Block Practical – Session 3

In session 1, you have learnt all about the task and simple reinforcement learning models.

In session 2, you have learnt about how reward and probability are combined to form 'value' and about how values are transformed into choices using a 'softmax' decision rule. You have created simulated behaviour and looked at how your experimental design (e.g number of trials) and your model can be validated using simulated agents.

<u>In this practical (session 3):</u>

- We will recap the important task details, why it is important to simulate data, and what our hypotheses were
- We will read the real data you collected into Matlab
- We will look at simple measures such as accuracy and reaction times and discuss how to check your data quality, including how to define 'outlier' participants whose data may need discarding
- We will then talk about model fitting and apply it to your data using different approaches (grid search; fminsearch). This will give us a set of parameters (e.g. learning rate) for each participant with which we can start asking the interesting scientific questions
- Finally, we will perform a simple model comparison to see whether one or two learning rates are better for explaining the data in this task

Statistical comparisons and your write-up will be covered in the next session

Parts of the session where you're being asked to do something are indicated with an arrow (→).

3.0 Reading in your data and plotting individual participant's choices

You have each collected four data sets now (two in which participants heard neutral sounds and two in which they heard aversive sounds). This means that altogether we have nearly 80 data sets to analyse today which is quite exciting. I have assembled all datasets that you uploaded by Wednesday evening into one file that you can load into Matlab. It's called data.mat and has the following fields

• 'opt1Chosen'	(whether option 1 was chosen 1=yes)
• 'magOpt1'	(the magnitude of option 1 and 2)
• 'magOpt2'	
• 'opt1Rewarded'	(whether option 1 was rewarded 1=yes)
• 'reactionTime'	(the RT on this trial)
• 'chosenOptionRewarded'	(whether the chosen option was rewarded 1=yes)
• 'isStableBlock'	(whether this is a stable block 1=yes)
• 'pointswon'	(the points accumulated in this trial)
• 'Opt1left'	(whether option 1 was shown on the left)
• 'trueProbability'	(the underlying probability of option 1)
• 'playsound'	(whether a sound was played 1=yes neutral; 2=yes aversive)

Open the script run_session3_final.m and evaluate cell 3.0.1; this loads your data.

Have a look at the next cell 3.0.2 and evaluate it. Can you remember what every line of this code does? It should start to look somewhat familiar by now.

→ Now **change the subject number** and evaluate the cell again. Try this a few times with a few different participants. Can you spot any differences between how well different individuals are tracking the underlying schedule? If so, what makes a 'good' subject, what makes a 'bad' subject (note down the subject numbers)? Try and express this in terms of the learning rate and softmax parameter?

____

____

<u>**3.1 Inspecting your data**</u>

<u>Accuracy</u>

Next we want to check the data quality, or in other words, ensure that all participants provided reasonable responses. It is possible that some participants responded entirely randomly. If this is the case, it may be necessary to exclude them from further analyses.

Look at cell 3.1.1, read through it and evaluate it.

→ *Question: How are we defining 'accurate trials' here? What would be a better definition (we can come back to that later in optional part at the end of the practical)? Accepting the fact that we are using 'objective' probability here, is the accuracy calculated correctly otherwise? The == means 'equals' i.e. it compares the left and the right side of the equation and gives a '1' if they are the same. Describe in what situations you will get a '1' = true and in what situations a '0' = false.*

____

____

____

Next we also look at accuracy separately for the volatile and the stable phase (3.1.2).

→ *Question: What would you expect to find?*

____

____

Evaluate the cell 3.1.2. Were you right about your prediction?

Judged by eye, do you think there any subjects that look like 'outliers'?

→ *Add a title, x-label and y-label to the last figure.*

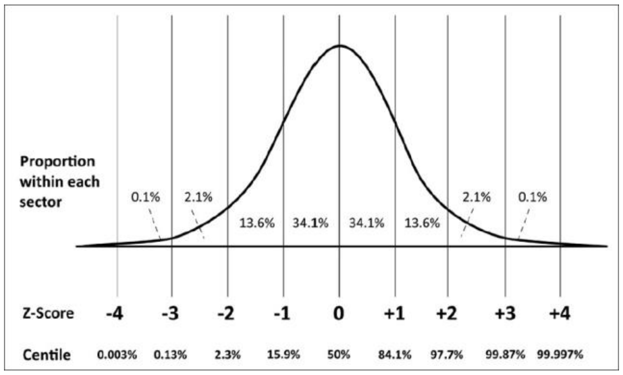
Reaction times

Next let's have a quick look at participant's reaction times as well. Have a look at cell 3.1.3 and evaluate it. What does it show and do any subjects strike you as unusual?

Defining outliers

Of course instead of judging 'by eye', we can also use a statistical criterion, for example to exclude subjects with accuracy or RT mean/variance below or above three times the standard deviation of all participants. This is done in the next cell 3.1.4.

How does the standard deviation relate to the z-score? For this, you may need to remember that the z-score of each participant (for a given measure, e.g. accuracy or RT) is the number of standard deviations their data point lies from the mean.


Proportion within each sector
0.1%
2.1%
13.6%
34.1%
34.1%
13.6%
2.1%
0.1%
Z-Score -4 -3 -2 -1 0 +1 +2 +3 +4
Centile 0.003% 0.13% 2.3% 15.9% 50% 84.1% 97.7% 99.87% 99.997%

→ Look through the code and **add one line** to calculate the total number of participants that this cut-off suggests should be treated as 'outliers'. There are some the tips in the code to help you with this. Change the cut-off to 2.5*SD and check again whether/how this number changes.

____

What do you think is a reasonable threshold to use going forwards?

____

Other aggregate measures

Now that we have started to get a feel for the data by looking at overall accuracy, we can repeat one of the analyses from session #2 and plot whether participants are likely to stay after a win and switch after a loss. As we discussed last week, this is not the most sensitive measure but it gives some insight into the data nevertheless.

Look at cell 3.1.5. and remind yourself what the line that defines Stay=… does, how is the code defining 'staying'?

Evaluate cell 3.1.5 and look at the plot.

→ What can you see and what can we learn from this result?

____

____

____

____

Optional: if you want to do a bit more programming before next week's session (or have time left at the end of this session), you could try and create your own cell 3.1.6 and repeat the same plot but separately for participants in the two stress conditions (aversive sounds or neutral sounds). You can copy and adjust our code for that, or use the 'help' in Matlab to find out what 'figure' 'plot' 'subplot' 'errorbar' 'set(gca…)' 'xlabel' 'ylabel' 'title' etc do

3.2 Fitting choices

A quick primer on Matlab 'functions':

For the purpose of model fitting, some of the code from session 1 and 2 has been put into a 'function' called **RLModel.m**. So far you have mostly worked with 'scripts' in Matlab. Functions are not very different but one important distinction is that they only know the variables that you 'pass on' in brackets behind the name of the function. For example, the call of the function RLModel(alpha) would mean the function RLModel only knows the variable 'alpha'; so everything that a function needs to know in order to complete its calculations needs to be passed on as function arguments in brackets. Functions can also return values/variables, for example error=RLModel(alpha) will return an error variable which is computed within the function RLModel.

You do **not** need to look at the code in RLModel.m as part of today's practical but it might be useful for you to look at it in your own time at home to understand what is being computed. Here is a quick summary of what it does

For a given subject and two parameters alpha (learning rate) and beta (inverse temperature)
- Calculate the subjective probabilities using the reinforcement learning algorithm covered in session 1
- Convert these predictions into choice probabilities using the utility and softmax as covered in session 2
- Calculate the error term, i.e. how far the predicted choices are from the real choices (see below)
- The error term is returned by the function. The smaller this error term, the better the fit.

Have a look at cell 3.2.1 which explains the other inputs that the function needs: alpha, beta, the subject number, the data, and whether or not you want to produce a summary plot of the fit.

3.2.1 Developing an intuition for the goodness of a fit

To help us understand when a fit is 'better' or 'worse', choose one subject and **evaluate cell 3.2.1** This should produce two plots. In each plot, the subject's choices are plotted in red and the model-predicted choice probability for option 1 in green.

Note: For this cell to work you need to be in the folder that contains RLModel.m (check), or do addpath(FOLDER) where the FOLDER is the one that contains the RLModel.m file

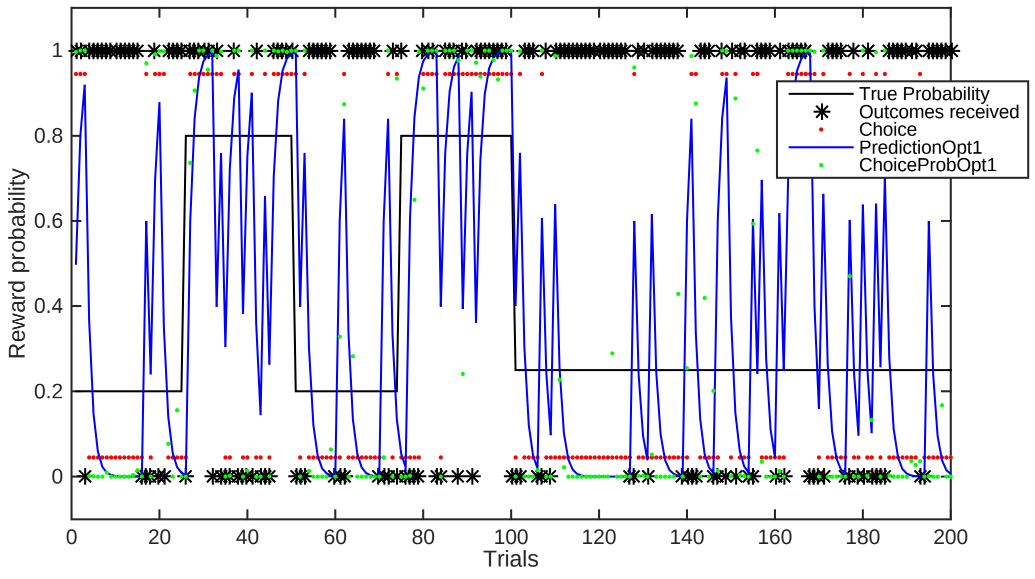
→ *Question: Compare the two fits. Which one do you think is better and how do you tell? Note down here which alpha/beta combination looked like it produced a better fit, and note down which subject you were looking at. In a later section of the practical, you will then be able to compare if the true alpha/beta estimated for this subject is indeed closer to the parameters you thought looked better.*

____

____

____

____

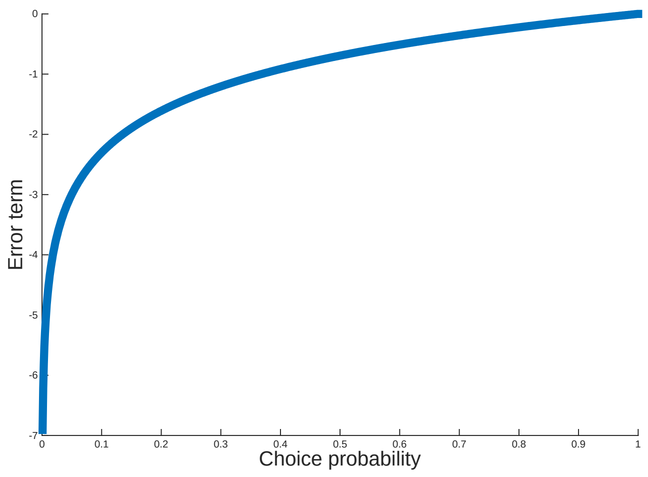


3.2.2 Capturing the fitting error mathematically: log-likelihood

In the previous section, we had to tell 'by eye' whether a fit was reasonable. The way to do that was to compare how far the 'green dots' (predicted choice probability of option 1) are from the 'red dots' (actual choices). Mathematically, we can capture this 'error term' with a very simple equation. We first save the choice probability of the option that was actually chosen in a vector (here: *probChoice*). If the values in this vector were all '1', this would mean that the prediction would be perfect, i.e. on each trial the model would have predicted with 100% certainty that the chosen option would be picked. Because log(1)=0, adding up the log of all predicted values would mean the total is zero in this scenario i.e.

error = sum(log(probChoice))

However, even for very good fits, the softmax function does not normally give a value of exactly 1 for *probChoice*; the values tend to be between 0-1. Look at the **log function** below, plotted for the interval between [0,1] which corresponds to the possible range of values the softmax returns.



→ *Question: If we sum the log of the choice probability as shown in the equation above for all trials, what trials are going to dominate the error term and why would this be good?*

____

____

____

____

The output of RLModel.m is exactly this error term, also referred to as the **log-likelihood**, with just one small difference. All the summed values obtained from the log-transformation are negative. But because we are looking for the 'smallest' error and optimization functions tend to be good at *minimizing* values, we invert the sign so that 'large numbers' now intuitively mean 'large error' and small numbers mean there was a small error.

error = -1*sum(log(probChoice))

This is called the **negative log-likelihood**.

→ *Question: When you ran cell 3.2.1, the error terms were printed in the Matlab command window. Was the fit that you thought looked better 'by eye' indeed the one with the smaller error?*

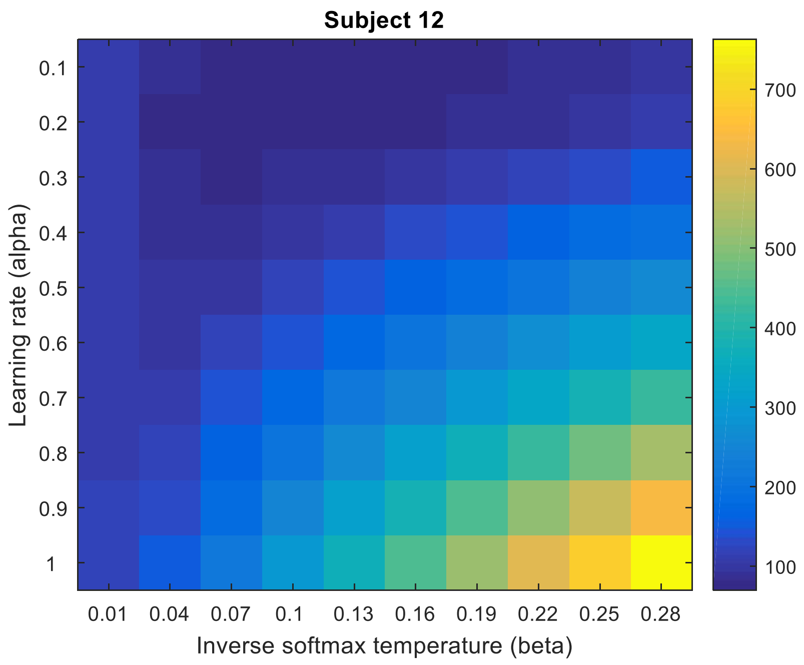
3.2.3 Finding the smallest error/best fit using a grid search

One very simple approach for finding the best fit is to simply try the fitting over a range of different parameter values for both alpha (learning rate) and beta (softmax temperature). This is called a grid-search because all possible combinations of alphas and betas form a 'grid' of parameter values in 'parameter space'. This might become clearer in the plot below. For each of the initializations, the function RLModel.m returns the error that is obtained when fitting the data with that particular combination of parameters, and it is then easy to check which error term is the smallest.

Let's first 'visually' check which value is the smallest for one subject by plotting the error term in a grid as shown below. The values 1-10 on the y axis refer to the ten parameter values for the learning rate, for example here [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]. The same is true for the x axis and the softmax parameter value.

→ *Execute cell 3.2.2 and look at the code and resulting figure. You can use the Matlab 'data cursor' (the black cross with a white 'sticky-note') to look at several of the dark blue values. If you do this for the same participant as above, you might find out that the two combinations we tried earlier were quite far off what now looks like the best combination.*

→ *Try and make the grid (slightly) finer or coarser by changing the two lines of code in 3.2.2 that determine the possible values gridLearningRate and gridInverseT*



→ *Go back to the cell 3.2.1 and replace one of the alpha and beta starting values by what you think is closer to the true 'best fitting' set of parameters based on your grid search (you have to use the same subject in both cells). Looking at it by eye, does the fit look better?*

If you like, you can try the same for a few different participants. This is a really nice way to look at how they differ and get a better intuition for how model fitting works.

3.2.3 and 3.2.4 Grid search separately for stable & volatile

We are now repeating the grid search again for one subject but this time separately for stable and volatile blocks. This is taken care of internally in RLModel.m and is it optional to look at this code if there is time left at the end of the practical.

→ *Execute cell 3.2.3 and run it for a few different subjects. Can you see any differences in the learning rates between stable & volatile blocks in these exemplary participants?*

→ *Execute cell 3.2.4 which does the exact same grid search but this time for all subjects. This might take a moment to run. Can you see a difference in the histograms for stable and volatile learning rates?*

→ *Add one line at the end of cell 3.2.4 to define a variable that contains the average parameter estimate across all subjects that we obtained from the grid search. Call it meanBestParam=…*

We will use this average parameter from the grid search in the next section as one of the initializations for the parameter fitting using Matlab's minimization algorithm.

3.3 Model fitting using fminsearch

While a grid search gives a good intuition for the landscape of your parameters, it is not usually the way parameter fitting is done. This has several reasons. For example, a grid search becomes intractable when the number of parameters is too large. It is also very inefficient because the algorithm might spend a lot of time determining the fit on combinations of parameters that are clearly far away from the minimum, so a lot of computing time can be wasted.

Some fitting algorithms use a method called 'gradient descent' which means that they use the slope of the 'error landscape' you just looked at and moves in the direction where the slope is decreasing, i.e. towards the suspected minimum.

Matlab has an inbuilt algorithm called 'fminsearch' which uses yet another algorithm that is quite efficient. It can still sometimes find local minima, rather than the global minimum, but it is much faster and more precise than a grid search.

→ *Read through cell 3.3.1 and execute it (this might take a little while!). If there is time left at the end, you can write a few lines of code to compare the parameter fits obtained using fminsearch with the parameter fits obtained using the grid search approach above, or look at the optional sections below.*

What do the resulting parameters look like? Is there a clear effect? If so, what does it mean, if not, do you have any idea why that might be?

____

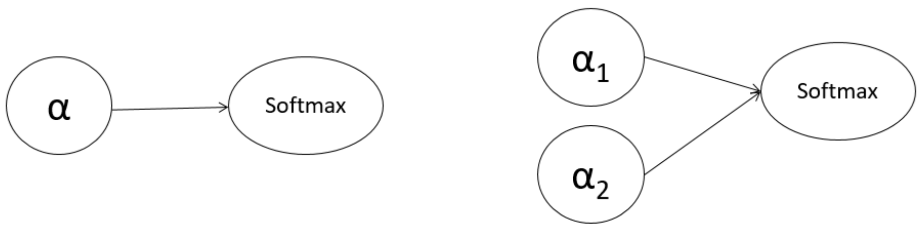
____

____

____

3.4 Model comparison

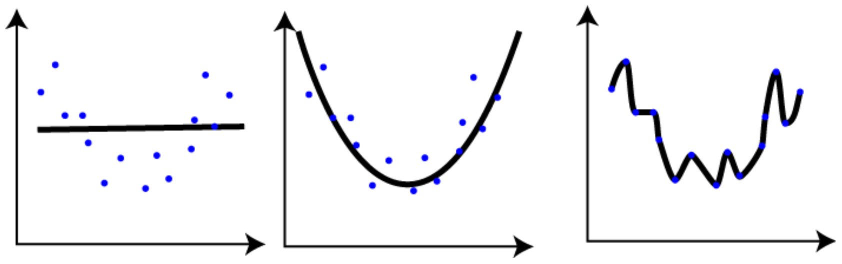
The last question we will try to answer today is whether a model with two learning rates performs better than a model with one learning rate.



We will now repeat the model fitting, but this time with just one learning rate, i.e. the same one for the volatile and stable phases of the experiment. → *Execute cell 3.4.1 for that*.

For both models, we have now obtained a measure of the goodness of the fit, which is the (negative) log-likelihood. The bigger the log-likelihood or the smaller the negative log-likelihood, the better your data was captured using a given model.

However, it is not valid to simply compare the two models based on their log-likelihood. Why is that? A model with more parameters is likely to be better at capturing the data, see below for an example.



So we need to correct for the number of parameters, which can be done using the **Akaike Information Criterion (AIC)**. It follows a very simple formula

AIC = -2*logLL + 2*numParam

→ *In your head, simulate what values (large or small) the AIC formula would give in the four different situations: a model has a large or small logLL and a large or small number of parameters. What does this mean for what values for AIC we expect for the best (winning) model – small or large ones?*

____

____

Because we know the number of parameters in our models (one alpha, one beta = 2 or two alphas, one beta = 3), we can now compute the AIC for each of the models and compare them. This is done in cell 3.4.2.

→ *Read through and execute cell 3.4.2 and look at the AIC difference for all subjects produced in the bar-plot, as well as the difference across the group (obtained by summing across all AIC values) which is printed out in the command window using the command 'fprintf'. What do we conclude from this session, can we say which model is better?*

____

____

Next week, we will use these parameter estimates to perform some interesting statistical tests and you will get your first 'results'! Have a good week until then…

Optional section (pick and choose!):

(1) If you have time left, you could write code here that does the AIC comparison for the people who encountered stable first or the people who encountered volatile first
(2) Try to go back to the plotting of accuracy from the beginning of our session, but now use the subjective RL-prediction of the underlying probabilities to define accurate trials, rather than the objective probabilities which are unknown to the participant.
(3) Write a few lines of code to compare the parameter fits obtained using fminsearch with the parameter fits obtained using the grid search approach above.
(4) Try to see if you can incorporate the fact that some participants might have to be excluded. We have ignored this so far in our plots. We can give you some tips for this!
(5) Try to do some basic statistics on the obtained parameter estimates, for example, comparing learning rates in stable and volatile blocks using Matlab's function 'ttest'.
(6) Look at the function **RLModel.m** and try and understand its different parts.
(7) Can you work out how the stable & volatile blocks are taken into account during the fitting in RLModel.m?
(8) Create a Win-stay/Lose-stay plot for the different stress manipulations (after 3.1.5 e.g. as 3.1.6).