The atomic components of thought*, Psychology Press.

Busemeyer, J. R. and Wang, Y.-M. (2000). Model comparisons and model selections based on generalization criterion methodology, *Journal of Mathematical Psychology* **44**(1): 171–189.

Eriksen, B. A. and Eriksen, C. W. (1974). Effects of noise letters upon the identification of a target letter in a nonsearch task, *Perception & Psychophysics* **16**(1): 143–149.

Evans, N. J., Brown, S. D., Mewhort, D. J. and Heathcote, A. (2018). Refining the law of practice., *Psychological review* **125**(4): 592.

Heathcote, A., Brown, S. and Mewhort, D. (2000). The power law repealed: The case for an exponential law of practice, *Psychonomic bulletin & review* **7**(2): 185–207.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A. et al. (2022). An empirical analysis of compute-optimal large language model training, *Advances in Neural Information Processing Systems* **35**: 30016–30030.

Kumar, A., Benjamin, A. S., Heathcote, A. and Steyvers, M. (2022). Comparing models of learning and relearning in large-scale cognitive training data sets, *npj Science of Learning* **7**(1): 24.

Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence, *Artificial intelligence* **33**(1): 1–64.

Newell, A. and Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice, *Cognitive skills and their acquisition* **1**(1981): 1–55.

Steyvers, M. and Benjamin, A. S. (2019). The joint contribution of participation and performance to learning functions: Exploring the effects of age in large-scale data sets, *Behavior research methods* **51**(4): 1531–1543.

Steyvers, M., Hawkins, G. E., Karayanidis, F. and Brown, S. D. (2019). A large-scale analysis of task switching practice effects across the lifespan, *Proceedings of the National Academy of Sciences* **116**(36): 17735–17740.

Steyvers, M. and Schafer, R. J. (2020). Inferring latent learning factors in large-scale cognitive training data, *Nature Human Behaviour* **4**(11): 1145–1155.

4. Modeling Active Learning

Much of the early computational modeling in psychology was based on information-processing theories where the focus was on understanding the flow of information through various stages from the initial stages processing the stimulus input to later stages where a response is formed. This information processing framework limited theories to explaining how people *passively* respond to input. However, this leaves out an important aspect of behavior: How do we seek out information to learn about the environment? In other words, how do we *actively* engage with our environment? In this chapter, we will focus on the computational problem of asking questions. This problem is very challenging when considering open-ended questions (the type we envision that conversational agents such as Alexa or Siri should be able to answer). However, we will consider highly simplified situations based on guessing games where the goal is to guess some concept (e.g. a number or a word) and the types of questions and possible responses are restricted. We will consider the computational strategies that can be used to formulate questions in order to gain as much information about the possible answer.

4.1 Number guessing game

Let's start with a simple guessing game to illustrate the computational problem. Suppose your friend randomly picks an integer number x between 1 and N. You can find out what the number is by asking your friend a series of questions where each individual question is in the form of a single integer number. Your friend will inform you if the number they have in mind matches your number or whether their number is smaller or larger. Therefore, the answers are restricted to three responses: *match*, *smaller*, and *larger*. For example, suppose we set N=16 and your friend picks $x = 4$ as the number to be found. If your first question is 12, your friend will say *smaller*. If you then guess 3, your friend will respond *larger*. The game ends when you guess the correct number, i.e., your friend responds *match*.

What is a good way to ask questions in this game? Specifically, what is the best first question to ask? This problem can be solved efficiently through **binary search**, a well known search algorithm in computer science. In this algorithm, the strategy is to ask a question corresponding to the middle point of the numbers that are still feasible. After each answer, the set of feasible answers is reduced

and the best next question corresponds to the middle point of that new set. This process of halving the intervals continues until the answer is a match. Figure 4.1 illustrates this process when $n = 12$.

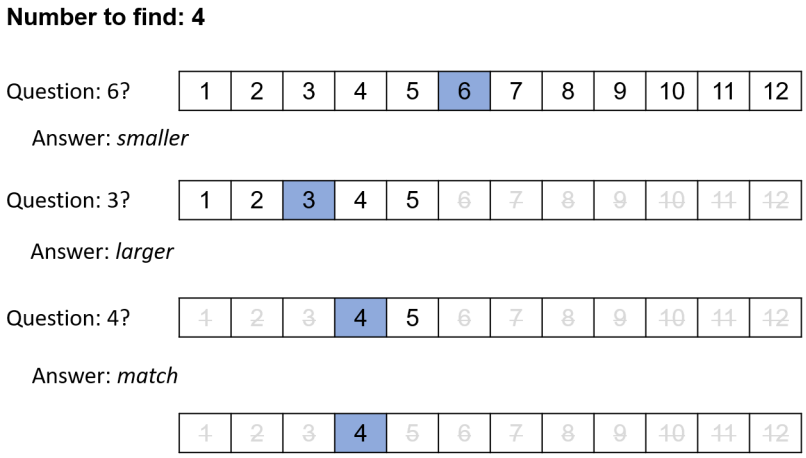

Figure 4.1: Illustration of binary search when $N = 12$. The answer (4) is found after three questions. The greyed out numbers are ruled out based on answers provided.

Why is this search algorithm so efficient? In the best case scenario, the answer is found after just a single question (that just happens to fall on the exact number x). In the worst case, you need to cut the n possibilities in half, and keep cutting the remaining possibilities in half for $log_2 N$ total times. Therefore, the number of questions asked scales, on average, with the logarithm of N[1]. Figure 4.2 shows the average number of questions as a function of N. Note that the x-axis is on a logarithmic scale, and the linear trend in the graph therefore confirms the logarithmic relationship. Note the efficiency of the binary search strategy — when $N = 1000$, it takes on average only 9 questions to find the answer. Compare this binary search with a different strategy where we just ask if each individual guess matches the number or not. This would lead to a **linear search** where the answer is found on average after $N/2$ questions. For example, when $N = 1000$, this would take on average 500 questions.

4.1.1 Exercises

Exercise 4.1 Implement the binary search algorithm and simulate this algorithm for different N and different x. Graph the average number of questions as a function of N and confirm that your results approximate a logarithmic function. To keep track of the remaining candidates, you can maintain two variables representing the lower and upper bound. Initially, the lower bound is 1 and the upper bound is N. After each answer, either the lower or upper bound is changed. ■

4.2 Word guessing game

Let's consider another popular game, the game of 20 questions. Your friend thinks of some object (e.g. from some predefined set of categories such as animals, vegetables, or minerals) and you can ask any question as long as the answer can be formulated as a "yes" or a "no". The goal is to identify the object with at most 20 yes-no questions.

A good strategy for this game is based on binary search. You pose a question that will split the remaining alternatives in half. If you are able to think of such splitting-half questions at each

[1] More formally, one can state that the time complexity is $\mathcal{O}(\log N)$.

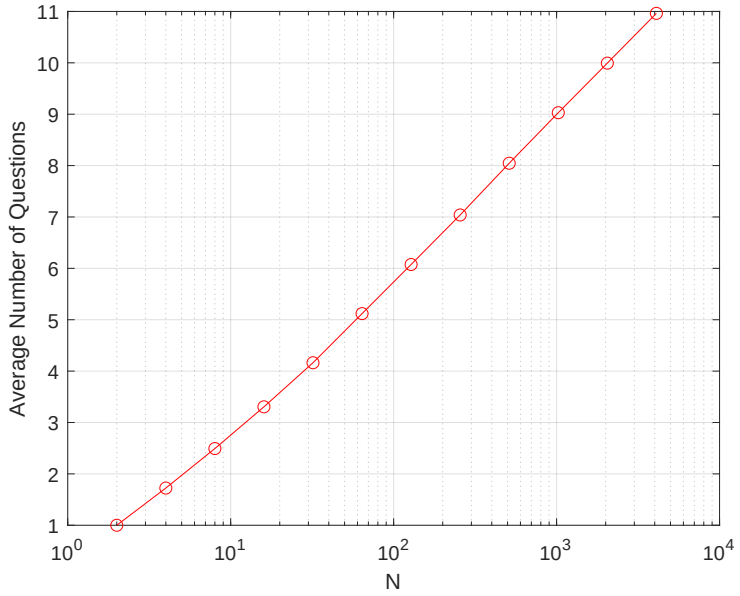

Figure 4.2: Average number of questions asked in guessing game as a function of N

stage, after 20 questions you should be able to distinguish between $2^{20} = 1,048,576$ objects, which should be sufficient to cover most of the words that people typically think of. As you might know, a popular first question is to ask if the object is bigger than a breadbox. The idea is that this question gets closest to cutting the possible answers in half (whether it actually does is another issue). If you have ever played the game of 20 questions online where the computer program attempts to guess your concept, it can be quite perplexing if the program is successful with just a few questions. Keep in mind that this result is based on the rapid elimination through binary search.

To make the game more concrete, let's consider a game with 25 animals and 8 possible questions as illustrated in Figure 4.3[2]. For each question and animal, the entry in the matrix is 1 (yes) or 0 (no) if the animal has the property. Therefore, in this representation we are assuming that the answer will be unambiguous: all players will agree on the answers to all questions. The column on the right shows the proportion of animals that produce a yes answer for each question. Note that the questions might not exactly split the possible answers in half, but some questions get closer to that goal than others.

	clam	slug	gnat	scorpion	kiwi	lark	housefly	octopus	tortoise	ostrich	flamingo	platypus	toad	seasnake	haddock	slowworm	cavy	hare	fruitbat	dogfish	dolphin	gorilla	seal	antelope	sealion	Proportion
is toothed?	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0.52
is cat sized?	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0.44
has hair?	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	1	1	1	0.36
has tail?	0	0	0	1	1	1	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	1	1	0.64
is airborne?	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0.20
has feathers?	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16
breathes?	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1	1	1	0.80
has fins?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	1	0.20

Figure 4.3: Feature matrix for word guessing game

[2]This feature matrix is based on the zoo data set from the UCI Machine Learning Repository

4.2.1 Exercises

Note that the matrix data and the list of concepts can be found in files `zoodata_small_matrix` and `zoodata_small_concepts`.

Exercise 4.2 In the simplified word guessing game in Figure 4.3, what is the best first question to ask?

- [a] is toothed?
- [b] is cat sized?
- [c] has hair?
- [d] has tail?
- [e] is airborne?
- [f] has feathers?
- [g] breathes?
- [h] has fins?

■

Exercise 4.3 In the simplified word guessing game in Figure 4.3, what is the worst question to ask first?

- [a] is toothed?
- [b] is cat sized?
- [c] has hair?
- [d] has tail?
- [e] is airborne?
- [f] has feathers?
- [g] breathes?
- [h] has fins?

■

Exercise 4.4 Suppose the first question asked is whether the animal is toothed and the answer is no. What is one of the best next questions to ask?

- [a] is toothed?
- [b] is cat sized?
- [c] has hair?
- [d] has tail?
- [e] is airborne?
- [f] has feathers?
- [g] breathes?
- [h] has fins?

■

Exercise 4.5 Create a program that implements the binary search strategy for the simplified word guessing game using the data from Figure 4.3. What is the minimum and maximum number of questions that need to be asked to guess any of the animals?

a Minimum = 1; Maximum = 2;
b Minimum = 2; Maximum = 3;
c Minimum = 3; Maximum = 4;
d Minimum = 4; Maximum = 5;
e Minimum = 5; Maximum = 6;

■

Exercise 4.6 Modify your program to implement a linear search strategy by asking questions in a random order (a different order every time the game is played). Note that this strategy is not *adaptive* as the next question does not depend on the answers received. What is the average number of questions needed to arrive at the answer?

a Between 3-4
b Between 4-5
c Between 5-6
d Between 6-7

■

4.3 More complex games

Up to this point, we have considered fairly simple situations. For example, the numbers or concepts are chosen uniformly from a set of alternatives. However, suppose that some concepts are more likely to be chosen a priori than others. For example, suppose you know that your friend is more likely to think of a mammal as opposed to a non-mammal. What is now a good strategy to ask questions given that prior knowledge? Similarly, what if there is some uncertainty about the answers your friend gives? Perhaps your friend might not know everything about animals or understand the feature that is being asked about. Also, there might be ambiguity about the features. Suppose your friend is thinking about a snake, and when asked if the object is venomous, your friend will answer no, because your friend is answering about the particular non-venomous snake that she has in mind. How do we take the possibility of ambiguity or errors about the features into account?

To develop strategies for these more complex problems, we will need to go beyond binary search, and use concepts from **information theory**. With an information theoretic framework, we can formalize what it means to ask a good question: *a good question is one that is expected to gain the most information about the possible answer*. Conveniently, there is a quantity called *expected information gain* which we can calculate for each possible question, and the best question to ask is the one with the highest expected information gain.

To develop this concept, let's first introduce the concept of **entropy**, a measure of uncertainty. Suppose at the start of the game the alternatives have equal probability of being chosen and each option has probability $1/N$. This is a state of maximum uncertainty. At the end of the game, if we have identified the correct answer, the probability of one of the correct answers is 1 and all other answers have probability 0. This is a state of minimal uncertainty.

More formally, entropy is defined as[3]:

$$E(X) = -\sum_{i}^{N} P(x_i) \log_2 P(x_i) \tag{4.1}$$

Let's illustrate this concept with some numerical examples. Suppose the game has six alternatives ($N = 6$). At the start of the game, if the alternatives have equal probability, $P(x_i) = [1/6\ \ 1/6\ \ 1/6\ \ 1/6\ \ 1/6\ \ 1/6]$, we have $E(X) = -6((1/6)\log_2(1/6)) = 2.59$. At the end of the game, one alternative has probability 1, and the others have probability zero, e.g., $P(x_i) = [1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$, and we have $E(X) = -(1\log_2 1) = 0$, corresponding to a complete lack of uncertainty. Intuitively, the game progresses from a state of high uncertainty to a state on minimal uncertainty.

Next, we consider changes in the uncertainty as a result of getting answers. **Information gain** is defined by the change in uncertainty as a results of getting an answer a to question q:

$$IG(X|q,a) = E(X) - E(X|q,a) \tag{4.2}$$

Note that the first term on the right side, $E(X)$, is the uncertainty we started with before receiving new information. The second term, $E(X|q,a)$, is the new uncertainty after we observe answer a for question q. Again, suppose we have six alternatives and we start the game with each alternative having equal probability. We have already derived that this leads to $E(X) = 2.59$. Suppose we now ask a question and the answer we receive eliminates the second half of the alternatives. Therefore, the probabilities now look like $P(x_i) = [1/3\ \ 1/3\ \ 1/3\ \ 0\ \ 0\ \ 0]$. The new entropy is $E(X|q,a) = 1.59$. Therefore, the information gain is $2.59 - 1.59 = 1$. We have reduced our uncertainty after learning the answer and therefore gained some information about the possible alternatives.

The final concept we need to introduce is **expected information gain**. If you know what answer will be given to a particular question, you can compute the standard information gain. However, at the time of asking a question, *you don't know yet what answer might be given*. Therefore, you need to consider all possible answers that might be given and calculate a weighted average of the entropy values where the weights correspond to the probability of receiving each answer. Formally, we have:

$$IG(X|q) = E(X) - \big(P(a = yes|X,q)E(X|q,a = yes) + P(a = no|X,q)E(X|q,a = no)\big) \tag{4.3}$$

Note that one key difference is that Eq. 4.2 assumes that we already know the answer a whereas Eq. 4.3 assumes we do not have such knowledge, and therefore the term a is absent from the term $IG(X|q)$. In this notation, $E(X|q,a = yes)$ and $E(X|q,a = no)$ represent the new uncertainty after learning that the answer to question q is yes and no respectively. The terms $P(a = yes|x,q)$ and $P(a = no|x,q)$ are the probabilities that a yes or no answer will be given to question q conditional on our knowledge of X (before learning about the answers). Note that these probabilities serve as weights in a weighted average of the new uncertainty.

To calculate $P(a = yes|X,q)$, we compute an weighted average of the binary feature values for each object i weighted by the probability $P(x_i)$. Suppose the feature matrix is represented

[3]The calculation of entropy can lead to numerical problems when one of the probabilities equals 0. Formally, the product $0\log 0$ is defined as 0 but programming languages might return a *not a number value* for $0\log 0$. To circumvent this problem, the sum should ignore these *not a number* values. In Matlab for example, if p is a vector of probabilities, entropy can be calculated without numerical problems by the expression `E = -nansum( p .* log2( p ));` Note also that the log is taken to the base 2, such that entropy is expressed in *bits*. However, for our purposes, this measurement in bits is not strictly necessary and the logarithm to any base will work fine.

by F such that $F_{k,i} = 1$ if object i has feature k, and $F_{k,i} = 0$ otherwise. We can then calculate $P(a = yes|X, q = k) = \sum_{i=1}^{N} F_{k,i} P(x_i)$. Similarly, we have $P(a = no|X, q = k) = \sum_{i=1}^{N} (1 - F_{k,i}) P(x_i)$. Note that in this notation, $q = k$ means that we are asking a question about feature k.

Finally, to put all of this together, what is the best question to ask? It is the question q that maximizes the expected information gain:

$$\arg\max_{k} IG(X|q = k) \tag{4.4}$$

It should be noted that this process can be repeated over several steps in order to consider the most informative question after just receiving the answer to the previous question. After receiving each answer, the probabilities $P(x_i)$ should be updated to reflect the new information received: all concepts that have been ruled out should be set to zero probability, and the remaining non-zero values should be normalized such that they sum to one[4]

	slug	scorpion	housefly	hare	gorilla	antelope
is toothed?	0	0	0	1	1	1
has hair?	0	0	1	1	1	1

Figure 4.4: Toy example of a feature matrix F with two features and six objects

4.3.1 Examples

Let's look at some worked out examples. Consider the toy example in Figure 4.4 with six animals and two features. What is the best first feature to ask about when the alternatives are all equally likely from the start? This scenario is explained in Figure 4.5. The two panels consider the consequences of asking about each feature. The graphs shows the probabilities over the six animals before asking the question (top) and after asking question and receiving an answer (bottom). The expected information gain is largest for questions related to the first feature (row 1 of 4.4). This result is consistent with the results from the binary search algorithm. Note that the question about the second feature can potentially lower the uncertainty the most ($E = 1$) when the answer is no. However, as that answer is not likely (only probability 1/3), it does not play a large role in the weighted average of entropy.

Now consider the scenario in 4.6 where the alternatives do not have equal probability a priori. The animals have prior probabilities $P(x_i) = [1/4 \;\; 1/4 \;\; 1/8 \;\; 1/8 \;\; 1/8 \;\; 1/8]$. This might happen for a variety of reasons. For example, some concepts might occur with very low frequency in our environment and they are less likely to be used as target concepts. Or you might have insights about your friend who is thinking of target concepts. In this scenario, the most informative question (associated with the highest information gain) is about feature 2 – a result opposite from the first scenario. The key difference is that the prior probabilities now favor the first two animals, and a question that can confirm or disconfirm whether the animals belong to this set of two is more informative than a question that splits the alternatives into two equal sets. This highlights an important finding. Questions that center on a small set of concepts *can* be informative as long as

[4]This step of updating $P(x_i)$ is a process of Bayesian updating where the new evidence, the answers received, are used to update the prior probabilities. We will revisit Bayesian inference in a different chapter

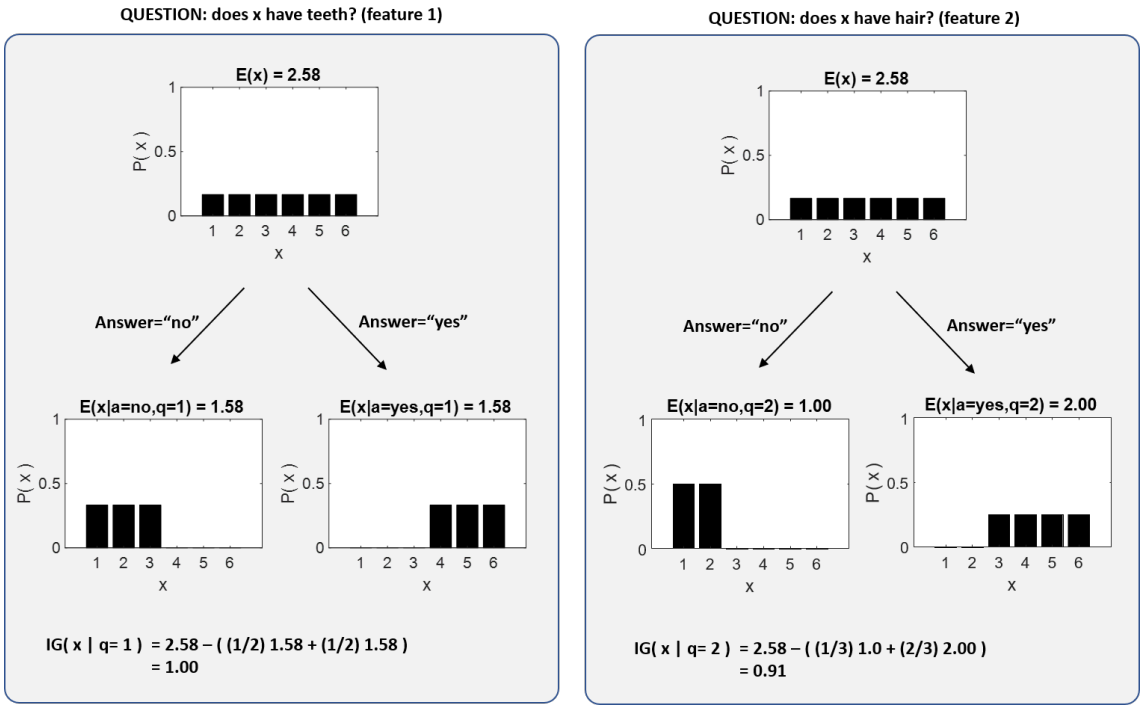

Figure 4.5: Illustration of the computation of expected information gain when asking about two different features (panels) and each alternative is equally likely a priori. Note that the first question is the best question to ask — it has the highest expected information gain.

the prior evidence for that set is sufficiently strong. In fact, a question that confirms just a single concept (i.e., asking about a property belonging to a single animal), could be the most informative if there is sufficient a priori evidence to support that concept.

4.3.2 Exercises

Exercise 4.7 What is the entropy $E(X)$ for the probability distribution $P(x_i) = [2/5 \;\; 1/5 \;\; 1/5 \;\; 1/5 \;\; 0 \;\; 0]$ ■

Exercise 4.8 Your friend throws fair six-sided die without you seeing the resulting outcome. Your friend informs you that the result is an even number. What is the information gain IG in this situation? ■

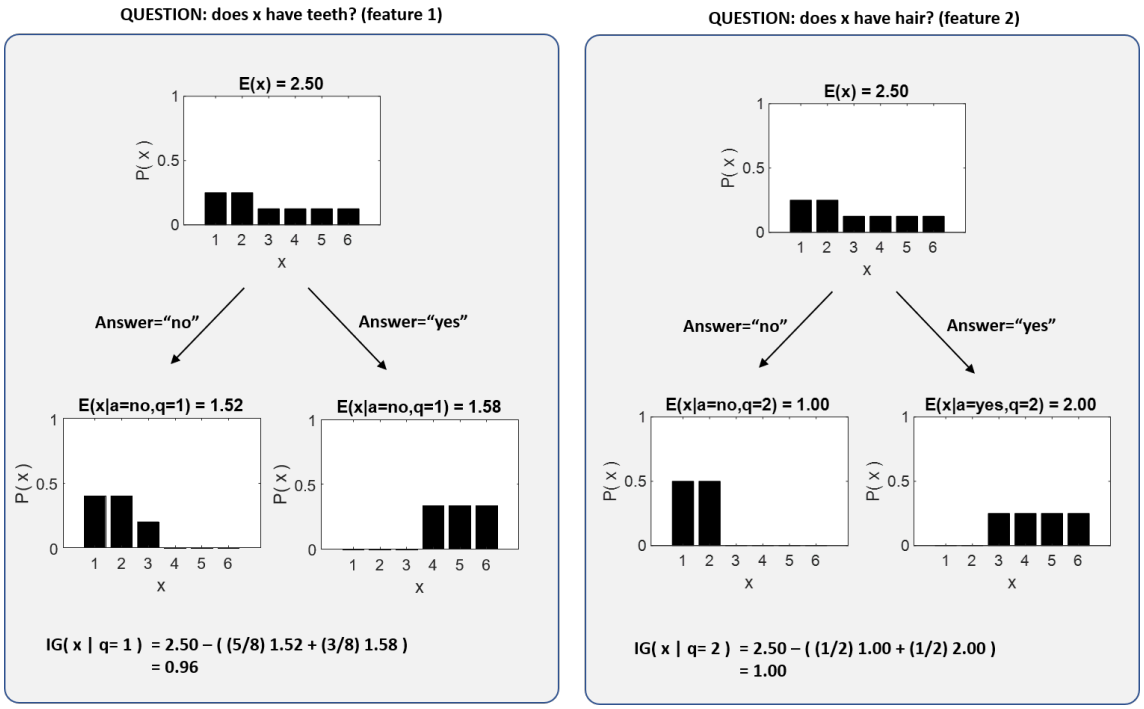

Figure 4.6: Illustration of the computation of expected information gain when asking about two different features (panels). A priori, the first two alternatives have probability 1/4 each and the remaining four alternatives have probability 1/8 each. Note that the second question is the best question to ask even though it does not split the alternatives in half.

Exercise 4.9 Develop code for the 20 questions game for the data in Figure 4.3 and use expected information gain to choose questions. First, confirm that the code performs the same as binary search when the concepts are all equally likely a priori. Next, use the prior as shown in Figure 4.7, which can be found in file `zoodata_small_priorprobs`. With this prior, mammals are given higher probability than non-mammals. What is the best first question to ask with this prior?

a. is toothed?
b. is cat sized?
c. has hair?
d. has tail?
e. is airborne?
f. has feathers?
g. breathes?
h. has fins?

■

Further Reading

Coenen, A., Nelson, J. D. and Gureckis, T. M. (2019). Asking the right questions about the psychology of human inquiry: Nine open challenges, *Psychonomic Bulletin & Review* **26**(5): 1548–1587.

Cohen, A. and Lake, B. M. (2016). Searching large hypothesis spaces by asking questions., *CogSci*.

	clam	slug	gnat	scorpion	kiwi	lark	housefly	octopus	tortoise	ostrich	flamingo	platypus	toad	seasnake	haddock	slowworm	cavy	hare	fruitbat	dogfish	dolphin	gorilla	seal	antelope	sealion
Class Type	Invertebrate	Invertebrate	Bug	Invertebrate	Bird	Bird	Bug	Invertebrate	Reptile	Bird	Bird	Mammal	Amphibian	Reptile	Fish	Reptile	Mammal	Mammal	Mammal	Fish	Mammal	Mammal	Mammal	Mammal	Mammal
Prior probability	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0769	0.0192	0.0192	0.0192	0.0192	0.0769	0.0769	0.0769	0.0192	0.0769	0.0769	0.0769	0.0769	0.0769
is toothed?	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
is cat sized?	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1
has hair?	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	1	1	1
has tail?	0	0	0	1	1	1	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	1	1
is airborne?	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
has feathers?	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
breathes?	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1	1	1
has fins?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	1

Figure 4.7: Feature matrix for word guessing game with higher prior probabilities for Mammals

Hawkins, R. and Goodman, N. (2017). Why do you ask? the informational dynamics of questions and answers.

Markant, D. B. and Gureckis, T. M. (2014). Is it better to select or to receive? learning via active and passive hypothesis testing., *Journal of Experimental Psychology: General* **143**(1): 94.

Rothe, A., Lake, B. M. and Gureckis, T. M. (2016). Asking and evaluating natural language questions., *CogSci*.

Rothe, A., Lake, B. M. and Gureckis, T. M. (2018). Do people ask good questions?, *Computational Brain & Behavior* **1**(1): 69–89.

5. Modeling Theory of Mind: Assistance Games

People readily help other people in a variety of circumstances. Developmental studies have shown that infants from a very young age display helping behaviors (Warneken and Tomasello; 2006)[1]. From a cognitive point of view, helping is an interesting behavior. It requires that the helping person knows about the goal of the other person and the actions that can be taken to accomplish the goal. For example, if we observe a person leaving money on a table in a restaurant, what should we do? Should we run after the person and return the money because we think the person unintentionally left the money on the table? Or, depending on the country where this place, did the person mean to leave a tip, and returning the money would be decidedly unhelpful? To be able to understand the goal of another person requires a capacity to take the perspective from the other person and reason from the observed actions to unobservable goals, a capacity known as *theory of mind*. Being an effective helper requires this capacity of theory of mind – an ability to think about what another person is thinking.

In this chapter, we will examine helping as a computational problem and propose a probabilistic perspective based on pragmatic reasoning (Goodman and Frank; 2016) to explain some very basic human behaviors. Developing such a computational model of helping furthers not only basic research in social cognition and cognitive science in general, but can also play an important role in developing better AI systems Puig et al. (2020). Humans are increasingly being supported by AI to make decisions in a variety of real-world applications. Some AI researchers have argued that AI systems will need to be designed from the ground up with the objective to be helpful to humans (Russell; 2019). Having computational models of human helpful behavior will be useful in this AI development.

5.1 Assistance Games

To keep things simple, we will study the computational problem of helping through gameplay in an *assistance game*. In an assistance game, there are multiple players who have to collaborate to reach a common objective. The game is asymmetric in the sense that some players have more information

[1] some of the classic studies on helping behaviors by young infants can be seen in this video

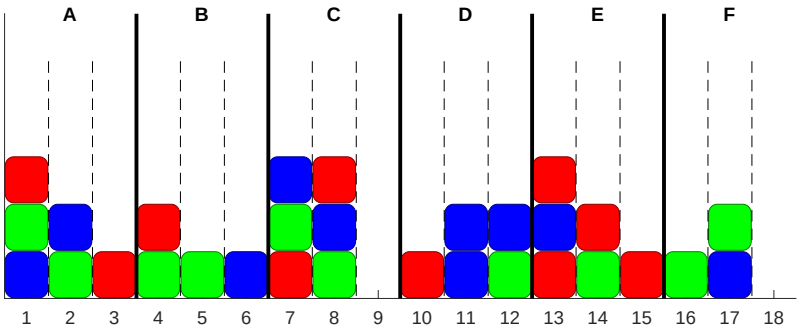

Figure 5.1: Illustration of the two-player collaborative helper game. In this game, there are three block colors, red, green, and blue and six different areas for the blocks, labeled A-F. Two players take turns moving one block at a time from one column (1-18) to another column (1-18). One player is assigned the architect role. This player is given a goal such as "move all red blocks to area B". The other player is assigned the helper role. This player does not know the goal but needs to infer from the architect's moves what the goal is and then make moves to be helpful. The objective of the game is to solve each problem in the lowest total number of total moves (i.e., moves across both players).

than others and therefore, effective collaboration requires that each player tries to fill in their own knowledge gaps by observing the actions of other players. In addition, the players who have more information should take actions to best signal their knowledge to other players.

Figure 5.1 shows an example of a simple two-person assistance game that we will use throughout this chapter. This Figure shows a particular configuration of blocks at the start of the game. Players move blocks according to the following two rules: 1) Each player can move one block at a time, and 2) a block has to be picked from the top of a stack and placed on the top of the stack in *another* column. Therefore, with these rules, a move can only involve blocks that are unobstructed by other blocks (i.e., the blue block in column 17 cannot be moved yet).

Each player is assigned a distinctive role. The *architect* is a player who is told what the goal is that needs to be accomplished. For example, suppose the architect is told that the goal is to move all red blocks to area B. The other player is assigned the role of *helper*. This player does not know what the goal is but needs to be as helpful as possible and assist the architect. Initially, the helper does not know anything about the goal but after observing the architect's moves, the goal (hopefully) becomes more clear. Players take turns with the architect taking the first turn. The objective of the game is for the players to accomplish the goal (which varies from game to game) in as few moves as possible. Note that in this setup, there is a clear reason for helping because doing so will lead to the goal being accomplished in the shortest time possible.

Let's make a few additional assumptions about the game that will help to constrain the computational problem. While the helper does not know the specific goal, let's assume that the helper does know the set of possible goals. Throughout this chapter, we will assume that the goals are of the form "move all blocks of color X to area Y". In addition, let's assume that each area (A-F) has sufficient capacity to fit all the blocks so there is no need to "free" an area of some blocks to make room for other blocks. Finally, we will assume that there is no possibility to use language to exchange information between players. The only way to communicate is through moving blocks.

Before we proceed to the computational model, let's go through some exercises to make sure that you understand the basic game setup.

5.1.1 Exercises

Exercise 5.1 In the example game shown in Figure 5.1, how many possible moves are there for the first player?

a 18
b 35
c 272
d 288
e 324

■

Exercise 5.2 In the example game, there are three block colors, red, green, and blue and six different areas for the blocks. If the goals are of the form "Move all blocks of color X to area Y", how many different goals are there?

a 3
b 6
c 9
d 12
e 18

■

Exercise 5.3 Suppose we start a game as illustrated in Figure 5.1. If the goal is to move all red blocks to area B, what is the minimum number moves required to accomplish this goal?

a 10
b 11
c 12
d 13
e > 13

■

5.2 Computational Model

We will consider two models of the architect, the basic architect and the pragmatic architect and one model for the helper, the basic helper. The difference between the basic and pragmatic models is that only the latter player considers how the actions it takes influences the other player.

5.2.1 Calculating Minimum Number of Moves

Before we describe the player models, let's first consider the most basic calculation for this game: how many moves does it require to accomplish a specific goal given a particular configuration of blocks? Let g represent the goal and c the current configuration of the game. This configuration represents the position and color of each block such that one can create a figure like 5.1. Let MINIMUMMOVES(g,c) represent the function that returns the minimum number of moves needed to accomplish goal g starting from configuration c. Initially, it might seem complicated to calculate this function and require some type of path search algorithm (such as A*). However, given our assumption that the goals are of the form "move all blocks of color X to area Y", this can be done using a simple counting algorithm as shown in Listing 5.1. In this algorithm, one simply counts the number of blocks of the target color that need to be moved to the target destination in addition to

any blocks that are obstructing the target blocks.

Algorithm 5.1 Pseudocode for the procedure that returns the minimum number of moves to accomplish goal g given configuration c

INPUT: g (goal), c (current configuration)
OUTPUT: n (number of moves)

```
1:  function MINIMUMMOVES(g,c)
2:      n ← 0                                   ▷ Initialize number of moves
3:      for i=1,...,NumColumns do               ▷ Loop over all columns
4:          if column i is not a target for goal g then
5:              if column i contains blocks of target color then
6:                  x ← Position of lowest target block in column i
7:                  h ← Total number of blocks in column i
8:                  n ← n + (h − x + 1)         ▷ Number of moves needed
                                                    for this column
9:              end if
10:         end if
11:     end for
12:     return n                                ▷ Return the number of moves
13: end function
```

5.2.2 Basic Architect

Let's first consider the basic model for the architect player. In this model, we assume that the basic architect evaluates the utility of each move by calculating the difference between the current number of moves needed to accomplish the goal g and the number of moves needed after a move, m, is made:

$$u(m|g,c) = \text{MINIMUMMOVES}(g,c) - \text{MINIMUMMOVES}(g,s(m,c)) \tag{5.1}$$

In this notation, $s(m,c)$ is a state-transition function that take the current configuration c and returns the new configuration of blocks after move m is made; $u(m|g)$ represents the utility of move m given goal g. Given the set of goals we are considering, this utility can take on values of +1, 0, and -1. A positive utility value of +1 is achieved when the move is a good one, and takes the solution one step closer to the goal. A negative utility of -1 is achieved when the move is counterproductive and (unnecessarily) leads to an extra step to solve. For example, when the goal is to move all red blocks to B, moving the blue block from column 2 to column 1 has utility -1 because the blue block obstructs the red block and needs to be removed again. Finally, a utility of 0 is achieved when the move is neither helpful nor counterproductive (e.g., the green block in column 17 is moved to column 18).

To model decisions by the architect, one option is to assume that the architect picks one of the moves that maximizes the utility. While this is certainly a reasonable model for a player who is familiar with the game, it does not allow for mistakes. To turn this into a *cognitive* model, as opposed to a model for the optimal player, we will have to assume that mistakes can be made occasionally. We will use a softmax rule (see Box 5.2.1), closely related to the Luce decision rule, in which a possible move m_i is selected with probability:

$$p(m_i|g,c) = \frac{e^{u(m_i|g,c)/\tau}}{\sum_{m_j \in M(c)} e^{u(m_j|g,c)/\tau}} \tag{5.2}$$

In this model, τ is known as the "temperature" parameter (owing to its origin in statistical mechanics). For our purposes, parameter τ reflects the amount of decision noise. If the decision noise is low, it is likely that an optimal move (a move with the highest utility) is chosen. In the extreme, when $\tau \to 0$, only moves with the highest utility will be chosen (if there are several moves that are all equally good, one is chosen at random). When the decision noise is increased, it becomes more and more likely that a suboptimal move is chosen, capturing the possibility of player mistakes.

Note that sum in the denominator is over all possible moves. We use $M(c)$ to refer to the set of possible moves that can be made from configuration c. Given the rules of the game, for example, moves cannot take blocks from an empty column and therefore, what moves are possible and not possible depends on the particular configuration of blocks. The sum in the denominator ensures that the equation results in a probability that appropriately sums to one.

5.2.3 Exercises

Exercise 5.4 Suppose we have a vector of utilities, [+1 0 +1 -1] corresponding to four possible moves. We use the softmax rule with a temperature parameter $\tau = 1/2$ to calculate the probability of making each move. What is the probability that the first move (corresponding to the first utility in the vector) will be selected? ■

Exercise 5.5 If we increase the parameter τ to reflect a large amount of decision noise, what will happen to the probability of making the first move?

a Increases
b Decreases
c Stays the same

■

Exercise 5.6 If we take the configuration in Figure 5.1 and the goal is to move all red blocks to area B, what is a more likely move that the Basic Architect will make: move X = moving the blue block from 7 to 9 or move Y = moving the red block from 8 to 4?

a move X is more likely than move Y
b move Y is more likely than move X
c move X is as likely as move Y

■

5.2.4 Basic Helper

Now we consider the basic model of the helping player. There are two problems that need to be addressed by the helper. First, the helper needs to infer the goal the architect player is trying to accomplish. This inference process will involve some uncertainty as it might not be entirely clear what the goal is after observing a few moves. Second, the helper will need to make a move that is most likely to be helpful even when the helper is still somewhat uncertain about the true goal.

To get started with the first problem, we will use *Bayes rule* as a way to model the inference process that the player might use. Let's assume that the architect has made a single move m. At this point, the helper needs to infer $p(g_j|m)$, the conditional probability of a particular goal g_j given move m. In other words, this conditional probability expresses how likely it is that goal g_j is the correct goal given the information available to the helper. Using Bayes rule, this conditional

probability can be evaluated as follows:

$$p(g_j|m,c) = \frac{p(m|g_j,c)p(g_j)}{\sum_k p(m|g_k,c)p(g_k)} \tag{5.4}$$

There are two kinds of terms on the right hand side: the conditional probability $p(m|g_j,c)$, also known as the *likelihood* and $p(g_j)$, also known as the *prior*. The likelihood term expresses how likely the observed information (i.e., the move made by the architect) is under the assumption that goal g_j is the correct goal. How does the helper evaluate this conditional probability $p(m|g_j,c)$? We are assuming that this is evaluated using Equation 5.2. This is a key step in the model. Essentially, in this model, we are assuming that the helper can reason from the architect's perspective and evaluate how likely it is that the architect would make the move it did, if the correct goal is g_j. In some situations, we can use non-uniform probabilities. For example, if the goal was picked by the architect, and the architect has a tendency to select some types of goals (e.g. involving red blocks) more than others, the prior can capture these asymmetries.

The prior term $p(g_j)$ in Equation 5.4 expresses the prior belief by the helper that goal g_j is correct, before any observations are made. To simplify things, we will assume a *uniform prior* where the same probability is assigned, a priori, to each goal, i.e. $p(g_j) = 1/G$, where G is the total number of goals.

The model so far describes how a helper infers the goal if the architect has made only a single move. Suppose the architect has made N moves. Let's assume that h_i represents event i in the sequence consisting of a move m_i made from a particular configuration c_i. We can now extend the model as follows:

$$p(g_j|h_1,\ldots,h_N) = \frac{\left[\prod_{(m_i,c_i)\in h_i} p(m_i|g_j,c_i)\right] p(g_j)}{\sum_k \left[\left[\prod_{(m_i,c_i)\in h_i} p(m_i|g_k,c_i)\right] p(g_k)\right]} \tag{5.5}$$

Now we turn to the next computational problem faced by the helper. How does the helper make an actual move?

Given that there is some uncertainty about goals, each move might or might not be helpful. The helper can take this uncertainty into account and compute the *expected utility* of each move. This is the utility one can expect if we consider the utility of a move given a hypothesized goal weighted by the probability that the hypothesized goal is the correct goal, summed over all possible goals:

$$u(m|h_1,\ldots,h_N) = \sum_j u(m|g_j,c_N)p(g_j|h_1,\ldots,h_N) \tag{5.6}$$

In this Equation, $u(m|g_j,c_N)$ can be evaluated according to Equation 5.1 where c_N represents the current state after the last move N by the architect. The term $p(g_j|h_1,\ldots,h_N)$ is evaluated according to Equation 5.6. Finally, we assume that the helper makes moves according to a softmax rule:

$$p(m_i|h_1,\ldots,h_N) = \frac{e^{u(m_i|h_1,\ldots,h_N)/\tau}}{\sum_{m_j\in M(c)} e^{u(m_j|h_1,\ldots,h_N)/\tau}} \tag{5.7}$$

Therefore, the helper can make some mistakes and execute a move that does not correspond to the highest utility move.

5.2.5 Exercises

Exercise 5.7 Implement the two-player game and create code to simulate moves according to the *basic architect* and the *basic helper*. Assume a starting configuration as shown in Figure 5.1 and assume that the goal is to move red blocks to area B. Finally, assume $\tau = 1/10$. Show the moves taken by each player when they take turns (the architect starts first). Repeat the simulation four times for different random seeds, and show the move sequences. ■

Exercise 5.8 Based on the code you developed in the previous exercise, simulate what the basic helper would do when it makes the first move, and therefore, there is no history of moves from the basic architect. You can simulate this situation in a number of ways, but one is to replace equation 5.4 with the uniform prior. In other words, a priori, before seeing any moves from the architect, the probability of each goal is equally likely. In your simulation, demonstrate some example moves that are made by the basic helper. In addition, comment on the following question: is each possible move equally likely, or, are there some moves preferred over others? If so, which types of moves are preferred? ■

5.2.6 Pragmatic Architect

We now consider an extension of the basic model for the architect. The problem with the basic architect is that does not consider the effects of its actions on the helper. The way the basic architect is designed, it does not matter that this is a two- or single player game. The model does not take into account how its sequence of moves might cue the helper about the correct goal. For example, Figure 5.2 shows the same initial configuration we have considered previously. Again, as before, we consider the goal of moving all red blocks to area B. Two potential moves, X and Y, are illustrated (out of many other potential moves). Both moves have utility +1 as the moves take the solution one step further towards the goal. In move X, a red block is moved to the target area. In the move Y, the blue obstructing block is moved out of the way to clear the path (eventually) for the bottom red block. For the basic architect, it does not matter whether the X or Y move is made first. Both are equally effective. However, from the helper's perspective, move C is somewhat ambiguous. Perhaps the architect is trying to clear the obstruction to move the green block. Alternatively, the architect is trying to clear the obstruction to move the red block.

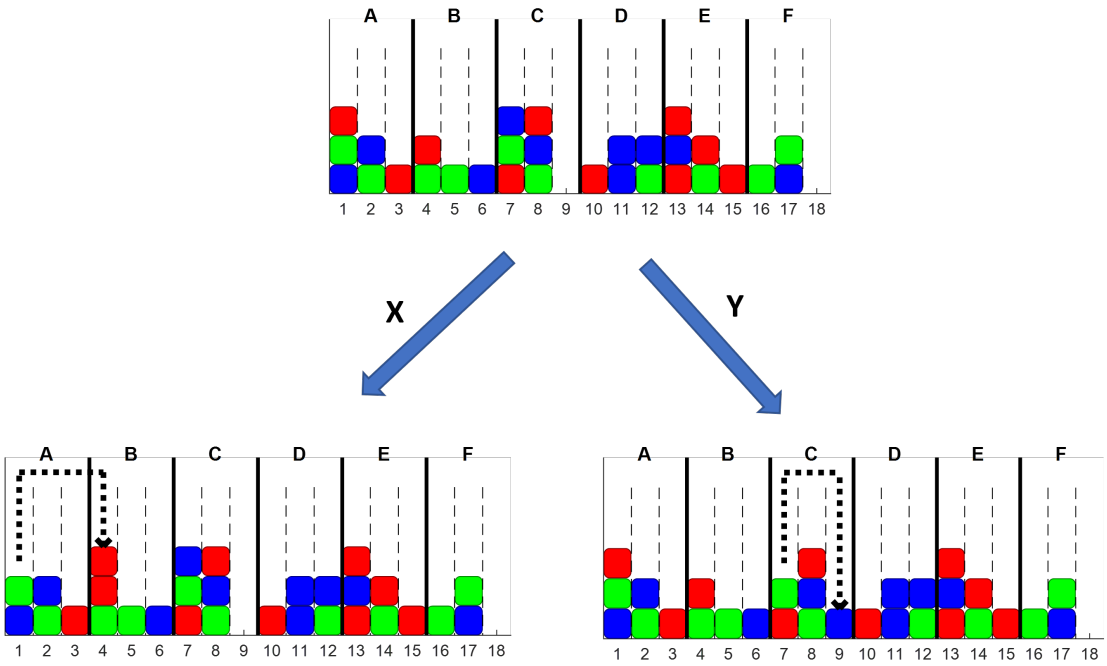

Figure 5.2: Illustration of two equally likely moves, X and Y, that can be played by the basic architect starting with the configuration at the top. The goal is to move all red blocks to area B.

We now consider an extension of the model for the architect that takes into account the helping player and the fact that some moves might be more informative than others. Therefore, this architect

is taking into account the pragmatics of the situation and considers the communicative consequences of its actions on the other player.

To simplify the exposition, let's assume that the architect is playing the first move. Let g_{true} represent the true goal that the architect needs to accomplish. We will assume that the pragmatic architect can reason from the perspective of the basic helper and mentally simulate what inferences the helper will make after the helper observes the move taken by the architect. Therefore, the pragmatic architect can evaluate how likely the helper thinks the true goal is after it has observed a move from the architect. A major assumption we make is that the pragmatic architect evaluates the utility of its move based on the likelihood that the helper recognizes the correct goal:

$$u(m|g_{true},c) = p(g_{true}|m) \tag{5.8}$$

where $p(g_{true}|m)$ is evaluated using Equation 5.4. With a utility computed for each move, we again apply a softmax decision rule to select a move:

$$p(m_k|g_{true},c) = \frac{e^{u(m_k|g_{true},c)/\tau}}{\sum_{m_j \in M(c)} e^{u(m_j|g_{true},c)/\tau}} \tag{5.9}$$

The model can be extended to the subsequent moves from the architect by using Equation 5.6 instead of Equation 5.4 such that the history of moves is taken into account.

With this model of the pragmatic architect, move X in Figure 5.2 is more likely than move Y for the simple reason that move X is more likely to lead to the correct goal recognition by the helper.

5.2.7 Exercises

Exercise 5.9 ** Implement the pragmatic architect with the same setup as in the previous exercise. Calculate what first moves are likely according to this player and demonstrate that moves that try to clear obstructions for red blocks are less likely than moves that move red blocks directly to the target area. ■

5.2.8 Model Extensions

By now, it should be clear that there are different levels of complexity in modeling the players. At each new level of complexity, we can assume that there is one more level of recursion. One player thinks about what another player thinks, who thinks in turn about the other player. In principle, this can lead to an infinite recursion but of course, cognitive limitations will prevent people from reaching deep levels of recursion in these assistance games.

From a computational perspective, it is not too difficult to state the next level of complexity for the helper model. In this model, the helper knows that the architect is pragmatic. Therefore, instead of using the basic architect model, the helper uses the pragmatic architect model to reason about the architect's actions. We can achieve this by evaluating the term $p(m|g_j,c)$ in Equation 5.4 not by Equation 5.2 but by Equation 5.9 instead. In similar ways, the model for the architect can be extended by assuming that the architect uses a more complex model of the helper. These developments are outside the scope of this book.

We intentionally simplified the assistance game to limit the complexity of the computational problem and implementation in computer code. However, it should be clear that there are many ways to extend the game and general helping paradigm. For example, there can be additional types of goals (e.g., move all red blocks on top of some blue blocks, make sure that green does not touch blue, clear out area B, etc). In addition, the game could involve a mixture of different human and AI players in order to investigate how AI agents can be as effective (or even more effective) as human

players. The game could be expanded to allow simple forms of communications short of natural language – perhaps players can ask yes/no questions. Finally, the game mechanics can replace the turn-taking requirement by more complex asynchronous setups where the architect does not have to wait for the helper to make a move and vice versa.

Further Reading

Goodman, N. D. and Frank, M. C. (2016). Pragmatic language interpretation as probabilistic inference, *Trends in cognitive sciences* **20**(11): 818–829.

Puig, X., Shu, T., Li, S., Wang, Z., Liao, Y.-H., Tenenbaum, J. B., Fidler, S. and Torralba, A. (2020). Watch-and-help: A challenge for social perception and human-ai collaboration, *arXiv preprint arXiv:2010.09890* .

Russell, S. (2019). *Human compatible: Artificial intelligence and the problem of control*, Penguin.

Warneken, F. and Tomasello, M. (2006). Altruistic helping in human infants and young chimpanzees, *science* **311**(5765): 1301–1303.

Box 5.2.1: Softmax function to model noise in decision-making

The softmax function is commonly used in cognitive models and machine learning to model the probabilistic process in choosing an option from a set of options. Think of it as a way to convert "utilities" or "value scores" of different choices into probabilities that a person will actually choose each option. Whereas a standard max operation would return the option with the highest utility, the softmax operation makes it possible that an option that is not associated with the maximum utility is chosen.

The equation for the softmax function with a temperature parameter τ is:

$$P(x_i) = \frac{e^{(x_i/\tau)}}{\sum_j e^{(x_j/\tau)}} \tag{5.3}$$

where x_i is the utility or value score for choice i, τ is the temperature parameter, and $P(x_i)$ is the probability of choosing option i. The temperature τ modulates the randomness in choice: when τ is close to zero, the choice becomes deterministic—essentially picking the highest utility option. As τ increases, the choice becomes more random, giving lower utility options a better chance of being chosen.

Computational Procedure for Sampling a Choice
Here are the steps to sample an option:

1. *Compute the Exponentials*: calculate $e^{(x_i/\tau)}$ for each option i.
2. *Sum the Exponentials*: add all these calculated values together.
3. *Calculate Probabilities*: use the softmax equation to find $P(x_i)$ for each choice i.
4. *Random Sampling*: generate a random number between 0 and 1 and use it to sample from the calculated probability distribution.

Example
Let's say a person is deciding between three food options: pizza, salad, and pasta. Their utilities for these are 2, 1, and 1.5 respectively. With $\tau = 1$:

1. *Compute the Exponentials*:
   - Pizza: $e^{(2/1)} = e^2 \approx 7.39$
   - Salad: $e^{(1/1)} = e^1 \approx 2.72$
   - Pasta: $e^{(1.5/1)} = e^{1.5} \approx 4.48$
2. *Sum the Exponentials*: $7.39 + 2.72 + 4.48 = 14.59$
3. *Calculate Probabilities*:
   - Pizza: $\frac{7.39}{14.59} \approx 0.51$
   - Salad: $\frac{2.72}{14.59} \approx 0.19$
   - Pasta: $\frac{4.48}{14.59} \approx 0.31$
4. *Random Sampling*:
   - Calculate cumulative sums: starting with the first option, add up the probabilities sequentially. For example, if you have probabilities $[0.51, 0.19, 0.31]$, the cumulative sums would be $[0.51, 0.70, 1.00]$.
   - Generate Random Number: Generate a random number r between 0 and 1.
   - Find Interval: Compare r against the cumulative sums to find the interval it falls into. If r is 0.6, you would see it falls into the second interval (between 0.51 and 0.70). This indicates that the choice corresponds to the second option, as it falls between the cumulative probabilities for the first and the second options.

Acknowledgements

I would like to acknowledge Priyam Das who helped to provide Python code and Abhilasha Kumar who helped with the development of the theoretical framework for the assistance game.