CSW

I think that you now are within reach of simulating the basic finding that blocked / early = good test performance (around 90%) and interleaved / middle / late = worse test performance (around 70%).

if you want to reduce performance in blocked / early from 100% to 90%, you can reduce beta_within

if you want to increase performance in interleaved / middle / late from 50% to 65%-70%, you can use a low mean value of beta_between and add variance around the mean. this approach fits well with the “bimodality” of individual subject plots.

> quote from august 14 2020 email chain: "individual subject plots for blocked/interleaved are attached": "one interesting thing that comes out of the individual subject plots is that, for the interleaved experiments, and for the explicit_interleaved study, you see a fairly bimodal distribution: a few subjects "get it" and learn to asymptote but most are at or close to chance. so, when we see 60% or 75% average accuracy in those conditions, it's a mix of a large number of subjects who don't learn at all, and a smaller number who fully learn, rather than everyone showing 60% accuracy (although there are a few subjects like that)."

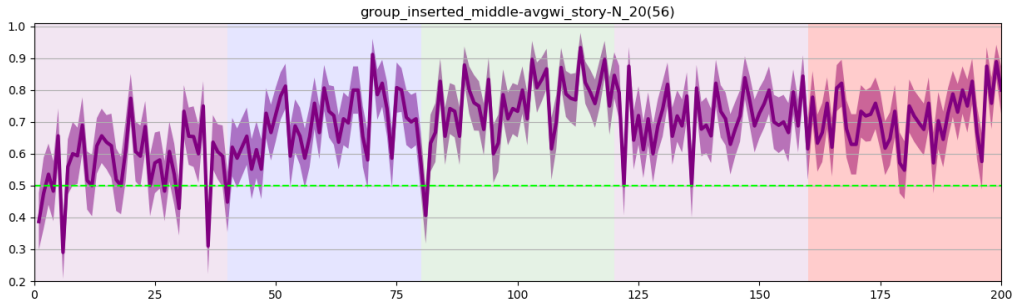
after you’re confident that you can account for average levels of test performance, we should make sure that we’re capturing key features of the prediction timecourse

one deviation between the model and the empirical timecourse is that the empirical timecourse in blocked shows dips at the start of blocks 3 and 4; to capture this, maybe try my idea about modifying how we handle beta_between and beta_within:

> currently, as i understand things, you apply beta_between to layer 0 and beta_within to layers 1, 2, and 3. another way of setting things up would be to “store” the z value (i.e., the active latent cause) at the end of each story, and then apply beta_between and beta_within to ALL layers. so, at all layers, when computing the prior, you first determine a) whether the latent cause matches the z value from the *end of the previous story*, and b) whether the latent cause matches the z value from the *previous timestep*. if condition “a” is met then you add beta_between to the prior. if condition “b” is met then you add beta_within to the prior. and if BOTH conditions are met then you add both beta_between and beta_within to the prior. this way of handling beta_between implements a primitive form of memory that spans multiple timesteps within a story (i.e., it allows the model to remember the previous story’s latent cause, even if it infers a new latent cause at some point within the current story). i think this could potentially work well with the idea proposed in #1 above -- this new approach to the betas will make the previous latent cause more probable at both layer 0 (which is already happening) but also at layers 1, 2, 3, which should pull down performance at block transitions. the tricky

> thing is that, if the previous latent cause becomes *too* probable, then the model will just infer that latent cause again, and it will fail to learn in the blocked condition. the goal is to make the previous latent cause *more* likely (so it messes up performance at block transitions, given the modification described in #1 above) without making the previous latent cause the *most*likely latent cause (i.e., to ensure that the model learns, the most likely latent cause either needs to be cause 1 or cause 2).

we also need to make sure that capture the empirical timecourse of middle and late properly; it differs from what is shown on slide 28 (the real timecourses don't stay below chance after the block switch).



to address this:

first, look at what happens when you add variance to beta_between to get test performance up to 70%; that might help, but it also might not be enough to get rid of the below-chance-after-block-switch issue.

one possibility is that the learning rate is actually *slower* than what we are using now, so the model does not actually improve that much in the first block when has only inferred one latent cause

> adding a learning rate parameter would involve incrementing the M_{tkij} tables by *lrate* instead of by 1 when a transition is taken. put another way, the current way of doing things is equivalent to an *lrate* of 1, but you could also use a smaller *lrate* if you wish. based on equation 4 in sam's document, i'm confident that these lrate changes won't simply cancel out (because the M tables are not the only thing in the equation -- there's also the lambdas…). i'm not sure about whether this change will also require multiplying N_{tk} by *lrate* in equation 1 in order to keep the math "correct".

… my advice is to see how things look after you add variance to beta_between (to boost asymptotic test performance in middle/late) and lower the lrate. that might get you pretty

close to fitting the average performance graph above. if not, here are some other things you can try…

another possibility is to transform the M's from cumulative averages to running averages (which adds a kind of forgetting). this will allow model performance to "recover" more quickly (and to go above chance) after a block boundary, even if it's only inferring one latent cause. this may be a tricky balance, because if it forgets too quickly, this will reduce the influence of the initial interleaved period on the M's and thus make it harder to get different results for early vs. middle and late.

yet another possibility is that some (but not all) subjects are inferring new latent causes at the block switch, which helps them "recover" more quickly after the switch. maybe this splitting would happen in the middle and late conditions at high values of the concentration parameter (even higher than what you tried in your previous parameter search). a potential problem with this approach is that, if participants split at the block switch, they might perfectly learn the schemas (as in the blocked condition) and perform too well. the only way that you can have splitting and *not* have perfect test prediction is if the model infers lots of latent causes, some of which pertain only to the interleaved condition and predict 50/50 transitions, and those "50/50" latent causes are assigned some probability at test (thereby pulling down the overall level of prediction accuracy).

> if the concentration parameter is not effective at increasing the # of inferred latent causes, you might want to consider making splitting stochastic (i.e., split based on the posterior probability of the not-yet-used latent cause) -- if you do this, i recommend trying a system for assigning the z's where, if you split, you record the new (split) latent cause as the z, but otherwise you use the MAP for the z. we may want to look into whether splitting is stochastic in SEM; we can also consult with sam about this general issue...

Appendix: means for different conditions

blocked
M= 0.9279411764705884
S= 0.12507566913824855

blocked rep
M= 0.9068910256410257
S= 0.12700813405464267

interleaved
M= 0.6467741935483872
S= 0.21315051620822667

interleaved rep
M= 0.6477430555555556

S= 0.18879176616127413

instructed
M= 0.6776881720430109
S= 0.20506180876910013

inserted early
M= 0.8934523809523808
S= 0.1498377177074258

inserted early rep
M= 0.9126893939393939
S= 0.12377861158330739

inserted middle
M= 0.7139583333333333
S= 0.20181586681235505

inserted middle rep
M= 0.6640625
S= 0.19229897399441723

inserted late
M= 0.7585069444444444
S= 0.21759110735062448

inserted late rep
M= 0.7236111111111111
S= 0.1789269498847553

1) i'd like to see what happens to the model predictions on slides 36 and 37 when you keep all of the parameters the same but lower beta_between (varying it smoothly from 100 all the way down to zero). i'm envisioning 5 plots: one each for blocked, interleaved, early, middle, late. for each plot, the y axis should be average prediction accuracy during the final test phase (i.e., averaging across timepoints 160-200) and the x axis should be the value of beta_between. i anticipate that, as you lower beta_between, interleaved test performance is going to get better at some point (as will test performance in the middle and late conditions) and i'm curious what the shape of the curve relating beta_between to test performance will look like. hopefully you will be able to find some values of beta_between that yield intermediate levels of performance in these conditions.

2) in the same spirit as #2, i'd like to see the same kinds of plots where you a) reduce beta_within in a graded fashion, holding the other parameters from slide 36 constant; b) vary alpha in a graded fashion (from 0 upwards, past its current value of 1.4 -- maybe do 0, .5, 1, 1.5, 2, 2.5 … up to 10), holding the other parameters from slide 36 constant

3) this is less of a priority than #1 and #2, but i was thinking more about the problem you identified on slide 34. you were able to resolve it by increasing within-story stickiness. however, i think that another way of addressing this would be to engineer things such that -- instead of the model inferring cause zero at story 43, layer 1 -- it infers a NEW cause (i.e., it creates one latent cause for 0-2-4 and one latent cause for 0-2-3). you should think about what you could tweak to make this happen (e.g., maybe changing alpha would increase the likelihood of the model forking off a new latent cause). to look at this more carefully, i would go back to the parameters that generated the curves shown at the top of slide 34, and look at the probabilities of a) cause 0, b) cause 1, and c) the new latent cause (cause 2, i guess) for story 43, layer 1. in the example shown on slide 34, cause 0 was the most probable. what would you need to change to make cause 2 the most probable? again, the most obvious thing would probably be alpha, but changing alpha could mess other things up. to be clear, if you end up being able to fit all of the human data well using the high beta_within value used on slide 36, then you don't need to bother with any of this. but if it turns out that using a high beta_within value messes up other things, you may be able to "retreat" to using a lower beta_within value if you can get the model to infer a new latent cause instead of cause 0 at this critical juncture (story 43, layer 1).

4) here is another thing i thought of that might not be necessary, but i nonetheless wanted to mention: currently, as i understand things, you apply beta_between to layer 0 and beta_within to layers 1, 2, and 3. another way of setting things up would be to "store" the z value (i.e., the active latent cause) at the end of each story, and then apply beta_between and beta_within to ALL layers. so, at all layers, when computing the prior, you first determine a) whether the latent cause matches the z value from the *end of the previous story*, and b) whether the latent cause matches the z value from the *previous timestep*. if condition "a" is met then you add beta_between to the prior. if condition "b" is met then you add beta_within to the prior. and if BOTH conditions are met then you add both beta_between and beta_within to the prior. this way of handling beta_between implements a primitive form of memory that spans multiple timesteps within a story (i.e., it allows the model to remember the previous story's latent cause, even if it infers a new latent cause at some point within the current story). i think this could potentially work well with the idea proposed in #1 above -- this new approach to the betas will make the previous latent cause more probable at both layer 0 (which is already happening) but also at layers 1, 2, 3, which should pull down performance at block transitions. the tricky thing is that, if the previous latent cause becomes *too* probable, then the model will just infer that latent cause again, and it will fail to learn in the blocked condition. the goal is to make the previous latent cause *more* likely (so it messes up performance at block transitions, given the modification described in #1 above) without making the previous

latent cause the *most* likely latent cause (i.e., to ensure that the model learns, the most likely latent cause either needs to be cause 1 or cause 2).

5) i just wanted to mention (for potential future use) that my idea of adding a “learning rate” might be useful for fine-tuning fit to human data. this would involve incrementing the M_{tkij} tables by *lrate* instead of by 1 when a transition is taken. put another way, the current way of doing things is equivalent to an *lrate* of 1, but you could also use a smaller *lrate* if you wish. based on equation 4 in sam’s document, i’m confident that these lrate changes won’t simply cancel out (because the M tables are not the only thing in the equation -- there’s also the lambdas…). i’m not sure about whether this change will also require multiplying N_{tk} by *lrate* in equation 1 in order to keep the math “correct”.

6) let’s say that, for the analyses shown in 3 and 4, you find that changing parameters tends to affect performance in a very “high gain” way (e.g., as you vary beta_between, maybe interleaved test performance will jump right from .50 to 1.00 without spending much time in an intermediate range). if the model behaves like this, it might be hard to fit human data where human performance is in an intermediate range (like .70). one simple way to address this problem would be to say that each parameter value is sampled (for a given simulated participant) from a gaussian distribution around a mean -- if the gaussian is wide enough (encompassing parameter vals that give rise to .50 accuracy and also vals that give rise to 1.00 accuracy) then -- averaging across simulated participants -- you will be able to get intermediate levels of accuracy.