**Cognitive Science Report: Recategorization**

**Introduction:**

This paper is a technical report for a model that uses a specific learning algorithm (hierarchical, precision-weighted prediction error learning) in a non-monotonic learning task (re-categorization). The goal of this paper is to demonstrate 1) the effectiveness of this sort of model in non-monotonic learning, and 2) to demonstrate how this model might be useful for understanding individual differences in learning.

**Task:**

The task used for this exercise was the “re-categorization task”. The re-categorization task is a simple category learning task, in which participants must induce the correct definition of a category by observing a collection of stimuli which may or may not belong to the category in question. Possessing or lacking a specific feature defines category membership. After a set period of time (ideally, long enough for the subjects to have correctly inferred the original category) the category-defining feature is changed.

In this particular version of the task, each stimulus consisted of a vector of 6 binary-valued features. Each stimulus could belong to one of two categories (simply labeled category 1 or category 2). Simulated “participants” viewed each stimulus, and then responded as to which category the stimulus belonged to. After making their guess, participants received feedback indicating the stimulus’ actual category. After 50 such trials the defining feature was switched (there were 100 trials overall). Before the switch category membership was defined by having the value of the 1st feature in the stimulus vector equal to “1”, after switching the category was defined by the 2nd value in the stimulus vector equaling “1” (see line 300 in the code for “sample\_recat\_experiment\_run” file).

When using human subjects, this task has a “cover story”, intended to (partially) conceal the true nature of the experiment. In the past this cover story has been that the subject is an interplanetary biologist learning to classify recently discovered martian bacteria as either oxygen resistant or not. Each bacteria has six binary features which may impart oxygen resistance (e.g. headbulbs vs. no headbulbs, bent vs. straight ribosomes, etc.). I mention this because, when describing the model behavior it seems more natural to use terms like “this bacteria has tail cilia” and “is oxygen resistant” rather than “feature 1 of this stimulus has value “present” “ and “is in category 1”.

**Model**

The model consists of two major components, a performance component, responsible for selecting the best category for a given trial, and a learning component, responsible for learning from feedback after each trial.

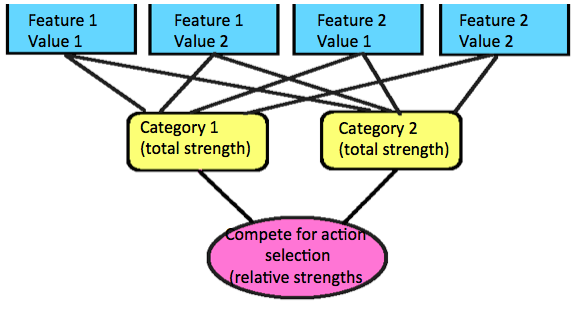
*Performance*

The performance component of the model encodes the stimulus and then selects the best category using prior knowledge of the relevant stimulus dimensions and category labels. The possible values that each feature may take are represented by individual nodes in a feedforward network (blue squares in the illustration below). When viewing a stimulus, the values present in the stimulus activate the corresponding nodes in the network. Once activated, the value nodes activate each of the known categories (via weighted links) in proportion to that value and category’s past co-occurrence frequency, (higher past co-occurences mean stronger link activation). The independent category weights are created by taking the product across all of each category’s active links. A final category is decided upon probabilistically using Gibbs sampling (the probability of a category being selected is given by : (category weight) /(Sum of all category weights) ).

*General form of model, performance (image):*

*Stimuli are encoded as a vector of stimulus values . Category weights (yellow) are calculated from activation of links (black) beween the active values and category definitions (yellow). Link activation is based on past co-occurrence history.*

*Categories compete for action selection (pink) using Gibbs sampling.*



*Learning*

The strength of each value-to-category link corresponds to the marginal likelihood of seeing a specific category given a specific feature value. This value is estimated from past experience. The model learns these associations over time. The model uses a Hierarchical Gaussian Filter (HGF), a Bayesian reward learning algorithm to update the feature to category links (for specifics see Mathys et al. 2011).

There are three important elements to consider for the learning/update function, “*What* gets updated, *when,* and how?”. In the current version of the model, “what” is updated is the link strength (a value between 0 and 1) between the original “category selected” and the values viewed. If the category selected was the correct category, this is considered a success and the link strength between the category and the viewed value(s) is incremented upward (asymptoting at 1). If the category selected was incorrect, the link strength between the category selected and the viewed value(s) is decremented (asymptoting at 0). How these values are updated is given by the HGF learning equations.

Without exhaustively detailing the workings of the HGF, it’s helpful to note the principles behind it, and their possible connection to “non-monotonic” learning. The HGF estimates the co-occurrence between a value and category on a scale from 0 (inversely occurring) to 1 (perfectly co-occurring). The HGF uses a form of error driven learning to estimate co-occurrence frequency. Each time the learner observes a category-value co-occurrence, this is recorded as 1 and subtracted or added (in a weighted manner) to the previous category estimate for that value. When the expected category and the actual category this is counted as a “0” and the weight of the links between the observed values and the actual category is decremented.

When estimating category-value co-occurrences the model also estimates the variability of this mapping (is the mapping consistent, and stable over time, or subject to occasional noise?), and the “volatility” of the environment (changes in variability over time, “true” environmental changes. E.g. has this feature gone from being a stable predictor to being unimportant?). Estimates of variability and volatility differentially impact the learning rates for the co-occurrence estimates. Since learning in the model is based on the error between expected and actual category mapping the (estimated) variability of the mapping is useful for adjusting the learning rate: If estimated variability is higher, then learning should be slower, so as to ignore spurious, but expected, deviations about the (inferred) “true” co-occurence value. In the case of volatility (changes in variability), the model should learn more quickly, as high volatility suggests actual, underlying, environmental change beyond the normal expectations of noise: the model should learn quickly in order to capture emerging regularities. This relates to non-monotonic learning by suggesting a mechanism for triggering non-monotonic change, namely when estimates of volatility exceed those for variability learners should shift from their current conceptions to and search for new ones.

While the variability and volatility are estimated dynamically (based on new information after each trial), there are three Bayesian priors that are static; a parameter for volatility, a parameter for variability, and a parameter coupling the two (this third parameter dictates how strongly changes in estimated variability impact estimates of volatility). These parameters govern learning rates, and therefore impact the overall behavior of the model. Changing these parameters prior to a run of the model could be considered to altering (as-yet-to-be-specified) individual differences, creating a new “learner” profile.

*Note: Comparison to previous versions of this model*

Previous versions of this model used a slightly different update method. The general architecture of input/feature nodes linked to category definitions and updated through a Bayesian reward function was the same, but the nodes themselves encoded the stimuli in a slightly different manner. In previous versions of the model, the input nodes corresponded to binary features, whereas in the current model each node corresponded to the *value* of a particular feature. For example, in the earlier model the feature “ribosomes” was encoded by a single node with two (mutually exclusive) values, arbitrarily assigned “1” or “0”; “1” for straight ribosomes, and “0” for bent ribosomes. In this set-up, observing and updating one value of a stimulus has automatic consequences for the weight assigned to the other values at that feature (e.g. decrementing the weight of “bent ribosomes” after a failed categorization instance, implies a relative increase in the strength of straight ribosomes for the category definition).

In the current model, since each value had its own node, changes in the weight of one value on a feature did not directly effect the changes in weights for other values at that feature. For example, the feature “ribosomes” was represented by two nodes (a “bent node”, “straight node”), each of which updated separately, each node could take a 0 or 1 value indicating whether or not it was present in the stimulus. Note that this set-up does not imply that feature-values are, from the point of view of the model/learner, mutually exclusive (even though the structure of the task is such that any two values of a single feature are, in fact, mutually exclusive). This change allowed the model to update only the weights of the value that was actually observed, a more psychologically plausible assumption.

Changing the model structure like this had some unexpected further implications for the updating rules. While the core update function (the Bayesian-reward learning element) remained unchanged, the interpretation of feedback had to change. In the original model, upon seeing what the correct category was, the model would adjust the weights of all values for the *correct* category definition. That is, if the model had guessed “oxygen resistant”, but the correct category was “oxygen intolerant” then the model would adjust the weights for the features mapping to “oxygen intolerant”. In the new model, since only single values (and not total features) were updated, this type of interpretation of feedback would mean that no value would ever be decremented, leading all value+category links to slowly plateau at the maximum weights (this did not happen in the original model because incrementing one value meant decrementing its complement).

In the new model, on a given trial, feedback was interpreted relative to the category selected by the model for that trial. In this case, the weights of the category-selected+value links were updated (either incremented or decremented, whether the category was correct, or not) not the weights of the actual-category+value links. (This meant that feedback could be used as a generic error signal: If the actual-category matched the expected category then the link strength between the expected category and the values viewed would be increased, while if the actual stimulus category did not match the expected category, then, the links for the expected category would be decreased).

**Model Output**

When run, the model outputs a list containing the updated (dynamic) parameters for each value-category link at each trial and a list of the model’s overall success or failure on each trial. These parameters include the estimate as to whether the given value is part of the category definition (category-link strength), the prediction error between the model’s previous estimate of the category-value strength and the most recent feedback received, the precision weighted prediction error, variability, subjective uncertainty on variability, and environmental volatility. In the results section, I focus primarily on interpreting and explaining the model’s overall success (and failure), as well as the meaning of the category link strength, variability, and volatility estimates for the target and misconception values.

After an experimental run the model produces an output file entitled test\_file.txt, which contains the dynamic parameters for each category+value link. These are formatted as a ‘parameter x trial’ csv. When the model is run using IDLE it outputs a list of trial successes (1 if the model’s prediction matched the trial output, 0 if it did not) through the IDLE interface. To present the results sensibly, I cut and pasted the CSV output and the IDLE output for each of the two different model runs into separate sheets on an excel spreadsheet, and labeled them appropriately. (Note on python code and output files: when running the model, the test\_file.txt opens for “appending” and not “writing” which means that running successive versions of the model will add text to the existing “test\_file” and not overwrite it or save a new version).

**Running the Model**

To run the model open the \_\_\_\_ file

**Variations in the model**

The model has three static parameters, in the form of Bayesian priors. As mentioned earlier, these priors roughly correspond to priors on variability (denoted in the model code as omega), volatility (denoted in the model code as sigma\_3), and how tightly changes in (estimated) variability and volatility influence one another (denoted as “kappa” in the model). I ran two versions of the model, created by changing the variability and volatility parameters. Changing these parameters alters the learning behavior of model. It might be argued that the values of these parameters correspond to individual differences. If this is the case, running the model with different parameter settings corresponds to simulating the behavior of different types of learners. (For example, supposing the kappa parameter represented “cognitive flexibility” changing its value would produce different simulated learners who differed in how flexible they were. The model currently remains agnostic as to which specific individual differences these parameters represent. I plan to explore this at a later point using parameter fitting methods, and comparing the best-fit parameters to empirical individual difference data).

In version One of the model, the three parameters were set using empirical estimates from Iglesias et. al (2013)[[1]](#footnote-1). In version two I created a slower and “stickier” learner. I altered the variability (omega) and volatility (sig\_3) priors which ought to slow down the learning rate. Note that more negative values of omega mean greater variability (this is because of the exponentiation of omega in the update equations). Greater values on the volatility term indicate an increase in expected “meaningful” environmental change.

These changes created a learner that believes that it is in a situation with a lot of error, but that is otherwise stable around some mean. We might hypothesize this is a learner who is less flexible, or has some sort of impairment in executive function (suppressing old information?). Alternatively, we might imagine that this would be the profile of a learner who received the instructions “martian bacteria are not know to mutate rapidly, but your equipment is known to be rather error prone”, suggesting low volatility (mutation rate), and high variability (error prone equipment). (Note, I left the coupling parameter unchanged in the two different model learner types).

**Results**

*Overall model performance*

The model performance was difficult to interpret. The performance on the actual task was at chance for both simulated leaners. Interestingly, the model value-category mappings were generally accurate for the misconception and target features, suggesting that the flaws lie either in the category weighting and selection function, or the updating rates for the “unimportant” values.

Analyzing the co-occurrence estimates and update rates for the “unimportant” values suggests a possible interaction between the update rates and the category selection algorithm. It seems that, since the selection algorithm was based on taking the product of all values active on a given trial, it was prone to being overwhelmed by extreme values. In the case of using the correct value (misconception or target values) this would not matter (as the value estimates should tend toward the extremes, as each positive value always corresponds 1:1 with the category estimate). Problematically, the estimates for “unimportant” value-category mappings seemed to fluctuate between extremes; either zero or one, but never sticking around .5 (an estimate of .5 would have indicated no percieved correlation between the values in question and the category label), as would be expected.

What is especially odd about these extreme fluctuations in otherwise unimportant features is that in the earlier version of the model, using the “feature” rather than “value” based update rule, the equivalent estimates did tend towards 0.5. It seems that the error based update rule (required by the shift from feature to value based nodes) that was introduced in the new model led to updates that tended to favor “extreme” interpretations of fluctuations in feature-category mappings. I think this was the case because, in the earlier model the two complementary values tended to “balance one another out” (increasing one decrreased the other) in an incremental fashion, whereas in the independent values structure of the new model, the weights of one value were not constrained by any others. [I am still not entirely sure how this worked out formally, though.]

Further, (I think) the new update rule makes model behavior more dependent on past model estimates; since the error function is taken relative to the category guessed by the model on the current trial (rather than the actual category value provided by the feedback), the model is prone to compound past estimation errors before eventually correcting them. This should make it prone to more extreme fluctuations in estimates than a model using only the actual category value. That is, both the new and old models will adjust weight “correctly” over time, but the new model will fluctuate more wildly in it’s estimates. The greater extremes in the value-category estimates in the new model will then dominate the selection function, leading to outputs that track closer to the “random,” unimportant features.

*Comparing different types of learners*

On a more “positive” note, the expected differences were observed between the two different types of learners.

High var, low val, slower rate, much slower change (nonlinear changes)

Also of note are the crisscrosses between the variability and volatility curves…

*Comments on this exercise/model performance*

This was a very interesting exercise, and I learned a lot, in unexpected ways. The nonlinearity of model behavior, and by extension, perhaps, cognitive systems was dramatically reinforced. A simple change from my original model, changing the stimulus encoding from a feature to a value based vector, necessitated another larger architectural change moving the feedback interpretation from “actual category” to match/mismatch between the expected and actual categories. Both changes seemed more psychologically plausible than the initial model (changing to a values based updating and match/mismatch rule seem to follow Kahneman’s suggestions that, psychologically “What You See Is All There Is)”, yet led to performance behavior that drastically differs from subjects’ actual performance on the task, while remaining somewhat similar in terms of importance assigned to the target and misconception values. Of further interest was the nonlinearity of the source(s) of the “problems” in the model. The problems seemed to arise from the interplay of the (otherwise beneficial) rapid learning of the HGF algorithm with the greater reliance on past (subjective) estimates implicit in the match/mismatch portion of the update function. These seemed to mutually compound one another; the match/mismatch creating runs of errors which reinforced faulty estimates of value-category weights, and then, incorrect value category weights overly influencing category selection. (I’m not sure if this is ultimately the correct “story” for the problems with the model, but what I can tell from the data is at least suggestive).

**To complete: running the model, instructions**

**Output, graphs, comparisons**

**# possible error in the category selection function?**

Iglesias, S., Mathys, C., Brodersen, K. H., Kasper, L., Piccirelli, M., den Ouden, H. E., & Stephan, K. E. (2013). Hierarchical prediction errors in midbrain and basal forebrain during sensory learning. *Neuron*, *80*(2), 519-530.

Mathys, C., Daunizeau, J., Friston, K. J., & Stephan, K. E. (2011). A Bayesian foundation for individual learning under uncertainty. *Frontiers in human neuroscience*, *5*, 39.

1. The Iglesias experiments used a simple cue-response paradigm ( a single stimulus, varying along a single binary dimension probabilistically cued the presentation of a sing [↑](#footnote-ref-1)