# Ten simple rules for the computational modeling of behavioral data

Robert C. Wilson[1,2,*] and Anne GE Collins[3,4,†]

[1]Department of Psychology, University of Arizona
[2]Cognitive Science Program, University of Arizona
[3]Department of Psychology, University of California, Berkeley
[4]Helen Wills Neuroscience Institute, University of California, Berkeley
[*]bob@arizona.edu
[†]annecollins@berkeley.edu

January 4, 2019

**Abstract**

Computational modeling of behavioral data has revolutionized psychology and neuroscience. By fitting models to experimental data we can probe the algorithms underlying behavior, find neural correlates of computational variables and more precisely understand the effects of drugs, illness and interventions. But with great power comes great responsibility. In this note we give ten simple rules to ensure that computational modeling is used with care.

## What is the computational modeling of behavioral data?

The goal of computational modeling in behavioral science is to use precise mathematical models to make better sense of behavioral data. The behavioral data most often come in the form of choices, but can also be reaction times, eye movements, or other easily observable behaviors. The models come in the form of mathematical equations that link the experimentally observable variables (e.g. stimuli, outcomes, past experiences) to behavior in the immediate future. In this sense, computational models instantiate different 'algorithmic hypotheses' about how behavior is generated.

Exactly what it means to 'make sense' of behavioral data is, to some extent, a matter of taste that will vary according to the researcher's goals (Kording, Blohm, Schrater, & Kay, 2018). In some cases, a simple model that can explain broad qualitative features of the

data is enough. In other case, more detailed models that make quantitative predictions are required (Breiman et al., 2001). The exact form of the models, and exactly what we do with them, is limited only by our imaginations, but four uses dominate the literature: simulation, parameter estimation, model comparison, and latent variable inference.

**Simulation** involves running the model with particular parameter settings to generate 'fake' behavioral data. This simulated data can then be analyzed in much the same way as one would analyze real data, to make precise, falsifiable predictions about qualitative and quantitative patterns in the data. Simulation is a way to make theoretical predictions more precise and testable. Some examples include (Cohen, Dunbar, & McClelland, 1990; A. G. Collins & Frank, 2014; Rescorla, Wagner, et al., 1972; Farashahi, Rowe, Aslami, Lee, & Soltani, 2017; Montague, Dayan, & Sejnowski, 1996b; Abbott et al., 2015).

**Parameter estimation** involves finding the set of parameter values that best account for real behavioral data for a given model. These parameters can be used as a succinct summary of a given data set (Ratcliff, 1978; Wilson, Nassar, & Gold, 2013; Daw, Gershman, Seymour, Dayan, & Dolan, 2011), for investigating individual differences (Frank, Moustafa, Haughey, Curran, & Hutchison, 2007; Starns & Ratcliff, 2010; A. G. Collins & Frank, 2012; Gillan, Kosinski, Whelan, Phelps, & Daw, 2016; Somerville et al., 2017) and quantifying the effects of interventions such as drugs, lesions, illness, or experimental conditions (Frank, Seeberger, & O'reilly, 2004; Lorains et al., 2014; Dowd, Frank, Collins, Gold, & Barch, 2016; Zajkowski, Kossut, & Wilson, 2017; Warren et al., 2017; Wimmer, Li, Gorgolewski, & Poldrack, 2018).

**Model comparison** involves trying to compute which of a set of possible models best describes the behavioral data, as a way to understand which mechanisms underlie behavior. This is especially useful when the different models make similar qualitative predictions but differ quantitatively (Wilson & Niv, 2012; Daw et al., 2011; A. Collins & Koechlin, 2012; A. G. Collins & Frank, 2012; Fischer & Ullsperger, 2013; Steyvers, Lee, & Wagenmakers, 2009).

**Latent variable inference** involves using the model to compute the values of hidden variables (for example values of different choices) that are not immediately observable in the behavioral data, but that the theory assumes are important for the computations occurring in the brain. Latent variable inference is especially useful in neuroimaging where it is used to help search for the neural correlates of the model (O'doherty, Hampton, & Kim, 2007; Wilson & Niv, 2015; Donoso, Collins, & Koechlin, 2014; Cohen et al., 2017), but also for EEG, ECOG, electrophysiology and pupillometry among many others (OReilly et al., 2013; A. G. Collins & Frank, 2018; Samejima, Ueda, Doya, & Kimura, 2005; Cavanagh, Wiecki, Kochar, & Frank, 2014; Nassar et al., 2012).

Each of these uses has its strengths and weaknesses, and each of them can be mishandled in a number of ways, causing us draw to wrong and misleading conclusions (Nassar & Frank, 2016; Palminteri, Wyart, & Koechlin, 2017). Here we present a beginner-friendly, pragmatic, practical and details oriented introduction (complete with example

code available at (Wilson & Collins, 2019)) on how to relate models to data and avoid many of these modeling mistakes. Our goal for this paper is to go beyond the mere mechanics of implementing models — as important as those mechanics are — and instead focus on the harder question of how to figure out what, exactly, a model is telling us about the mind. For this reason we focus primarily on the simplest modeling techniques most accessible to beginning modelers, but almost all of our points apply more generally and readers interested in more advanced modeling techniques should consult the many excellent tutorials, didactic examples, and books on the topic (Busemeyer & Diederich, 2010; Daw, 2011; Daw & Tobler, 2014; Heathcote, Brown, & Wagenmakers, 2015; Huys, 2017; Turner et al., 2013; Vandekerckhove, Matzke, & Wagenmakers, 2015; Wagenmakers & Farrell, 2004; Rigoux, Stephan, Friston, & Daunizeau, 2014; Nilsson, Rieskamp, & Wagenmakers, 2011; Farrell & Lewandowsky, 2018). Our hope is that, regardless of the techniques you use, by following these 10 simple steps (Figure 1), you will be able to minimize your modeling mishaps and unleash the power of computational modeling on your own behavioral data!

# 1 Design a good experiment!

Computational modeling is a powerful technique, but it can never replace good experimental design. Modeling attempts to capture how information is manipulated behind the scenes to produce the behavior; thus it is fundamentally limited by the behavioral data, which is itself fundamentally limited by the experimental protocol. A researcher studying perception would not attempt to fit a reinforcement learning model to a perceptual decision making task; and a researcher studying the differential effects of gain and loss would not do it in a gambling task with only gains. While obvious in these simple cases, the question becomes more difficult as the complexity of the model increases: is a given learning protocol rich enough to allow identification of dynamic changes in learning rate? of working memory or episodic memory contributions to learning? of reward range adaptation? Often, the answer to these questions will be 'no' unless the protocol has been deliberately designed to provide this power.

So, how should you go about designing a good experiment with computational modeling in mind? While this process will always be something of an art form, we suggest you ask yourself the following questions to optimize your experimental design:

**What scientific question are you asking?** While this sounds obvious, it is easy to get sucked into an experiment design without ever asking the most basic questions about your goals. What cognitive process are you targeting? What aspect of behavior are you trying to capture? What hypotheses are you trying to pick apart? For example, you may be trying to identify how working memory contributes to learning or how behavioral variability can be used to explore. Keeping your scientific goals in mind when you design the task can save much time later on.

**Does your experiment engage the targeted processes?** This may be a difficult question to answer, and require expert knowledge/piloting. However, you need to know that the experimental design actually engages the processes you are trying to model.

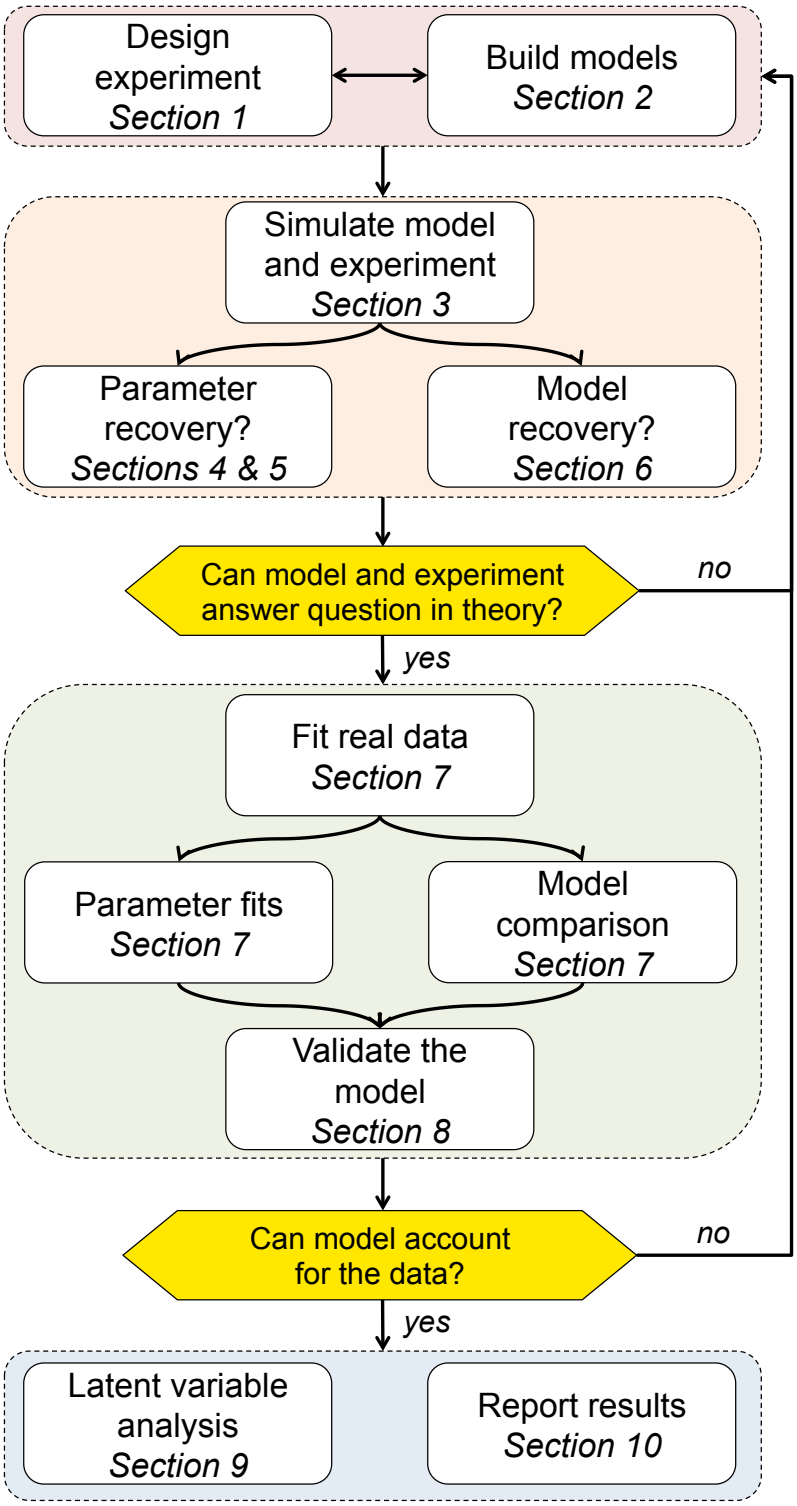


Figure 1: Schematic of the 10 rules and how they translate into a process for using computational modeling to better understand behavior.

**Will signatures of the targeted processes be evident from simple statistics of the data?** In addition to engaging the processes of interest, the best experiments make these processes identifiable in classical analyses of the behavioral data. For example, if you are investigating working memory contributions to learning, you may look for a signature of load on behavior by constructing an experimental design that varies load, to increase chances of probing working memory's role in learning. Seeing evidence of the computations of interest in simple analyses of behavior builds confidence that the modeling process will actually work. **In our experience, computational modeling is rarely informative when there is no evidence of an effect in model-independent analyses of behavior.**

To answer these questions, it is important to have a clear theoretical hypothesis of what phenomenon is to be modeled. In fact, although designing a good experiment is the first step, it goes hand-in-hand with designing a good model, and the two steps should be done in parallel.

**An illustrative example: the multi-armed bandits task**

While the ten rules in this paper are quite general, we will illustrate many of our points using simple examples from our own field of reinforcement learning. Code for implementing all of these examples is available on GitHub (Wilson & Collins, 2019). The goal of these example studies is to understand how people learn to maximize their rewards in a case where the most rewarding choice is initially unknown.

More specifically, we consider the case in which a participant makes a series of $T$ choices between $K$ slot machines, or 'one-armed bandits', to try to maximize their earnings. If played on trial $t$, each slot machine, $k$, pays out a reward, $r_t$, which is 1 with reward probability, $\mu_t^k$, and otherwise 0. The reward probabilities are different for each slot machine and are initially unknown to the subject. In the simplest version of the task, the reward probabilities are fixed over time.

The three **experimental parameters** of this task are: the number of trials, $T$, the number of slot machines, $K$, and the reward probabilities of the different options, $\mu_t^k$, which may or may not change over time. The settings of these parameters will be important for determining exactly what information we can extract from the experiment. In this example we will assume that $T = 1000$, $K = 2$, and that the reward probabilities are $\mu_t^1 = 0.2$ for slot machine 1 and $\mu_t^2 = 0.8$ for slot machine 2.

# 2 Design good models

Just as bad experiments can limit our ability to test different hypotheses, bad models – quite literally the mathematical embodiment of our hypotheses – can further limit the conclusions we can draw. This point is especially important if we are designing new models, but even well established computational models can be problematic in some cases (Broomell & Bhatia, 2014).

There are a number of different approaches for designing models that have been successfully used in the literature. Perhaps the simplest approach is to use heuristics to find

a 'reasonable' way to handle information to produce the target behavior. This approach was how the delta rule (see Model 3 below) was first invented (Rescorla et al., 1972). Another approach is to scour the artificial intelligence, computer science, and applied mathematics literature for algorithms that have been used to solve similar problems for artificial agents. This approach has been fruitfully applied in the field of reinforcement learning, where algorithms such as Q-learning and temporal difference learning have been related to human and animal behavior and brain function (Watkins & Dayan, 1992; Montague, Dayan, & Sejnowski, 1996a). Another approach is to take a Bayes-optimal perspective, to design algorithms that perform optimally given a model of the environment and the task. Ideal observer models in vision are one example where this approach has been applied successfully (Geisler, 2011). More generally, Bayes-optimal models can be further pursued by investigating simpler algorithms that approximate the ideal strategy, or by imposing bounded rationality constraints, such as limited computational resources, on ideal observer agents (Courville & Daw, 2008; Nassar, Wilson, Heasly, & Gold, 2010; A. Collins & Koechlin, 2012; Daw & Courville, 2007; Lieder, Griffiths, Huys, & Goodman, 2018).

Regardless of the approach (or, better yet, approaches) you take to design your models, it is important to keep the following points in mind:

**A computational model should be as simple as possible, but no simpler.** Einstein's old edict applies equally to models of the mind as it does to models of physical systems. Simpler, more parsimonious models are easier to fit and easier to interpret and should always be included in the set of models under consideration. Indeed, formal model comparison techniques (described in detail in Appendix B) include a penalty for overly complex models, which are more likely to overfit the data and generalize poorly, favoring simpler models so long as they can account for the data.

**A computational model should be interpretable (as much as possible).** In the process of developing models that can account for the behavioral data, researchers run the risk of adding components to a model that are not interpretable as a sensible manipulation of information. For example, a negative learning rate is difficult to interpret in the framework of reinforcement learning. While such uninterpretable models may sometimes improve fits, nonsensical parameter values may indicate that something important is missing from your model, or that a different cognitive process altogether is at play.

**The models should capture *all* the hypotheses you plan to test.** While it is obviously important to design models that can capture your main hypothesis, it is even more important to design models that capture competing hypotheses. Crucially, competing models should not be strawmen - they should have a genuine chance of relating to behavior in the task environment - and you should put equal effort into fitting these models as you do your favored hypothesis. Better yet, you shouldn't have a favored hypothesis at all — let the data determine which model is the best fit, not your *a priori* commitment to one model or another.

**Example: Modeling behavior in the multi-armed bandits task**

We consider five different models of how participants could behave in the multi-armed bandits task.

**Model 1: Random responding.** In the first model, we assume that participants do not engage with the task at all and simply press buttons at random, perhaps with a bias for one option over the other. Such random behavior is not uncommon in behavioral experiments, especially when participants have no external incentives for performing well. Modeling such behavior can be important if we wish to identify such 'checked out' individuals in a quantitative and reproducible manner, either for exclusion or to study the checked-out behavior itself. To model this behavior we assume that participants choose between the two options randomly, perhaps with some overall bias for one option over the other. This bias is captured with a parameter $b$ (which is between 0 and 1) such that the probability of choosing the two options is

$$p_t^1 = b \quad \text{and} \quad p_t^2 = 1 - b \tag{1}$$

Thus, for two bandits, the random responding model has just one free parameter, controlling the overall bias for option 1 over option 2, $\boldsymbol{\theta}_1 = b$.

**Model 2: Noisy win-stay-lose-shift.** The win-stay-lose-shift model is one of the simplest models that adapts its behavior according to feedback. Consistent with the name, the model repeats rewarded actions and switches away from unrewarded actions. In the noisy version of the model, the win-stay-lose-shift rule is applied probabilistically, such that the model applies the win-stay-lose-shift rule with probability $1-\epsilon$, and chooses randomly with probability $\epsilon$. In the two-bandit case, the probability of choosing option $k$ is

$$p_t^k = \begin{cases} 1 - \epsilon/2 & \text{if } (c_{t-1} = k \text{ and } r_{t-1} = 1) \text{ OR } (c_{t-1} \neq k \text{ and } r_{t-1} = 0) \\ \epsilon/2 & \text{if } (c_{t-1} \neq k \text{ and } r_{t-1} = 1) \text{ OR } (c_{t-1} = k \text{ and } r_{t-1} = 0) \end{cases} \tag{2}$$

where $c_t = 1, 2$ is the choice at trial $t$, and $r_t = 0, 1$ the reward at trial $t$. While more complex to implement, this model still only has one free parameter, the overall level of randomness, $\boldsymbol{\theta}_2 = \epsilon$.

**Model 3: Rescorla Wagner.** In this model, participants first *learn* the expected value of each slot machine based on the history of previous outcomes and then use these values to make a *decision* about what to do next. A simple model of learning is the Rescorla-Wagner learning rule (Rescorla et al., 1972) whereby the value of option $k$, $Q_t^k$ is updated in response to reward $r_t$ according to

$$Q_{t+1}^k = Q_t^k + \alpha(r_t - Q_t^k) \tag{3}$$

where $\alpha$ is the learning rate, which takes a value between 0 and 1 and captures the extent to which the prediction error, $(r_t - Q_t^k)$, updates the value. For simplicity, we assume that the initial value, $Q_0^k$, is zero, although it is possible to treat the $Q_0^k$ as a free parameter of the model.

A simple model of decision making is to assume that participants use the options' values to guide their decisions, choosing the most valuable option most frequently, but occasionally making 'mistakes' (or exploring) by choosing a low value option. One choice rule with these properties is known as the 'softmax' choice rule, which chooses option $k$ with probability

$$p_t^k = \frac{\exp(\beta Q_t^k)}{\sum_{i=1}^{K} \exp(\beta Q_t^i)} \tag{4}$$

where $\beta$ is the 'inverse temperature' parameter that controls the level of stochasticity in the choice, ranging from $\beta = 0$ for completely random responding and $\beta = \infty$ for deterministically choosing the highest value option.

Combining the learning (equation 3) and decision rules (equation 4) gives a simple model of decision making in this task with two free parameters: the learning rate, $\alpha$, and the inverse temperature $\beta$. That is, in our general notation, for this model $\boldsymbol{\theta}_3 = (\alpha, \beta)$.

**Model 4: Choice kernel.** This model tries to capture the tendency for people to repeat their previous actions. In particular, we assume that participants compute a 'choice kernel,' $CK_t^k$, for each action, which keeps track of how frequently they have chosen that option in the recent past. This choice kernel updates in much the same way as the values in the Rescorla-Wagner rule, i.e. according to

$$CK_{t+1}^k = CK_t^k + \alpha_c(a_t^k - CK_t^k) \tag{5}$$

where $a_t^k = 1$ if option $k$ is played on trial $t$, otherwise $a_t^k = 0$, and $\alpha_c$ is the choice-kernel learning rate. For simplicity we assume that the initial value of the choice kernel is always zero, although, like the initial $Q$-value in the Rescorla-Wagner model, this could be a parameter of the model. Note that with $\alpha_c = 1$, this model is very similar to model 2 (win-stay-lose-shift). From there, we assume that each option is chosen according to

$$p_t^k = \frac{\exp(\beta_c CK_t^k)}{\sum_{i=1}^{K} \exp(\beta_c CK_t^i)} \tag{6}$$

where $\beta_c$ is the inverse temperature associated with the choice kernel.

Combining the choice kernel (equation 5) with the decision rule (equation 6) gives a simple model of decision making in this task with two free parameters: the choice-kernel learning rate, $\alpha_c$, and the choice-kernel inverse temperature $\beta_c$. That is, in our general notation, for this model $\boldsymbol{\theta}_4 = (\alpha_c, \beta_c)$.

**Model 5: Rescorla Wagner + choice kernel.** Finally, our most complex model mixes the reinforcement learning model with the choice kernel model. In this model, the values update according to equation 3, while the choice kernel updates according to equation 5. The terms are then combined to compute the choice probabilities as

$$p_t^k = \frac{\exp(\beta Q_t^k + \beta_c C K_t^k)}{\sum_{i=1}^{K} \exp(\beta Q_t^i + \beta_c C K_t^i)} \tag{7}$$

This most complex model has four free parameters, i.e. $\boldsymbol{\theta}_5 = (\alpha, \beta, \alpha_c, \beta_c)$

# 3 Simulate, simulate, simulate!

Once you have an experimental design and a set of computational models, a really important step is to create *fake*, or *surrogate* data. That is, you should use the models to simulate the behavior of participants in the experiment, and to observe how behavior changes with different models, different model parameters, and different variants of the experiment. This step will allow you to refine the first two steps: confirming that the experimental design elicits the behaviors assumed to be captured by the computational model. To do this, here are some important steps.

**Define model-independent measures that capture key aspects of the processes you are trying to model.** Finding qualitative signatures (and there will often be more than one) of the model is crucial. By studying these measures with simulated data you will have greater intuition about what is going on when you use the same model-independent measures to analyze real behavior (Daw et al., 2011; A. G. Collins & Frank, 2012; A. Collins & Frank, 2013; Nassar, Helmers, & Frank, 2018).

**Simulate the model across the range of parameter values.** Then, visualize behavior as a function of the parameters. Almost all models have free parameters. Understanding how changes to these parameters affect behavior will help you to better interpret your data and understand individual differences in fit parameters. For example, in probabilistic reinforcement learning tasks modeled with a simple delta-rule model (Model 3; Equation 3), the learning rate parameter, $\alpha$, can relate to both speed of learning and noisiness in asymptotic behavior, as can the inverse temperature parameter, $\beta$ (in Equation 4), as seen in Figure 2B.

**Visualize the simulated behavior of different models.** This will allow you to verify that behavior is qualitatively different for different models, making their predictions in the experimental setup different (Figure 2A). If the behavior of different models is *not* qualitatively different, this is a sign that you should try to design a better experiment. While not always possible, distinguishing between models on the basis of qualitative patterns in the data is always preferable to quantitative model comparison (Navarro, 2018).

More generally, the goal of the simulation process is to clarify how the models and experimental design satisfy your goal of identifying a cognitive process in behavior. If the

answer is positive - i.e. the experiment is rich enough to capture the expected behavior, the model's parameters are interpretable, and competing models make dissociable predictions - you can move on to the next step. Otherwise, you should loop back through these first three sections to make sure that your experimental design and models work well together, and that model parameters have identifiable effects on the behavior, which is a prerequisite for the fourth step, fitting the parameters (c.f. Figure 1).

**Example: Simulating behavior in the bandits task**

To simulate behavior we first need to define the parameters of the task. These include the total number of trials, $T$ (= 1000 in the example), as well as the number of bandits, $K(= 2)$, and the reward probability for each bandit, $\mu^k$ ( = 0.2 and 0.8 for bandits 1 and 2 respectively). The experiment parameters should match the actual parameters used in the experiment.

Next we define the parameters of the model. One way to do this is to sample these parameters randomly from prior distributions over each parameter, the exact form of which will vary from model to model. These prior distributions should generally be as broad as possible, but if something is known about the distribution of possible parameter values for a particular model, this is one place to include it.

With the free parameters set we then proceed with the simulation. First we simulate the choice on the first trial, $a_1$, by assuming that the model chooses option $k$ with probability, $p_1^k$. Next we simulate the outcome, $r_1$, of this choice. In Models 2-5, we use the action and/or outcome to update the choice probabilities for the next trial. Repeating this process for all trials up to $t = T$ completes one simulation. The simulations can then be analyzed in the same way participants' data is, ideally with the same code taking different inputs. This process should be repeated several times, with different parameter settings, to get a handle on how the model behaves as a function of its parameters.

To illustrate how one might visualize the simulated results we look at two model-independent measures: the probability of repeating an action, $p(\text{stay})$, and the probability of choosing the correct option, $p(\text{correct})$. In Figure 2A below we plot $p(\text{stay})$ as a function of the reward on the last trial for each of the models with a particular set of parameters (M1: $b = 0.5$, M2: $\epsilon = 0.05$, M3: $\alpha = 0.1$, $\beta = 5$, M4: $\alpha_c = 0.1$, $\beta_c = 3$, M5: $\alpha = 0.1$, $\beta = 5$, $\alpha_c = 0.1$, $\beta_c = 1$). For some models (in particular the win-stay-lose-shift model, Model 2) we expect a strong dependence on past reward, but for others (such as the random responder, Model 1) we expect no dependence.

A more thorough exploration of the parameter space for Model 3 is shown in Figure 2B, where we plot the $p(\text{correct})$ in the first and last 10 trials as a function of the learning rate, $\alpha$, and softmax parameter, $\beta$.

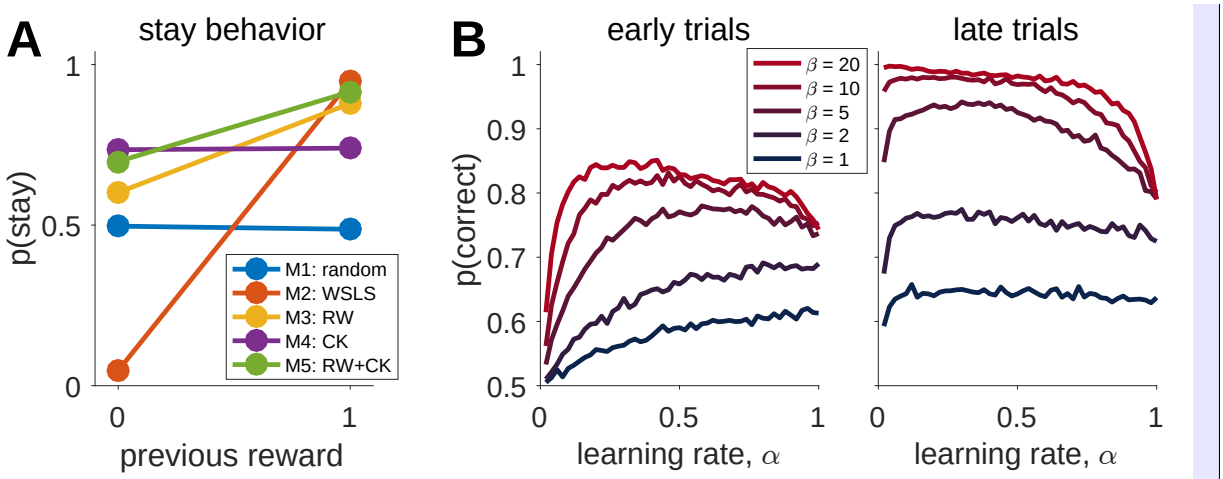


Figure 2: Simulating behavior in the a two-armed bandit task. (**A**) Win-stay-lose-shift behavior varies widely between models. (**B**) Model 3 simulations (100 per parameter setting) show how the learning rate and softmax parameters influence two aspects of behavior: early performance (first 10 trials), and late performance (last 10 trials). The left graph shows that learning rate is positively correlated with early performance improvement only for low $\beta$ values, or very low $\alpha$ values. For high $\beta$ values, there is a U-shape relationship between learning rate and early speed of learning. The right graph shows that with high $\beta$ values, high learning rates negatively influence asymptotic behavior. Thus, both parameters interact to influence both the speed of learning and asymptotic performance.

# 4 Fit the parameters

A key component of computational modeling is estimating the values of the parameters that best describe your behavioral data. While there are a number of different ways of estimating parameters, here we focus on the maximum-likelihood approach, although almost all of our points apply to other methods such as Markov Chain Monte Carlo approaches (Lee & Wagenmakers, 2014). Mathematical details, as well as additional discussion of other approaches to model fitting can be found in Appendix A.

In the maximum likelihood approach to model fitting, our goal is to find the parameter values of model $m$, $\hat{\boldsymbol{\theta}}_m^{MLE}$, that maximize the likelihood of the data, $d_{1:T}$, given the parameters, $p(d_{1:T}|\boldsymbol{\theta}_m, m)$. Maximizing the likelihood is equivalent to maximizing the log of the likelihood, $LL = \log p(d_{1:T}|\boldsymbol{\theta}_m, m)$, which is numerically more tractable[1]. A simple mathematical derivation shows that this log-likelihood can be written in terms of the choice probabilities of the individual model as

$$LL = \log p(d_{1:T}|\boldsymbol{\theta}_m, m) = \sum_{t=1}^{T} \log p(c_t|d_{1:t-1}, s_t, \boldsymbol{\theta}_m, m) \tag{8}$$

[1]the likelihood is a product of many numbers smaller than 1, which can be rounded to 0 with limited precision computing. By contrast, the log-likelihood is a sum of negative numbers, which usually is tractable and will not be rounded to 0.

where $p(c_t|d_{1:t-1}, s_t, \boldsymbol{\theta}_m, m)$ is the probability of each individual choice given the parameters of the model and available information up to that choice, which is at the heart of the definition of each model (for example in Equations 1-7).

In principle, finding the maximum likelihood parameters is as 'simple' as maximizing $LL$. In practice, of course, finding the maximum of a function is not a trivial process. The simplest approach, a brute force search of the entire parameter space, is occasionally useful, and may help understand how different parameters interact (see Figure 3). However, this approach is unfeasible outside of the simplest cases (e.g. one or two parameters with tight bounds) due to the high computational costs of evaluating the likelihood function at a large number of points.

Fortunately, a number of tools exist for finding local maxima (and minima) of functions quickly using variations on gradient ascent (or descent). For example, Matlab's fmincon function can use a variety of sophisticated optimization algorithms (e.g. (Moré & Sorensen, 1983; Byrd, Gilbert, & Nocedal, 2000)) to find the minimum of a function. So long as one remembers to feed fmincon the *negative* log-likelihood (whose minimum is at the same parameter values as the maximum of the positive log-likelihood), using tools such as fmincon can greatly speed up model fitting. Even here, though, a number of problems can arise when trying to maximize $LL$ that can be reduced by using tips and tricks below. Most of the tips come from understanding that optimization algorithms are not foolproof and in particular are subject to numerical constraints.

**Be sure your initial conditions give finite log-likelihoods.** Optimizers such as fmincon require you to specify initial parameter values from which to start the search. Perhaps the simplest way in which the search process can fail is if these initial parameters give log-likelihoods which are not finite numbers (e.g. infinities or NaNs, not a number in Matlab speak). If your fitting procedure fails this can often be the cause.

**Beware rounding errors, zeros and infinities.** More generally, the fitting procedure can go wrong if it encounters infinities or NaNs during the parameter search. This can occur if a choice probability is rounded down to zero, thus making the log of the choice probability $-\infty$. Likewise if your model involves exponentials (e.g. the softmax choice rule in Equation 4), this can lead to errors whereby the exponential of a very large number is 'rounded up' to infinity. One way to avoid these issues is by constraining parameter values to always give finite choice probabilities and log-likelihoods at the boundaries. One way to diagnose these issues is to include checks in the code for valid log-likelihoods.

**Be careful with constraints on parameters.** If the constraints are ill chosen, it is possible that the solution will be at the bounds, which is often a red flag.

**Only include parameters that have an influence on the likelihood.** If only two parameters impact the likelihood, but the optimizer attempts to fit three, it will usually find the optimum for the two relevant parameters and a random value for the third; however, it will lead to slower and less efficient fitting.

**Beware local minima!** Finally, a key limitation of optimizers like fmincon is that they are only guaranteed to find *local* minima which are not guaranteed to be the *global* minima corresponding to the best fitting parameters. One way to mitigate this issue is to run the fitting procedure multiple times with random initial conditions, recording the best fitting log-likelihood for each run. The best fitting parameters are then the parameters corresponding to the run with the highest log-likelihood. There is no hard-and-fast rule for knowing how many starting points to use in a given situation, besides the fact that more complex models will require more starting points. Thus, it must be determined empirically in each case. One way to validate the number of starting points is by plotting the best likelihood score as a function of the number of starting points. As the number of initial conditions increases, the best fitting likelihood (and corresponding parameters) will improve up to an asymptote close to the true maximum of the function (e.g. Figure 3).

**Example: Contending with multiple local maxima**

As a real example with local maxima we consider the case of a simplified version of the mixed reinforcement learning and working memory model from (A. G. Collins & Frank, 2012). For simplicity, we relegate the details of this model to Appendix C. To appreciate the example all one really needs to know is that in its simplest version, this model has two parameters: $\rho$, which captures the effect of working memory, and $\alpha$, which captures the learning rate of reinforcement learning. As is seen in Figure 3 below, this model (combined with an appropriate experiment) gives rise to a log-likelihood surface with multiple local maxima. Depending on the starting point, the optimization procedure can converge to any one of these local maxima, meaning that the 'maximum' likelihood fits may not reflect the global maximum likelihood.

To mitigate this concern, a simple and effective approach is to repeat the optimization procedure many times, keeping track of the best fitting log-likelihood and parameters in each case. An approximation to the global maximum is to take the best log-likelihood from this list of fits. Results of this multiple iteration procedure can be summarized by plotting the best log-likelihood as a function of the number of starting points, or similarly the distance from the so-far best parameters to the final best parameters as a function of the number of starting points (Figure 3B). As the number of starting points increases, the best-fitting log-likelihood and parameters will converge to the global maximum. This plot also allows us to judge when we have used enough starting points - specifically, if the best fitting parameters appears to have reached asymptote, that gives us a good indication that the fit is the best we can do.

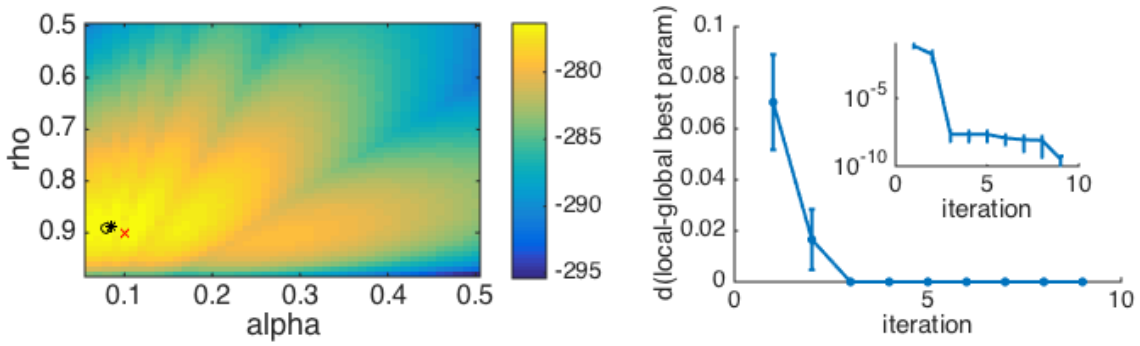


Figure 3: An example with multiple local minima. (**A**) Log-likelihood surface for a working memory reinforcement learning model with two parameters. In this case there are several local minima, all of which can be found by the optimization procedure depending on the starting point. Red x: generative parameters; black circle: optimum with brute search method; black *: optimum with fmincon and multiple starting points (**B**) Plotting the distance from the best fitting parameters after $n$ iterations to the best fitting parameters after all iterations as a function of the number of starting points $n$, gives a good sense of when the procedure has found the global optimum.

# 5 Check you can recover your parameters

Before reading too much into the best fitting parameter values, $\boldsymbol{\theta}_m^{MLE}$, it is important to check whether the fitting procedure gives meaningful parameter values in the best case scenario - i.e. when fitting fake data where the 'true' parameter values are known. Such a procedure is known as 'Parameter Recovery,' and is a crucial part of any model-based analysis.

In principle, the recipe for parameter recovery is quite simple. First, simulate fake data with known parameter values. Next, fit the model to this fake data to try to 'recover' the parameters. Finally, compare the recovered parameters to their true values. In a perfect world the simulated and recovered parameters will be tightly correlated, with no bias. If there is only a weak correlation between simulated and recovered parameters and/or a significant bias, then this is an indication that there is either a bug in your code (which from our own experience we suggest is fairly likely) or the experiment is underpowered to assess this model.

To make the most of your parameter recovery analysis, we suggest the following tips:

**Make sure your simulation parameters are in the right range.** An important choice for parameter recovery is the range of simulation parameters you wish to recover. Some models/experiments only give good parameter recovery for parameters in a particular range - if the simulation parameters are too big or too small they can be hard to recover. An illustration of this is the softmax parameter, $\beta$, where very large $\beta$ values lead to almost identical behavior in most experiments. Thus parameter recovery may fail for large $\beta$ values, but work well for small $\beta$ values. Of course, selecting

only the range of parameters that *can* be recovered by your model is not necessarily the right choice, especially if the parameter values you obtain when fitting real data are outside of this range! For this reason we have the following recommendations for choosing simulation parameter values:

1. If you have already fit your data we recommend matching the range of your simulation parameters to the range of values obtained by your fit.
2. If you have not fit your data, but you are using a model that has already been published, match the range of parameters to the range seen in previous studies.
3. Finally, if the model is completely new and the 'true' parameter values are unknown, we recommend simulating over as wide a range as possible to get a sense of whether and where parameters can be recovered. You can rely on your exploration of how model parameters affect simulated behavior to predict a range beyond which parameters will not affect behavior much.

**Plot the correlations between simulated and recovered parameters.** While the correlation coefficient between simulated and recovered parameters is a useful number for summarizing parameter recovery, we also strongly recommend that you actually plot the simulated vs recovered parameters. This makes the correlation clear, and also reveals whether the correlation holds in some parameter regimes but not others. It also reveals any existing bias (e.g. a tendency to recover higher or lower values in average).

**Make sure the recovery process does not introduce correlations between parameters.** In addition to looking at the correlations between simulated and recovered parameters, we also recommend looking at the correlation between the recovered parameters themselves. If the simulation parameters are uncorrelated with one another, correlations between the recovered parameters is an indication that parameters in the model are trading off against one another(Daw, 2011). Such trade-offs can sometimes be avoided by reparameterizing the model (e.g. (Otto, Raio, Chiang, Phelps, & Daw, 2013)) or redesigning the experiment. Sometimes, however, such trade-offs are unavoidable. In these cases it is crucial to report the trade-off in parameters so that a 'correlation' between fit parameter values is not over-interpreted in real data.

**A note about parameter differences between different populations or conditions.** A growing use of model fitting is to compare parameter values between populations (e.g. schizophrenia patients vs healthy controls (A. G. Collins, Brown, Gold, Waltz, & Frank, 2014)) or conditions (e.g. transcranial magnetic stimulation to one area or another (Zajkowski et al., 2017)). If your primary interest is a difference like this, then parameter recovery can be used to give an estimate of statistical power. In particular, for a proposed effect size (e.g. on the average difference in one parameter between groups / conditions) you can simulate and recover parameters for the groups/conditions and then perform statistical tests to detect group differences in this simulated data set. The power for this effect size is then the frequency with which the statistical tests detect no effect given that the effect is there.

**Remember that even successful parameter recovery represents a best-case scenario!** What does successful parameter recovery tell you? That data generated by a known model with given parameters can be fit to recover those parameters. This is the best case you could possibly hope for in the model-based analysis and is unlikely to ever occur as the 'true' generative process for behavior — i.e. the inner workings of the mind and brain — is likely much more complex than any model you could conceive. There's no easy answer to this problem. We only advise that you remember to be humble when you present your results!

**Example: Parameter recovery in the reinforcement learning model**

We performed parameter recovery with Model 3, the Rescorla Wagner model, on the 2-armed bandits task. As before, we set the means of each bandit at $\mu_1 = 0.2$ and $\mu_2 = 0.8$ and the number of trials, $T = 1000$. We then simulated actions of the model according to Equations 3 and 4, with learning rate, $\alpha$, and softmax temperature, $\beta$, set according to

$$\alpha \sim U(0,1) \quad \text{and} \quad \beta \sim \text{Exp}(10) \tag{9}$$

After simulating the model, we fit the parameters using a maximum likelihood approach to get fit values of learning rate, $\alpha$, and softmax parameter, $\beta$. We then repeated this process 100 times using new values of $\alpha$ and $\beta$ each time. The results are plotted in Figure 4 below. As is clear from this plot, there is fairly good agreement between the simulated and fit parameter values. In addition, we can see that the fit for $\beta$ is best with a range, $0.1 < \beta < 10$ and that outside this range the correspondence between simulation and fit is not as good. Depending on the values of $\beta$ that we obtain by fitting human behavior this worse correspondence at small and large $\beta$ may or may not be problematic. It may be a good idea to use the range of parameters obtained from fitting the real data to test the quality of recovery within the range that matters.

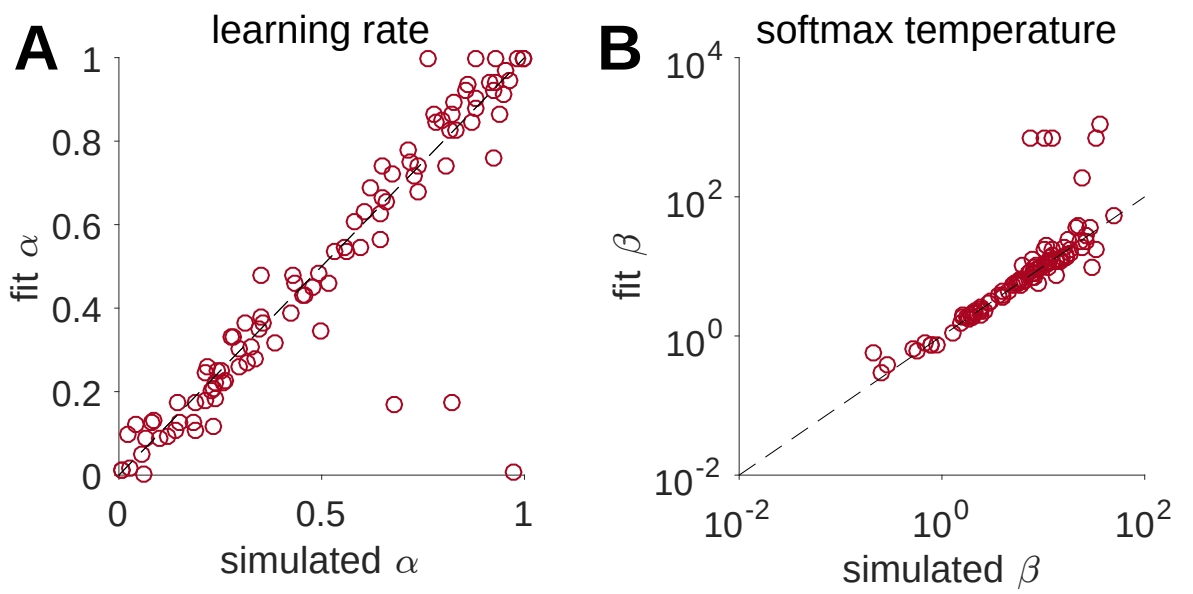

Figure 4: Parameter recovery for the Rescorla Wagner model (model 3) in the bandits task with 1000 trials.

# 6 Can you arbitrate between different models?

In model comparison, our goal is to determine which model, out of a set of possible models, is most likely to have generated the data. There are a number of different ways to make this comparison which are briefly summarized in Appendix B. Here we focus on the most common method which is related to the log-likelihood computed in Section 4.

A simplistic approach to model comparison would be to compare the log-likelihoods of each model at the best fitting parameter settings, $p(d_{1:T}|\hat{\boldsymbol{\theta}}_m, m)$. However, if the data, $d_{1:T}$, used to evaluate the log-likelihood are the same data used to fit the parameters, then this approach will lead to overfitting, as the model with most free parameters will almost always fit this 'training' data best. As an extreme example, consider the case of a model with one 'parameter' per choice which is the identity of the choice the person actually made. Such a 'model' would fit the data perfectly, but would of course tell us nothing about how the choices were actually determined and make no predictions about what choices would be made in a different setting. Overfitting is a problem in that it decreases the generalizability of the model.

To avoid the overfitting problem it is necessary to somehow account for the degrees of freedom in the model. There are several methods for doing this (including penalties for free parameters and testing the fit on out of sample data) that are discussed in more detail in the Appendix. Here we focus on one of the simplest methods, the Bayes Information Criterion, $BIC$, which has an explicit penalty for free parameters.

$$BIC = -2 \log \hat{LL} + k_m \log(T) \tag{10}$$

where $\hat{LL}$ is the log-likelihood value at the best fitting parameter settings, and $k_m$ is the number of parameters in model $m$. The model with the smallest $BIC$ score is the model that best fits the data. Thus, the positive effect of $k_m$ in the last term corresponds to a penalty for models with large numbers of parameters.

While Equation 10 is simple enough to apply to find the model that, apparently, best fits your data, a number of additional steps are required to be sure that your model comparison results are meaningful.

**Validate model comparison with simulated data.** Just as parameter fitting should be validated by parameter recovery on simulated data, so should model comparison be validated by model recovery on simulated data. More specifically, this process involves simulating data from all models (with a range of parameter values as in the case of parameter recovery) and then fitting that data with all models to determine the extent to which fake data generated from model $A$ is best fit by model $A$ as opposed to model $B$. This process can be summarized in a **confusion matrix** (see Figure 5 below for an example) that quantifies the probability that each model is the best fit to data generated from the other models. In a perfect world the confusion

matrix will be the identity matrix, but in practice this is not always the case e.g. (Wilson & Niv, 2012).

**Compare different methods of model comparison.** If the confusion matrix has large off-diagonal components then you have a problem with model recovery. There are a number of factors that could cause this problem - ranging from a bug in the code to an underpowered experimental design. However, one cause that is worth investigating is whether you are using the wrong method for penalizing for free parameters. In particular, different measures penalize parameters in different ways which are 'correct' under different assumptions. If your confusion matrix is not diagonal, it may be that the assumptions underlying your measures (e.g. BIC) do not hold for your models, in which case it might be worth trying another metric for model comparison (e.g. AIC (Wagenmakers & Farrell, 2004), see Appendix B).

**The elephant in the room with model comparison.** As wonderful as it is to find that your model 'best' fits the behavioral data, the elephant in the room (or perhaps more correctly *not* in the room) with all model comparison is that it only tells you which model fits best **of the models you considered**. In and of itself, this is rather limited information as there are infinitely many other models that you did not consider. This makes it imperative to start with a good set of models that rigorously capture the competing hypotheses (that is, think hard in Step 2). In addition, it will be essential to validate (at least) your winning model (see Step 9) to show how simulating its behavior can generate the patterns seen in the data that you did not explicitly fit, and thus obtain an *absolute* measure of how well your model relates to your data.

**Example: Confusion matrices in the bandits task**

To illustrate model recovery, we simulated behavior of the five models on the two-armed bandits task. As before the means were set at $\mu_1 = 0.2$ and $\mu_2 = 0.8$, and the number of trials was set at $T = 1000$. For each simulation, model parameters were sampled randomly for each model. Each simulated data set was then fit to each of the given models to determine which model fit best (according to BIC). This process was repeated 100 times to compute the confusion matrices which are plotted below in Figure 5.

The difference between these two confusion matrices is in the priors from which the simulation parameters were sampled. In panel A parameters were sampled from the following priors:

| Model | Priors |
|---|---|
| Model 1 | $b \sim U(0,1)$ |
| Model 2 | $\epsilon \sim U(0,1)$ |
| Model 3 | $\alpha \sim U(0,1)$, $\beta \sim \text{Exp}(1)$ |
| Model 4 | $\alpha_c \sim U(0,1)$, $\beta_c \sim \text{Exp}(1)$ |
| Model 5 | $\alpha \sim U(0,1)$, $\beta \sim \text{Exp}(1)$, $\alpha_c \sim U(0,1)$, $\beta_c \sim \text{Exp}(1)$ |

In panel B, all of the softmax parameters $\beta$ and $\beta_c$ were increased by 1. This has the effect of reducing the amount of noise in the behavior, which makes the models more easily identifiable and the corresponding confusion matrix more diagonal. The fact that the confusion matrix can be so dependent on the simulating parameter values means that it is crucial to match the simulation parameters to the actual fit parameters as best as possible. **Models that are identifiable in one parameter regime may be impossible to distinguish in another!**

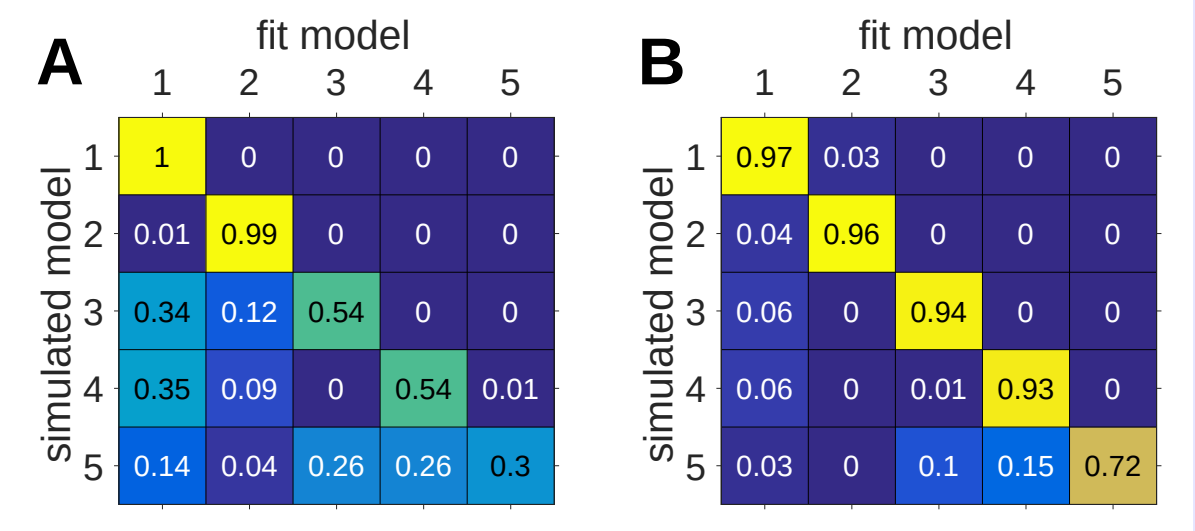


Figure 5: Confusion matrices in the bandits task showing the effect of prior parameter distributions on model recovery. (**A**) When there are relatively large amounts of noise in the models (possibility of small values for $\beta$ and $\beta_c$) models 3-5 are hard to distinguish from one another. (**B**) When there is less noise in the models (i.e. minimum value of $\beta$ and $\beta_c$ is 1) the models are much easier to identify.

# 7 Run the experiment and analyze the actual data

Once all the previous steps have been completed, you can finally move on to modeling your empirical data. The first step to complete is of course to analyze the data *without* the model, in the same way that we did for model simulations in Section 3. This model-independent analysis is extremely important: you designed the experiment to test specific hypotheses, and constructed models to reflect them. Simulations showed expected patterns of behaviors given those hypotheses. If the model-independent analyses do not show evidence of the expected results, there is almost no point in fitting the model. Instead, you should go back to the beginning, either re-thinking the computational models if the analyses show interesting patterns of behavior, or re-thinking the experimental design or even the scientific question you are trying to answer. **In our experience, if there is no model-independent evidence that the processes of interest are engaged, then a model-based analysis is unlikely to uncover evidence for the processes too.**

If, however, the behavioral results are promising, the next step is to fit the models developed previously and perform model comparison. After this step, you should check

that the parameter range obtained with the fitting is within a range where parameter and model recovery were good. If the range is outside what you explored with simulations, you should go back over the parameter and model recovery steps to match the empirical parameter range, and thus ensure that the model fitting and model comparison procedures lead to interpretable results.

An important point to remember is that human behavior is always messier than the model, and it is unlikely that the class of models you explored actually contains the 'real' model that captures human behavior. At this point, you should consider looping back to Steps 2-5 to improve models, guided by in depth model-independent analysis of the data.

For example, you may consider modeling 'unimportant parameters' - representing mechanisms that are of no interest to your scientific question, but may still affect your measures. Modeling these unimportant parameters usually captures variance in the behavior that would otherwise be attributed to noise, and as such, makes for a better estimation of 'important' parameters. For example, capturing pre-existing biases (e.g. a preference for left/right choices) in a decision or learning task provides better estimation of the inverse temperature, by avoiding attributing systematic biases to noise, which then affords better estimation of other parameters like the learning rate - this is evident in Figure 6.

**Example: Improving parameter recovery by modeling unimportant parameters**

To illustrate the effect that 'unimportant' parameters (parameters that represent mechanisms that are of no interest to your scientific question, but may still affect your measures) can have on fitting results, we model the effect of a side bias on parameter recovery in Model 3. In particular, we assume that, in addition to choosing based on learned value, the model also had a side bias, $B$, that effectively changes the value of the left bandit. That is, in the two-bandit case, the choice probabilities are given by

$$p_t^{left} = \frac{1}{1 + \exp\left(\beta(Q_t^{right} - Q_t^{left} - B)\right)} \tag{11}$$

We then simulated behavior with this model for a range of parameter values and fit the model with the original version of model 3, without the bias, and the modified version of model 3, with the bias. Unlike our previous simulations we used a version of the bandits task in which the mean rewards could change over time - specifically switching between $\mu = \{0.2, 0.8\}$ and $\mu = \{0.8, 0.2\}$ every 50 trials.

As can be seen below, including the 'unimportant' bias in the fit greatly improves the extent to which both the learning rate, $\alpha$, and softmax parameter, $\beta$, can be recovered.

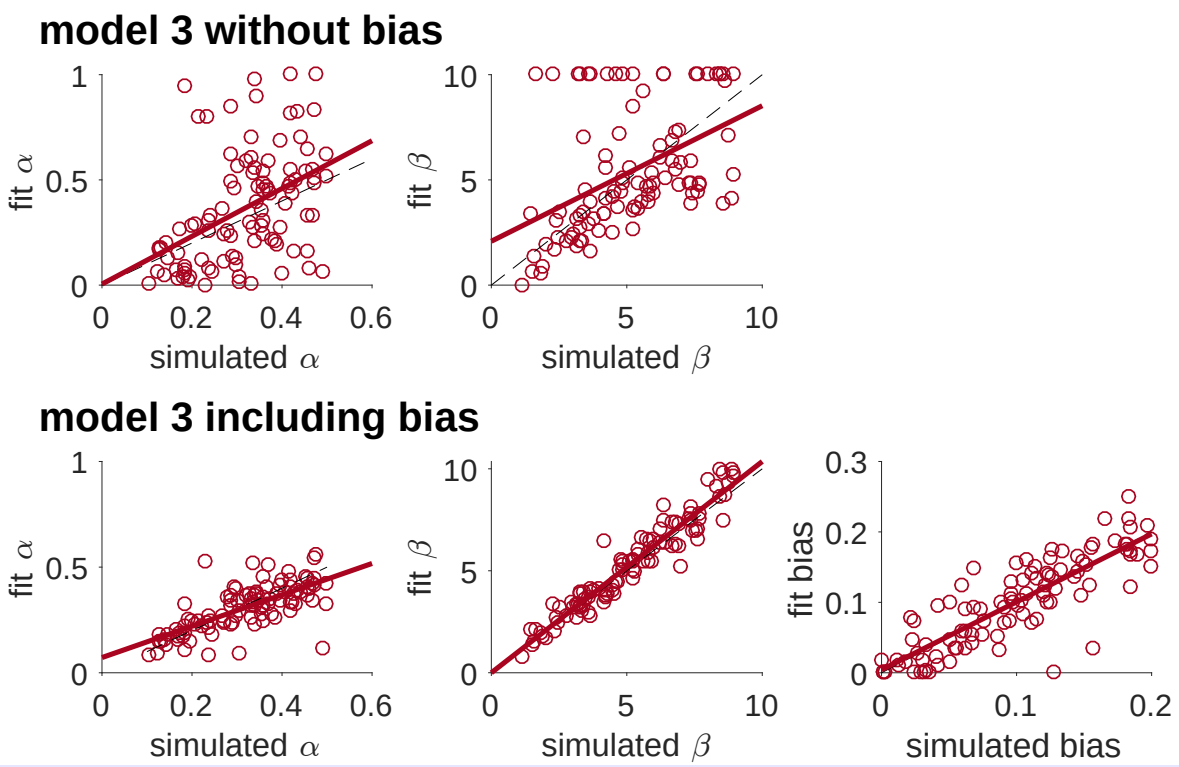


Figure 6: Modeling unimportant parameters provides better estimation of important parameters. The top row shows parameter recovery of the model without the bias term. The bottom row shows much more accurate parameter recovery, for all parameters, when the bias parameter is included in the model fits.

# 8 Validate (at least) the winning model

All the previous steps measure a *relative* goodness of fit. Does model A fit better than model B? However, before interpreting any results from a model, it is essential to ensure that the model actually usefully captures the data in an *absolute* sense. This step is called model validation, and should never be skipped - it is possible to fit a model, get high fit measures, and nevertheless completely miss the essence of the behavior.

One method for model validation is computing the average trial likelihood as an absolute measure of fit. While this measure has some nice properties - for example the best possible value is 1 when the model predicts behavior perfectly - it offers limited value when choices are actually stochastic (which may be the case in many situations (Drugowitsch, Wyart, Devauchelle, & Koechlin, 2016)) or the environment is complex. In these cases, the best possible likelihood per trial is less than 1, but it is unknown what the best possible likelihood per trial could be. For this reason, while the likelihood per trial can be a useful tool for model validation (Leong, Radulescu, Daniel, DeWoskin, & Niv, 2017), interpreting it as an absolute measure of model fit is of limited value.

A better method to validate a model is to simulate it with the fit parameter values (Palminteri et al., 2017; Nassar & Frank, 2016; Navarro, 2018). You should then analyze

the simulated data in the same way you analyzed the empirical data, to verify that all important behavioral effects are qualitatively and quantitatively captured by the simulations with the fit parameters. For example, if you observe a qualitative difference between two conditions empirically, the model should reproduce it. Likewise if a learning curve reaches a quantitative asymptote of $0.7$, simulations shouldn't reach a vastly different one.

Some researchers analyze the posterior prediction of the model conditioned on the past history, instead of simulated data. In our previous notation they evaluate the likelihood of choice $c_t$ given past data, $d_{1:t-1}$, where the past data includes choices made by the subject, *not* choices made by the model, $p(c_t|d_{1:t-1}, s_t, \boldsymbol{\theta}_m, m)$. In some cases, this approach leads to very similar results to simulations, because simulations sample choices based on a very similar probability, where the past data, $d_{1:t-1}$, include choices made by the *model*. However, it can also be dramatically different if the path of actions sampled by the participant is widely different from the paths likely to be selected by the model (leading to very different past histories).

Palminteri and colleagues (Palminteri et al., 2017) offer a striking example of this effect, where Model A fits better than Model B by any quantitative measure of model comparison, but is completely unable to capture the essence of the behavior. In their example, data is generated with a reinforcement learning agent (which takes the place of the subject) on a reversal learning task (where a choice that was previously good becomes bad, and reciprocally). These data are then fit either with either a win-stay lose-shift model (model B), or a simplistic choice kernel model which assumes that previous choices tend to be repeated (model A). Because of the autocorrelation in the choices made by the reinforcement learning agent, model A, which tends to repeat previous actions, fits better than model B, whose win-stay-lose-shift choices only depend on the action and outcome from the last trial. However, model A is completely insensitive to reward, and thus is unable to generate a reversal behavior when it is simulated with the fit model parameters. This example is an extreme case of the importance of choice kernels: if the policy generating choices is off, simulating the data will reveal this better than model comparison will, because the latter uses the subjects' choice history.

If your validation step fails, you should go back to the drawing board! **Be careful interpreting results from a model that is not well validated!** Of course, exactly what it means for a model to 'fail' the validation step is not well defined - no model is perfect, and there is no rule of thumb to tell us when a model is *good enough*. The most important aspect of validation is for you (and your readers) to be aware of its limitations, and in which ways they may influence any downstream results.

**Example: Model validation where the fit model performs too well**

Most examples of model validation involve a case where a model that fits well performs poorly on the task in simulation. For example, in the Palminteri et al example, the choice kernel model cannot perform the task at all because its behavior is completely independent of reward. Here we offer a slightly different example of failed model validation in which the model performs *better* in simulation than it does when yoked to simulated subject behavior. Moreover, this model appears to fit data gener-

ated from a different model better than it fits data generated from itself!

In this example we imagine a deterministic stimulus-action learning task in which subjects are presented with one of three stimuli ($s_1$, $s_2$, and $s_3$) which instruct the subject which of three actions ($a_1$, $a_2$, and $a_3$) will be rewarded when chosen. The first two stimuli $s_1$ and $s_2$ imply that $a_1$ is the correct action. The third stimulus, $s_3$, implies that the $a_3$ is the correct action on this trial. The second action, $a_2$, is always wrong, regardless of the stimulus presented, and is never rewarded.

The two models we consider are both reinforcement learning agents. The first, a 'blind' agent, does not see the stimulus at all and learns only about the value of the three different actions, i.e. $Q(a_i)$, regardless of the stimulus. The second, a 'state-based' agent, observes the stimulus and learns a value for each action that can be different for each stimulus, i.e. $Q(a_i, s_i)$. Because the blind agent ignores the stimulus, the best it can possibly do is to choose action 1 on every trial and be correct 2/3 of the time. Conversely, because the state-based agent takes the stimulus into account, in principle it can learn to perform perfectly on the task.

Learning in both models occurs via a Rescorla-Wagner rule with different learning rates for positive and negative prediction errors. Thus for the blind agent, the values update according to

$$Q(a_t) \leftarrow \begin{cases} Q(a_t) + \alpha_P(r_t - Q(a_t)) & \text{if } (r_t - Q(a_t)) > 0 \\ Q(a_t) + \alpha_N(r_t - Q(a_t)) & \text{if } (r_t - Q(a_t)) < 0 \end{cases} \tag{12}$$

while for the state-based agent, values update according to

$$Q(a_t, s_t) \leftarrow \begin{cases} Q(a_t, s_t) + \alpha_P(r_t - Q(a_t, s_t)) & \text{if } (r_t - Q(a_t, s_t)) > 0 \\ Q(a_t, s_t) + \alpha_N(r_t - Q(a_t, s_t)) & \text{if } (r_t - Q(a_t, s_t)) < 0 \end{cases} \tag{13}$$

In both models, these values guide decisions via softmax decision rule with inverse temperature parameter, $\beta$.

We begin by simulating two different agents: one using the blind algorithm and the other using the state-based approach. Parameters in the models are set such that the learning curves for the two agents are approximately equal (Figure 7A, blind model: $\alpha_P = 0.5$, $\alpha_N = 0$, $\beta = 6.5$ state-based model: $\alpha_P = 0.65$, $\alpha_N = 0$, $\beta = 2$). In both cases the agents start from an accuracy of 1/3 and asymptote at an accuracy of around 2/3 — the blind agent because this is the best it can do, the state-based agent because the softmax parameter is relatively small and hence performance is limited by noise.

Next we consider how the state-based model fits behavior from these two different agents. In Figure 7B, we plot the average likelihood with which the state-based model predicts the actual choices of the blind and state-based agents, i.e. the average $p(c_t|d_{1:t-1}, \boldsymbol{\theta}_m, m = \text{state-based})$. As is clear from this figure, the state-based model predicts choices from the blind agent with *higher* likelihood than choices from the state-based agent! While counter intuitive, this result does not imply that the state-based model is unable to fit its own behavior. Instead, this result reflects the difference

in noise (softmax parameters) between the two subjects. The blind RL subject has a high $\beta$, implying less noise, allowing the state-based model to fit it quite well. Conversely, the state-based RL subject has a low $\beta$, implying more noise, meaning that the behavior is harder to predict even when it is fit with the correct model.

That the state-based model fits state-based behavior better than it fits blind behavior is illustrated in Figure 7C. Here we plot the simulated learning curves of the state-based model using the parameter values that were fit to either the state-based agent or the blind agent. While the fit to the state-based agent generates a learning curve quite similar to that of the subject (compare blue lines in panels A and C), the state-based fit to the blind agent performs too well (compare yellow lines in panels A and C). Thus the model validation step provides support for the state-based model when it is the correct model of behavior, but rules out the state-based model, when the generating model was different.

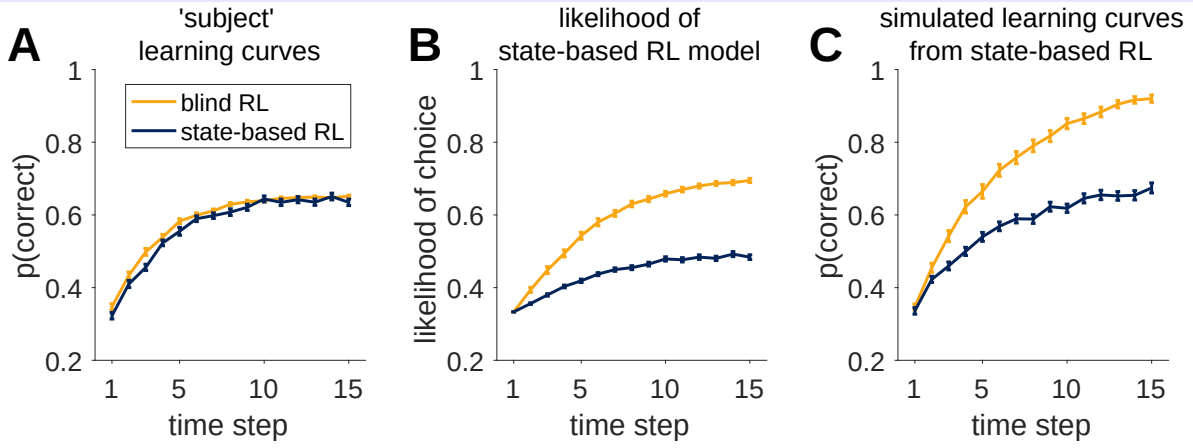


Figure 7: An example of successful and unsuccessful model validation. (**A**) Behavior is simulated by one of two reinforcement learning models (a blind agent and a state-based agent) performing the same learning task. Generative parameters of the two models were set so that the learning curves of the models were approximately equal. (**B**) Likelihood per trial seems to indicate a worse fit for the state-based-simulated data than the blind-simulated data. (**C**) However, validation by model simulations with fit parameters show that the state-based model captures the data from the state-based agent, but not from the the blind agent.

# 9 Estimate latent variables

One particularly powerful application of model-based analysis of behavior involves estimating the *latent* variables in the model. Latent variables are the hidden components of the algorithms underlying the behavior that are not directly observable from behavior itself. These latent variables shed light on the internal workings of the model and, if we are to take the model seriously, should have some representation in the subjects' mind and brain (Cohen et al., 2017; O'doherty et al., 2007).

Extracting latent variables from the model is as simple as simulating the model and recording how the latent variables evolve over time. The parameters of the simulation

should be the fit parameters for each subject. In most cases, it is useful to yoke the choices of the model to the choices the participants actually made, thus the latent variables evolve according to the experience participants actually had. This is especially true if choices can influence what participants see in the future.

Once estimated, the latent variables can be used in much the same way as any other observable variable in the analysis of data. Perhaps the most powerful application comes when combined with physiological data such as pupil dilation, EEG, and fMRI. The simplest of these approaches uses linear regression to test whether physiological variables correlate with the latent variables of interest. Such an approach has lead to a number of insights into the neural mechanisms underlying behavior (Nassar et al., 2012; Daw et al., 2011; Donoso et al., 2014; A. G. Collins & Frank, 2018; Fischer & Ullsperger, 2013), although, as with any modeling exercise, latent variable analysis should be done with care (Wilson & Niv, 2015).

# 10 Reporting model-based analyses

Congratulations! You have developed, simulated, and fit your model, and maybe several other competing models to your data. You have estimated parameters, computed model comparison scores, and validated whether your model can generate realistic looking behavior. It's time to start writing! But what exactly should you report in your paper? And how should you report it?

## 10.1 Model selection

In many modeling papers, a key conclusion from the work is that one model fits the data better than other competing models. To make this point convincingly, we recommend including the following things in your paper, either as main results or in the supplementary material.

**Model recovery analysis: Confusion matrix.** Before anyone should believe your model comparison results, you need to demonstrate the ability of your analysis/experiment to distinguish between models under ideal conditions of simulated data. The best way to visualize these results is with a confusion matrix as outlined in section **??**. If the model comparison result is central to your paper, we recommend including the confusion matrix as a Figure in the main text. If model comparison is less important we recommend including it in the supplementary materials.

**Number of subjects best fit by each model** The simplest way to visualize how well the winning model fits the data is with a histogram showing the number of subjects best fit by each model. Obviously if all subjects are best fit with one model the story is simple. The more likely scenario is that some subjects will be best fit by other models. Such a result is important to acknowledge in the paper as it may reflect the use of different strategies by different people or that the 'correct' model lies somewhere in between the models you have considered.

**Group level statistics: Exceedance probabilities.** A more sophisticated way to report model comparison results is by computing the probability that a single model best describes all the data. This is clearly an assumption whose merits should be discussed in your paper. In cases where it is valid, the method of (Rigoux et al., 2014) computes these 'Exceedance Probabilities' - the probability that each model generated all the data. These probabilities can also be reported in histogram or table form.

**Model-independent measures of simulated data.** The cleanest way to demonstrate one model's superiority is if that model can account for qualitative patterns in the data that are not captured by other models (see section 8).

## 10.2 Parameter fits

Many modeling papers involve fitting parameters to behavioral data. In some cases this is the main point of the paper, for example to show that parameter values differ between groups or treatments, in other cases parameter fitting is secondary to model comparison. In all cases we recommend reporting the fit parameter values in as transparent a way possible (i.e. more than just the means and standard error).

**Report distributions of parameter values.** The simplest way to report parameter fits is to plot a distribution of all fit parameter values (for example in the form of a histogram, a cloud of points, or a cumulative distribution function [cite examples for each]). This gives a great sense of the variability in each parameter across the population and can also illustrate problems with fitting. For example, if a large number of fit parameters are clustered around the upper and lower bounds this may indicate a problem with the model.

**Plot pairwise correlations between fit parameter values.** A deeper understanding of the relationships *between* fit parameters can be obtained by making scatter plots of the pairwise correlations between parameters. As with histograms of individual parameters, this approach gives a sense of the distribution of parameters, and can provide evidence of problems with the model - for example, if two parameters trade off against one another it is a sign that these parameters may be unidentifiable in this experiment.

**Report parameter recovery.** Finally, all parameter fit analyses should sit on the shoulders of a comprehensive parameter recovery analysis with simulated data. If parameters cannot be recovered in the ideal case of simulated data, there is little they can tell us about real behavior.

## 10.3 Share your data and code!

The most direct way to communicate your results is to share the data and code. This approach encourages transparency and ensures that others can see *exactly* what you did.

Sharing data and code also allows others to extend your analyses easily, by applying it to their own data or adding new models into the mix.

Ideally the data you share should be the raw data for the experiment, with minimal or no preprocessing (apart from the removal of identifying information). The code you share should reproduce all steps in your analysis, including any preprocessing/outlier exclusion you may have performed as well as generating all of the main and supplementary figures in the paper. In a perfect world, both data and code would be shared publicly on sites such as GitHub, DataVerse etc ... However, this is not always possible, for example if the data comes from collaborators who do not agree to data sharing, or if further analyses are planned using the same data set. In this case, we recommend having a clean set of 'shareable' code (and hopefully data too) that can be sent via email upon request.

### 10.4 Should you always report all of your modeling results?

Finally, if you are using an established model, it can be tempting to skip many of the steps outlined above and report only the most exciting results. This temptation can be even greater, if you are using code developed by someone else that, perhaps, you do not fully understand. In our opinion, taking shortcuts like this is wrong. For one thing, your experiment or population may be different and the model may perform differently in this regime. For another, quite often 'established' models (in the sense that they have been published before), have not been validated in a systematic way. More generally, as with any research technique, when using computational modeling you need to demonstrate that you are applying the method correctly, and the steps we outline here can help. In conclusion, even if developing the model is not the central point of your paper, you should report all of your modeling results.

## 11 What now?

A modeler's work is never done. To paraphrase George Box, there are no correct models, there are only useful models (Box, 1979). To make your model more useful, there are a number of next steps to consider to test whether your model really does describe a process in the mind.

**Improve the model to account for discrepancies with your existing data set.** Model fits are never perfect and, even in the best cases, there are often small discrepancies with actual data. The simplest next step is to try to address these discrepancies by improving the model, either by including additional factors (such as side bias, or lapse rates) or by devising new models entirely.

**Use your model to make predictions.** The best models don't just explain data in one experiment, they *predict* data in completely new situations. If your model does not easily generalize to new situations, try to understand why that is and how it could be adjusted to be more general. If your model does generalize, test its predictions against new data - either data you collect yourself from a new experiment, or data from other studies that (hopefully) have been shared online.

# Epilogue

Our goal for this paper was to offer practical advice, for beginners as well as seasoned researchers, for the computational modeling of behavioral data. To this end we offered guidance on how to generate models, simulate models, fit models, compare models, validate models, and extract latent variables from models to compare with physiological data. We have talked about how to avoid common pitfalls and misinterpretations that can arise with computational modeling, and lingered, quite deliberately, on the importance of good experimental design. Many of these lessons were lessons we learned the hard way, by actually making these mistakes for ourselves over a combined 20+ years in the field. By following these steps, we hope that you will avoid some of the errors that slowed our own research and that the overall quality of computational modeling in behavioral science will improve.

# Acknowledgements

We are grateful to all our lab members who provided feedback on this paper, in particular Waitsang Keung, Sarah Master, Sam McDougle, and William Ryan. We also gratefully acknowledge the contribution of many others in our previous labs and collaborations, with whom we learned many of the techniques, tips and tricks presented here. This work was supported by NIA Grant R56 AG061888 to RCW and NSF Grant 1640885 to AGEC.

# References

Abbott, J. T., Austerweil, J. L., Griffiths, T. L., Abbott, J. T., Austerweil, J. L., & Griffiths, T. L. (2015). Optimal Foraging Random Walks on Semantic Networks Can Resemble Optimal Foraging. *Psychological Review*, *122*, 558–559. doi: 10.1037/a0038693

Akaike, H. (1974). A new look at the statistical model identification. *IEEE transactions on automatic control*, *19*(6), 716–723.

Box, G. E. (1979). Robustness in the strategy of scientific model building. In *Robustness in statistics* (pp. 201–236). Elsevier.

Breiman, L., et al. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, *16*(3), 199–231.

Broomell, S. B., & Bhatia, S. (2014). Parameter recovery for decision modeling using choice data. *Decision*, *1*(4), 252.

Busemeyer, J. R., & Diederich, A. (2010). *Cognitive modeling*. Sage.

Byrd, R. H., Gilbert, J. C., & Nocedal, J. (2000). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, *89*(1), 149–185.

Cavanagh, J. F., Wiecki, T. V., Kochar, A., & Frank, M. J. (2014). Eye tracking and pupillometry are indicators of dissociable latent decision processes. *Journal of Experimental Psychology: General*, *143*(4), 1476.

Cohen, J. D., Daw, N., Engelhardt, B., Hasson, U., Li, K., Niv, Y., ... others (2017). Computational approaches to fmri analysis. *Nature neuroscience*, *20*(3), 304.

Cohen, J. D., Dunbar, K., & McClelland, J. L. (1990). On the control of automatic processes: a parallel distributed processing account of the stroop effect. *Psychological review*, *97*(3), 332.

Collins, A., & Frank, M. (2013). Cognitive control over learning: Creating, clustering, and generalizing task-set structure. *Psychological review*.

Collins, A., & Koechlin, E. (2012, mar). Reasoning, Learning, and Creativity: Frontal Lobe Function and Human Decision-Making. *PLoS Biology*, *10*(3), e1001293. Retrieved from `http://dx.plos.org/10.1371/journal.pbio.1001293` doi: 10.1371/journal.pbio.1001293

Collins, A. G., Brown, J. K., Gold, J. M., Waltz, J. A., & Frank, M. J. (2014). Working memory contributions to reinforcement learning impairments in schizophrenia. *Journal of Neuroscience*, *34*(41), 13747–13756.

Collins, A. G., & Frank, M. J. (2012). How much of reinforcement learning is working memory, not reinforcement learning? a behavioral, computational, and neurogenetic analysis. *European Journal of Neuroscience*, *35*(7), 1024–1035.

Collins, A. G., & Frank, M. J. (2014). Opponent actor learning (opal): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. *Psychological review*, *121*(3), 337.

Collins, A. G., & Frank, M. J. (2018). Within-and across-trial dynamics of human eeg reveal cooperative interplay between reinforcement learning and working memory. *Proceedings of the National Academy of Sciences*, 201720963.

Courville, A. C., & Daw, N. D. (2008). The rat as particle filter. In *Advances in neural information processing systems* (pp. 369–376).

Daw, N. D. (2011). Trial-by-trial data analysis using computational models. *Decision making, affect, and learning: Attention and performance XXIII*, *23*, 3–38.

Daw, N. D., & Courville, A. C. (2007). The pigeon as particle filter. *Advances in neural information processing systems*, 1–8.

Daw, N. D., Gershman, S. J., Seymour, B., Dayan, P., & Dolan, R. J. (2011). Model-based influences on humans' choices and striatal prediction errors. *Neuron*, *69*(6), 1204–1215.

Daw, N. D., & Tobler, P. N. (2014). Value learning through reinforcement: the basics of dopamine and reinforcement learning. In *Neuroeconomics (second edition)* (pp. 283–298). Elsevier.

Donoso, M., Collins, A. G., & Koechlin, E. (2014). Foundations of human reasoning in the prefrontal cortex. *Science*, *344*(6191), 1481–1486.

Dowd, E. C., Frank, M. J., Collins, A., Gold, J. M., & Barch, D. M. (2016). Probabilistic reinforcement learning in patients with schizophrenia: Relationships to anhedonia and avolition. *Biological psychiatry: cognitive neuroscience and neuroimaging*, *1*(5), 460–473.

Drugowitsch, J., Wyart, V., Devauchelle, A.-D., & Koechlin, E. (2016). Computational precision of mental inference as critical source of human choice suboptimality. *Neuron*, *92*(6), 1398–1411.

Farashahi, S., Rowe, K., Aslami, Z., Lee, D., & Soltani, A. (2017). Feature-based learning improves adaptability without compromising precision. *Nature Communications*, *8*(1). doi: 10.1038/s41467-017-01874-w

Farrell, S., & Lewandowsky, S. (2018). *Computational modeling of cognition and behavior*. Cambridge University Press.

Fischer, A. G., & Ullsperger, M. (2013). Real and fictive outcomes are processed differently but converge on a common adaptive mechanism. *Neuron*, *79*(6), 1243–1255.

Frank, M. J., Moustafa, A. A., Haughey, H. M., Curran, T., & Hutchison, K. E. (2007). Genetic triple dissociation reveals multiple roles for dopamine in reinforcement learning. *Proceedings of the National Academy of Sciences*, *104*(41), 16311–16316.

Frank, M. J., Seeberger, L. C., & O'reilly, R. C. (2004). By carrot or by stick: cognitive reinforcement learning in parkinsonism. *Science*, *306*(5703), 1940–1943.

Geisler, W. S. (2011). Contributions of ideal observer theory to vision research. *Vision research*, *51*(7), 771–781.

Gillan, C. M., Kosinski, M., Whelan, R., Phelps, E. A., & Daw, N. D. (2016). Characterizing a psychiatric symptom dimension related to deficits in goal-directed control. *Elife*, *5*, e11305.

Heathcote, A., Brown, S. D., & Wagenmakers, E.-J. (2015). An introduction to good practices in cognitive modeling. In *An introduction to model-based cognitive neuroscience* (pp. 25–48). Springer.

Huys, Q. J. (2017, jan). Bayesian Approaches to Learning and Decision-Making. In *Computational psychiatry: Mathematical modeling of mental illness* (pp. 247–271). Academic Press. doi: 10.1016/B978-0-12-809825-7.00010-9

Kass, R. E., & Raftery, A. E. (1995). Bayes factors. *Journal of the american statistical association*, *90*(430), 773–795.

Kording, K., Blohm, G., Schrater, P., & Kay, K. (2018). Appreciating diversity of goals in computational neuroscience.

Lee, M. D., & Wagenmakers, E.-J. (2014). *Bayesian cognitive modeling: A practical course*. Cambridge university press.

Leong, Y. C., Radulescu, A., Daniel, R., DeWoskin, V., & Niv, Y. (2017). Dynamic interaction between reinforcement learning and attention in multidimensional environments. *Neuron*, *93*(2), 451–463.

Lieder, F., Griffiths, T. L., Huys, Q. J., & Goodman, N. D. (2018). Empirical evidence for resource-rational anchoring and adjustment. *Psychonomic Bulletin & Review*, *25*(2), 775–784.

Lorains, F. K., Dowling, N. A., Enticott, P. G., Bradshaw, J. L., Trueblood, J. S., & Stout, J. C. (2014). Strategic and non-strategic problem gamblers differ on decision-making under risk and ambiguity. *Addiction*, *109*(7), 1128–1137.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Montague, P. R., Dayan, P., & Sejnowski, T. J. (1996a). A framework for mesencephalic dopamine systems based on predictive hebbian learning. *Journal of neuroscience*, *16*(5), 1936–1947.

Montague, P. R., Dayan, P., & Sejnowski, T. J. (1996b, mar). A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *The Journal of*

*neuroscience : the official journal of the Society for Neuroscience*, *16*(5), 1936–47.

Moré, J. J., & Sorensen, D. C. (1983). Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, *4*(3), 553–572.

Nassar, M. R., & Frank, M. J. (2016). Taming the beast: extracting generalizable knowledge from computational models of cognition. *Current opinion in behavioral sciences*, *11*, 49–54.

Nassar, M. R., Helmers, J. C., & Frank, M. J. (2018). Chunking as a rational strategy for lossy data compression in visual working memory. *Psychological review*, *125*(4), 486.

Nassar, M. R., Rumsey, K. M., Wilson, R. C., Parikh, K., Heasly, B., & Gold, J. I. (2012). Rational regulation of learning dynamics by pupil-linked arousal systems. *Nature neuroscience*, *15*(7), 1040.

Nassar, M. R., Wilson, R. C., Heasly, B., & Gold, J. I. (2010). An approximately bayesian delta-rule model explains the dynamics of belief updating in a changing environment. *Journal of Neuroscience*, *30*(37), 12366–12378.

Navarro, D. J. (2018). Between the devil and the deep blue sea: Tensions between scientific judgement and statistical model selection. *Computational Brain & Behavior*, 1–7.

Nilsson, H., Rieskamp, J., & Wagenmakers, E.-J. (2011). Hierarchical bayesian parameter estimation for cumulative prospect theory. *Journal of Mathematical Psychology*, *55*(1), 84–93.

O'doherty, J. P., Hampton, A., & Kim, H. (2007). Model-based fmri and its application to reward learning and decision making. *Annals of the New York Academy of sciences*, *1104*(1), 35–53.

Otto, A. R., Raio, C. M., Chiang, A., Phelps, E. A., & Daw, N. D. (2013). Working-memory capacity protects model-based learning from stress. *Proceedings of the National Academy of Sciences*, *110*(52), 20941–20946.

OReilly, J. X., Schüffelgen, U., Cuell, S. F., Behrens, T. E., Mars, R. B., & Rushworth, M. F. (2013). Dissociable effects of surprise and model update in parietal and anterior cingulate cortex. *Proceedings of the National Academy of Sciences*, 201305373.

Palminteri, S., Wyart, V., & Koechlin, E. (2017). The importance of falsification in computational cognitive modeling. *Trends in cognitive sciences*, *21*(6), 425–433.

Ratcliff, R. (1978). A theory of memory retrieval. *Psychological review*, *85*(2), 59.

Rescorla, R. A., Wagner, A. R., et al. (1972). A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, *2*, 64–99.

Rigoux, L., Stephan, K. E., Friston, K. J., & Daunizeau, J. (2014). Bayesian model selection for group studiesrevisited. *Neuroimage*, *84*, 971–985.

Samejima, K., Ueda, Y., Doya, K., & Kimura, M. (2005, nov). Representation of action-specific reward values in the striatum. *Science (New York, N.Y.)*, *310*(5752), 1337–40. doi: 10.1126/science.1115270

Schwarz, G., et al. (1978). Estimating the dimension of a model. *The annals of statistics*, *6*(2), 461–464.

Somerville, L. H., Sasse, S. F., Garrad, M. C., Drysdale, A. T., Abi Akar, N., Insel, C., & Wilson, R. C. (2017). Charting the expansion of strategic exploratory behavior during adolescence. *Journal of experimental psychology: general*, *146*(2), 155.

Starns, J. J., & Ratcliff, R. (2010). The effects of aging on the speed–accuracy compromise: Boundary optimality in the diffusion model. *Psychology and aging*, *25*(2), 377.

Steyvers, M., Lee, M. D., & Wagenmakers, E.-J. (2009). A bayesian analysis of human decision-making on bandit problems. *Journal of Mathematical Psychology*, *53*(3), 168–179.

Turner, B. M., Forstmann, B. U., Wagenmakers, E.-J., Brown, S. D., Sederberg, P. B., & Steyvers, M. (2013). A bayesian framework for simultaneously modeling neural and behavioral data. *NeuroImage*, *72*, 193–206.

Vandekerckhove, J., Matzke, D., & Wagenmakers, E.-J. (2015). Model comparison and the principle. In *The oxford handbook of computational and mathematical psychology* (Vol. 300). Oxford Library of Psychology.

Wagenmakers, E.-J., & Farrell, S. (2004). Aic model selection using akaike weights. *Psychonomic bulletin & review*, *11*(1), 192–196.

Warren, C. M., Wilson, R. C., van der Wee, N. J., Giltay, E. J., van Noorden, M. S., Cohen, J. D., & Nieuwenhuis, S. (2017). The effect of atomoxetine on random and directed exploration in humans. *PloS one*, *12*(4), e0176034.

Watkins, C. J. C. H., & Dayan, P. (1992, may). Q-learning. *Machine Learning*, *8*(3-4), 279–292. Retrieved from `http://link.springer.com/10.1007/BF00992698` doi: 10.1007/BF00992698

Wilson, R. C., & Collins, A. (2019). *Code for the figures in the "ten simple rules for computational modeling of psychological data" paper.* Retrieved 2019-01-04, from `https://github.com/AnneCollins/TenSimpleRulesModeling`

Wilson, R. C., Nassar, M. R., & Gold, J. I. (2013). A mixture of delta-rules approximation to bayesian inference in change-point problems. *PLoS computational biology*, *9*(7), e1003150.

Wilson, R. C., & Niv, Y. (2012). Inferring relevance in a changing world. *Frontiers in human neuroscience*, *5*, 189.

Wilson, R. C., & Niv, Y. (2015). Is model fitting necessary for model-based fmri? *PLoS computational biology*, *11*(6), e1004237.

Wimmer, G. E., Li, J. K., Gorgolewski, K. J., & Poldrack, R. A. (2018). Reward learning over weeks versus minutes increases the neural representation of value in the human brain. *The Journal of Neuroscience*, *38*(35), 0075–18. doi: 10.1523/JNEUROSCI.0075-18.2018

Zajkowski, W. K., Kossut, M., & Wilson, R. C. (2017). A causal role for right frontopolar cortex in directed, but not random, exploration. *eLife*, *6*, e27430.

# A The theory of model fitting

Formally, the goal of model fitting is to estimate the parameters, $\boldsymbol{\theta}_m$, for each model, $m$, that best fit the behavioral data. To do this we take a Bayesian approach and aim to compute (or at least approximate) the posterior distribution over the parameters given the data, $p(\boldsymbol{\theta}_m|d_{1:T}, m)$. By Bayes' rule we can write this as

$$p(\boldsymbol{\theta}_m|d_{1:T}, m) = \frac{p(d_{1:T}|\boldsymbol{\theta}_m, m)p(\boldsymbol{\theta}_m|m)}{p(d_{1:T}|m)} \tag{14}$$

where $p(\boldsymbol{\theta}_m|m)$ is the prior on the parameters, $\boldsymbol{\theta}_m$, $p(d_{1:T}|\boldsymbol{\theta}_m, m)$ is the likelihood of the data given the parameters and the normalization constant, $p(d_{1:T}|m)$, is the probability of the data given the model (more on this below). Because the probabilities tend to be small, it is often easier to work with the log of these quantities

$$\log p(\boldsymbol{\theta}_m|d_{1:T}, m) = \underbrace{\log p(d_{1:T}|\boldsymbol{\theta}_m, m)}_{\text{log likelihood}} + \log p(\boldsymbol{\theta}_m|m) - \log p(d_{1:T}|m) \tag{15}$$

The log-likelihood often gets its own symbol, $LL = \log p(d_{1:T}|\boldsymbol{\theta}_m, m)$, and can be written

$$\log p(d_{1:T}|\boldsymbol{\theta}_m, m) = \log \left( \prod_{t=1}^{T} p(c_t|d_{1:t-1}, s_t, \boldsymbol{\theta}_m, m) \right) = \sum_{t=1}^{T} \log p(c_t|d_{1:t-1}, s_t, \boldsymbol{\theta}_m, m) \tag{16}$$

where $p(c_t|d_{1:t-1}, s_t, \boldsymbol{\theta}_m, m)$ is the probability of each individual choice given the parameters of the model, which is at the heart of the definition of each model (for example in Equations 1-7).

In a perfect world we would evaluate the log-posterior, $\log p(\boldsymbol{\theta}_m|d_{1:T}, m)$, exactly, but this can be difficult to compute and unwieldy to report. Instead, we must approximate it. This can be done using sampling approaches such as Markov Chain Monte Carlo approaches (Lee & Wagenmakers, 2014), which approximate the full posterior with a set of samples. Another approach is to report a point estimate for the parameters such as the maximum of the log-posterior (the maximum *a posteriori* (MAP) estimate), or the maximum of the log-likelihood (the maximum likelihood estimate (MLE))[2].

$$\begin{aligned} \hat{\boldsymbol{\theta}}_m^{MAP} &= \arg\max_{\boldsymbol{\theta}_m} \log p(\boldsymbol{\theta}_m|d_{1:T}, m) \\ \hat{\boldsymbol{\theta}}_m^{MLE} &= \arg\max_{\boldsymbol{\theta}_m} \log p(d_{1:T}|\boldsymbol{\theta}_m, m) \end{aligned} \tag{17}$$

Note that with a uniform prior on $\boldsymbol{\theta}_m$, these two estimates coincide.

These approaches for estimating parameter values each have different strengths and weaknesses. The MCMC approach is the most principled as, with enough samples, it gives a good approximation of the posterior distribution over each parameter value. This approach also gracefully handles small data sets and allows us to combine data from

[2]Note that the log transformation does not change the location of the maximum, so the maximum of the log-likelihood occurs at the same value of $\boldsymbol{\theta}_m$ as the maximum of the likelihood.

different subjects in a rigorous manner. Despite these advantages, the MCMC approach is more complex (especially for beginners) and can be slow to implement. On the other hand, point estimates such as the MAP and MLE parameter values are much quicker to compute and often given similar answers to the MCMC approach when the amount of data is large (which is often the case when dealing with young and healthy populations). For this reason, we focus our discussion on the point estimate approaches, focusing in particular on maximum likelihood estimation.

# B The theory of model comparison

In model comparison, our goal is to figure out which model of a set of possible models is most likely to have generated the data. To do this we compute (or at least try to estimate) the probability that model $m$ generated the data, $p(m|d_{1:T})$. Note that this is the normalization constant from equation 14. As with parameter recovery, this probability is difficult to compute directly and so we turn to Bayes' rule and write

$$\begin{aligned} p(m|d_{1:T}) &\propto p(d_{1:T}|m)p(m) \\ &= \int d\boldsymbol{\theta}_m p(d_{1:T}|\boldsymbol{\theta}_m, m)p(\boldsymbol{\theta}_m|m)p(m) \end{aligned} \tag{18}$$

where $p(m)$ is the prior probability that model $m$ is the correct model and $p(d_{1:T}|m)$ is the likelihood of the data given the model. In most cases, $p(m)$ is assumed to be constant and so we can focus entirely on the likelihood, $p(d_{1:T}|m)$. As before it is easier to handle the log of this quantity which is known as the Bayesian evidence, $E_m$, (Kass & Raftery, 1995) for model $m$. More explicitly

$$E_m = \log p(d_{1:T}|m) = \log \int d\boldsymbol{\theta}_m p(d_{1:T}|\boldsymbol{\theta}_m, m)p(\boldsymbol{\theta}_m|m) \tag{19}$$

If we can compute $E_m$ for each model, then the model with the largest evidence is most likely to have generated the data.

Note that by integrating over the parameter space, the Bayesian evidence implicitly penalizes free parameters. This is because, the more free parameters, the larger the size of the space over which we integrate and, consequently, the smaller $p(\boldsymbol{\theta}_m|m)$ is for any given parameter setting. Thus, unless the model predicts the data well for all parameter settings, it pays a price for each additional free parameter. This idea, that simpler models should be favored over more complex models if they both explain the data equally well, is known as Occam's razor (see Chapter 28 in (MacKay, 2003)).

Unfortunately, because it involves computing an integral over all possible parameter settings, computing $E_m$ exactly is usually impossible. There are several methods for approximating the integral based on either replacing it with a sum over a subset of points or replacing it with an approximation around either the MAP or MLE estimates of the parameters. The latter approach is the most common and three particular forms are used: the Bayes Information Criterion (BIC) (Schwarz et al., 1978), Akaike information criterion (AIC) (Akaike, 1974) and the Laplace approximation (Kass & Raftery, 1995). Here we will

focus on BIC which is an estimate based around the maximum likelihood estimate of the parameters, $\hat{\boldsymbol{\theta}}_m^{MLE}$,

$$BIC = -2\log\hat{\mathcal{L}} + k_m\log(T) \approx -2\log E_m \tag{20}$$

where $k_m$ is the number of parameters in model $m$ and $\hat{\mathcal{L}}$ is the value of the log-likelihood at $\hat{\boldsymbol{\theta}}_m^{MLE}$.

Finally, we have found it useful to report the results of model comparison in terms of the likelihood-per-trial $LPT$, which can be thought of as the 'average' probability with which the model predicts each choice.

$$LPT = \exp\left(\frac{E_m}{T}\right) \tag{21}$$

# C Working memory model used for local minima example

The model and experimental designs used in figure 3 are a simplified version of those in (A. G. Collins & Frank, 2012). In short, the experiment attempts to parse out working memory contributions to reinforcement learning by having participants and agents learn stimulus-action contingencies from deterministic feedback, with a different number of stimuli $ns$ being learned in parallel in different blocks. This manipulation targets WM load and isolates WM contributions; see (A. G. Collins & Frank, 2012) for details.

The simplified model assumes a mixture of a classic RL component (with parameters $\alpha$ and $\beta$), and of a working memory component with perfect one-shot learning. The mixture is controled by parameter $\rho$, capturing the prior willingness to use working memory vs. RL, and capacity parameter $K$, which scales the mixture weight in proportion of the proportion of stimuli that may be held in working memory: $\min(1, \frac{K}{ns})$. The original model assumes additional dynamics for the working memory policy and working memory vs. RL weights that render the model more identifiable (A. G. Collins & Frank, 2012).