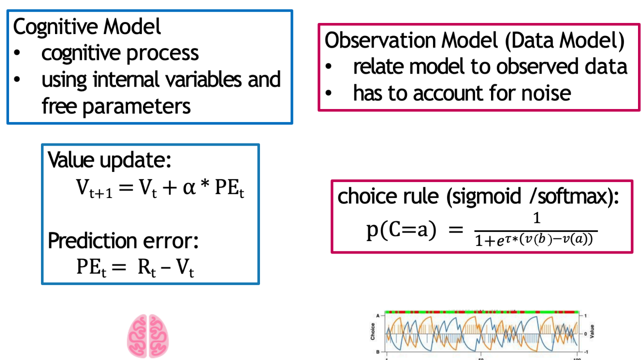
# Implementing the Rescorla-Wagner model in Stan

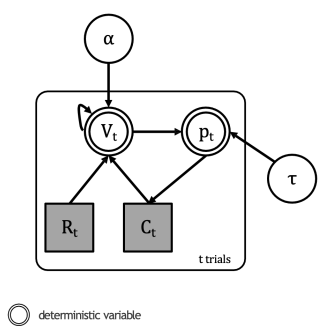
H

We now have the underlying structure of our computational model, consisting of the Rescorla-Wagner model and the softmax choice rule:




The RW model represents the latent cognitive processes driving learning and valuation, whilst the softmax translates this behaviour into observed choices

Let's now implement this practically in Stan. Again, we we have done in previous cases, let's first visually depict our model in the graphical format:




$$\alpha \sim Uniform(0,1)$$
$$\tau \sim Uniform(0,3)$$
$$p_t(C=A) = \frac{1}{1+e^{\tau(V_t(B)-V_t(A))}}$$
$$V^c_{t+1} = V^C_t + \alpha(R_t - V^C_t)$$

Our RL model written as a directed acrylic graph

Within our model we have the following:

**Parameters (white circles):**

$\alpha$ (learning rate): A continuous parameter bounded between 0 and 1

- Represented by: $\alpha \sim Uniform(0, 1)$
- This is unobserved - we'll need to estimate it from the data

$\tau$ (temperature): A continuous parameter bounded between 0 and $x$, where $x < \infty$ (in this example 3)

- Represented by: $\tau \sim Uniform(0, 3)$
- Also unobserved and needs to be estimated

**Observed variables (grey squares):**

$R_t$ (reward): A discrete variable representing the outcome on each trial

$C_t$ (choice): A discrete variable representing which option was selected on each trial

**Deterministic variables (double circles):**

$V_t$ (value): A continuous variable representing the expected value of each option

$p_t$ (probability): A continuous variable representing the probability of selecting each option

> **!** Bounds for the learning rate and inverse temperature
>
> Whilst the learning rate is always bounded between 0 and 1, the inverse temperature parameter is not necessarily bounded. The theoretical range of this parameter is $[0, +\infty]$, yet in practice, it is suggested to introduce an upper limit to avoid unstable model estimation. Some have suggested a reasonable range to be $[0, 10]$[1].

## Constructing the Rescorla-Wagner model for a single subject

**Let's firstly fit this model to a single participant.** The data (`_data/rl_sp_ss.RData`) is structured as follows:

[1] Zhang, L., Lengersdorff, L., Mikus, N., Gläscher, J., & Lamm, C. (2020). Using reinforcement learning models in social neuroscience: frameworks, pitfalls and suggestions of best practices. Social Cognitive and Affective Neuroscience, 15(6), 695-707.

```
> head(rl_ss)
     [,1] [,2]
[1,]    2   -1
[2,]    1    1
[3,]    1    1
[4,]    1    1
...
[97,]    1   -1
[98,]    2   -1
[99,]    1    1
[100,]   1    1
```

We just have 100 trials of data consisting of two unnamed columns corresponding to choice (1 or 2) and reward (-1 or +1).

A simple implementation of a RL model for this single participant is located within the following script (`_scripts/my_1st_rw.stan`).

Let's break it down block-by-block:

In the `data` block we declare what data we'll feed into our model:

```
data {
    int<lower=1> nTrials;
    int<lower=1,upper=2> choice[nTrials];
    int<lower=-1,upper=1> reward[nTrials];
}
```

- `nTrials`: An integer declaring the number of trials. We bound it to be at least 1 since we need data to fit the model.
- `choice`: An array of integers with length `nTrials`. Each element must be either 1 or 2, representing which option was chosen.
- `reward`: An array of integers with length `nTrials`. Each element must be either -1 or 1, representing losses and wins respectively.

Then in the `parameters` block we declare the parameters we want to estimate:

```
parameters {
    real<lower=0,upper=1> alpha; // learning rate
    real<lower=0,upper=20>tau; // softmax inv.temp.
}
```

- `alpha`: The learning rate, bounded between 0 and 1

- `tau`: The inverse temperature parameter, bounded between 0 and 20

Finally, the `model` block implements the actual Rescorla-Wagner model:

```
model {
    real pe;
    vector[2] v;
    vector[2] p;

    for (t in 1:nTrials) {
        p = softmax( tau * v); // action probability computed via softmax
        choice[t] ~ categorical(p);

        pe = reward[t] - v[choice[t]]; // compute pe for chosen value only
        v[choice[t]] = v[choice[t]] + alpha * pe; // update chosen V
    }

}
```

First, we declare the local variables:

- `pe`: A scalar to store the prediction error
- `v`: A vector of length 2 to store value estimates for both options
- `p`: A vector of length 2 to store choice probabilities

The main loop then iterates through trials and implements our RL model:

- `p = softmax(tau * v)`: Converts values to probabilities using the softmax function, multiplying by `tau` to implement the temperature scaling

- `choice[t] ~ categorical(p)`: Tells Stan that choices are distributed according to our computed probabilities

- `pe = reward[t] - v[choice[t]]`: Computes the prediction error for the chosen option

- `v[choice[t]] = v[choice[t]] + alpha * pe`: Updates the value estimate using the Rescorla-Wagner equation

Layering complexity

Most RL models build upon the RW equation, but retain it at their core. Take for example the case where we aim to incorporate separate learning rates, for negative and positive outcomes. In our model, we would make the following changes:

1. Rename the existing learning rate parameter to `alpha_pos` and introduce a new learning rate for negative outcomes `alpha_neg`.

```
parameters {
    real<lower=0,upper=1> alpha_pos; // rename existing learning rate
    real<lower=0,upper=1> alpha_neg; // add negative-trial learning rate
    real<lower=0,upper=20>tau;
}
```

2. And then in the `model` block, create a for loop to calculate `v[choice[t]]` with the appropriate learning rate depending on the outcome:

```
model {
    real pe;
    vector[2] v;
    vector[2] p;

    for (t in 1:nTrials) {
        p = softmax(tau * v);
        choice[t] ~ categorical(p);

        pe = reward[t] - v[choice[t]];

        if (reward[t] > 0)
            v[choice[t]] = v[choice[t]] + alpha_pos * pe; // update with positive learn
        else
            v[choice[t]] = v[choice[t]] + alpha_neg * pe; // update with negative learn
    }
}
```

So now that we have our Stan model completed, we can run it in `R`.

The script to do so is (`_scripts/reinforcement_learning_single_parm_main.R`).

Within this script, we need to firstly load the data and perform some pre-processing (renaming columns) before saving it as a data list:

```
# load the data and assign
load('_data/rl_sp_ss.RData')
sz <- dim(rl_ss)
nTrials <- sz[1]
```

```
dataList <- list(nTrials=nTrials,
                 choice=rl_ss[,1],
                 reward=rl_ss[,2])
```

Looking at the data now, it makes more sense than before:

```
> str(dataList)
List of 3
 $ nTrials: int 100
 $ choice : num [1:100] 2 1 1 1 2 1 1 1 1 1 ...
 $ reward : num [1:100] -1 1 1 1 -1 1 1 -1 -1 1 ...
```

We just need to declare which model we want to use:

```
modelFile <- '_scripts/reinforcement_learning_sp_ss_model.stan`
```

with the rest of the script setting the various components of our sampling approach as encountered in earlier workshops.

The model that we are using in the script (`_scripts/reinforcement_learning_sp_ss_model.stan`) is slightly different to the one above. In this version we:

Provide initial choice values of 0 in a `transformed data` block:

```
transformed data {
  vector[2] initV;  // initial values for V
  initV = rep_vector(0.0, 2);
}
```

Keep track of values at every trial:

```
model {
  vector[2] v[nTrials+1];   // Creates an array of nTrials+1 vectors, each vector has 2 e
  ...
  v[1] = initV;             // Sets initial values
  for (t in 1:nTrials) {
    ...
    v[t+1] = v[t];         // Copies current trial's values to next trial
    v[t+1, choice[t]] = v[t, choice[t]] + lr * pe[t];  // Updates chosen value for next t
  }
}
```

There are smaller changes (i.e., the `categorical_logit`), but these are the main two of note.

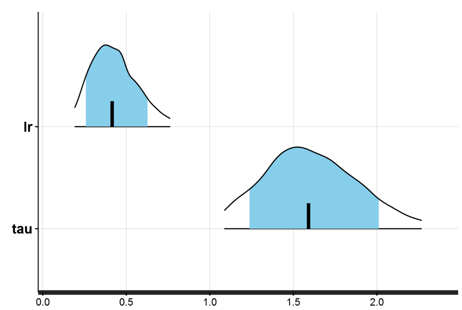
Running the model

The `R` script is enveloped in a wrapper function, which runs the model either for an individual or for multiple subjects. You can simply run it for individuals as it is, or simply by highlighting and running the three components for the individual subject model.

Exercise 7

**1. Run the `R` script specifically for the model `reinforcement_learning_sp_ss_model.stan`. Examine the traceplot and posterior density plot for the learning rate and inverse temperature.**

After running the model, you should get the following posterior density distribution for the learning model and inverse temperature parameters:




Posterior density distributions for the learning rate and inverse temperature parameters (single subject)

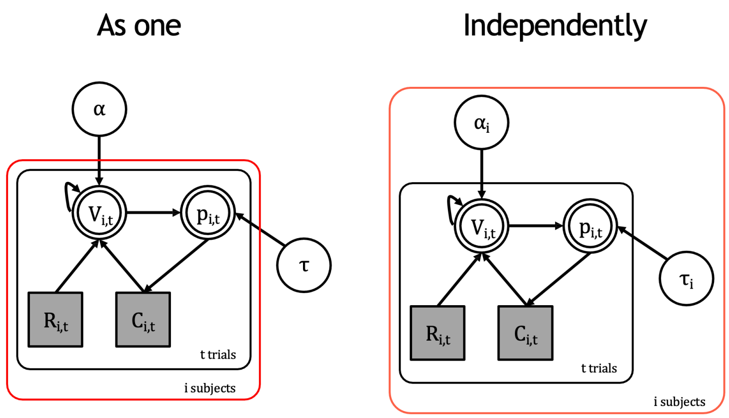
## Fitting multiple subjects

The model above estimates the latent parameters for a single individual. Of course, most studies involve more than one person, and so we would like to estimate parameters for a group of individuals rather than a single subject.

There are several ways to do this.

Directed acrylic graphs for implementing the Rescorla-Wagner model across multiple subjects

Firstly, you could implement a subject-loop depicted on the left diagram above. **However, this approach assumes a single learning rate and single inverse temperature for all participants.** It essentially forces the model to find "average" parameters that might not represent any actual subject well.



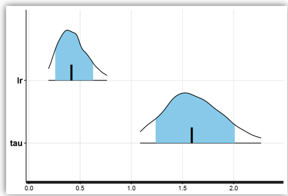

This is equivalent to modeling a hypothetical same-person completing the task $n$ times.

On-the-other-hand, you could also fit multiple participants independently (right side), which **assumes that each person has their own learning rate and inverse temperature.** This is clearly a much better approach as it takes into account individual differences in cognition.

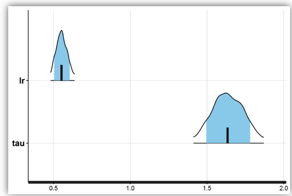
> **Exercise 8**
>
> **1. Run both models using the provided scripts and data. Examine the posterior density distribution and MCMC trace plots for each.**
> Stan models:
>
> - Subject-loop (`_scripts/reinforcement_learning_sp_ms_model.stan`)
> - Individual fitting (`_scripts/reinforcement_learning_mp_indv_model.stan`)
>
> `R` scripts:
>
> - Subject-loop (`_scripts/reinforcement_learning_single_parm_main.R`)
> - Individual fitting (`_scripts/reinforcement_learning_multi_parm_main.R`)
>
> **2. How does the posterior density plots differ between running the model for a single participant and for multiple subjects using a subject-loop?**

**Click to reveal the density plots**

N = 1




N = 10




You can see the HDI is narrower for both parameters when observing the results from 10 subjects versus a single subject.