Comparing competing models of retrieval processes

A Bayesian approach to sentence processing

Mick L. van het Nederend

A thesis presented for the degree of
Master of Science

Universiteit Utrecht

Artificial Intelligence
Universiteit Utrecht
The Netherlands
December 2018

Abstract

In order to correctly parse a sentence, its underlying structure needs to be understood. The functional task of every word in a sentence stands in relation to other words through the notion of dependency, and the task for the person taking in a sentence is to lay such dependency links between the word they are currently attending and one of the previously attended words. How exactly the chosen previously attended word is retrieved from memory is often under specified. Therefore Nicenboim & Vasishth (2018) compared two models with each other in terms of their power to describe the speed/accuracy trade-off of this process. We expand on this work by using these two models with a dataset that includes individuals with aphasia. The first model is the activation-based race model. It assumes that resolving syntactic dependencies is related to the *activation* of previously retrieved candidate dependants. When confronted with a new item, these candidates accumulate activation over time. The dependant (and thus, the interpretation) associated with the accumulator that first surpasses its threshold is chosen. The second model is the direct access model which assumes instant access to previously retrieved words. The difference in listening times here is explained by a backtrack-and-repair process that may take place when the initial parse is deemed incorrect. The activation-based race model is implicitly assumes that incorrect interpretations are generally associated with longer listening times, whereas the direct access model is ties incorrect interpretations with shorter listening times. The resulting fits on the empirical data show that although the data tells us that the mean listening times for all of its cross sections are shorter for incorrect trials, the direct access model does not perform better. Instead, the resulting fits indicate that both models have problems fitting certain different aspects of the data.

Acknowledgements

Thanks to Shravan Vasishth, Bruno Nicenboim and Dario Paape for their supervision, knowledge and advice. Thanks everyone else at Vasishth's lab at the University of Potsdam for sharing ideas and knowledge. Thanks to Rick Nouwen for supervision and advice. Thanks to Chris Janssen for sharing this opportunity and being a motivator throughout the master's program.

Contents

1 Introduction

1.1 A case for cognitive modeling

The term Artificial Intelligence (AI) is arguably the most used buzzword in context of current technological advancements. The definition posed by Russell and Norvig, from their standard work *Artificial Intelligence: A Modern Approach*, catches a great deal of the current endeavours within the field: *"The designing and building of intelligent agents that receive percepts from the environment and take actions that affect that environment."* (Russell & Norvig, 1995). As you may notice though, this is rather broad. This is no coincidence, as the goals of AI and its branches are not clear-cut, and the term is often used loosely. There is an important distinction to be made between the two dominant ways of interpreting the field as a whole: on the one hand there is the purist way, aiming for a path that will lead to a broad form of human-level intelligence, and on the other hand there is the more pragmatic approach, which often seems intelligent, but may often boil down to 'a smart way to solve a complex problem'. This distinction is often referred to as "weak AI" versus "Artificial General Intelligence" (AGI), or "strong AI". Weak AI is characterized by being very domain specific and driven by concrete problems, whereas strong AI, a more long-term endeavour, focuses on the *hard* problem of AI: the aim for multipurpose agents that adapt on the fly to their dynamic environment without explicitly instructing every step of the way.

To illustrate, consider Deep Blue (Campbell, Hoane Jr, & Hsu, 2002), a chess engine developed by IBM that bested chess world champion Kasparov in 1996. Considering both the complexity of the game, as well as Kasparov's incredible expertise, a truly remarkable feat. It is however also truly exemplary for weak AI: Deep Blue is an agent incredibly good at performing within the boundaries of an incredibly narrow domain (and agnostic in pretty much all other thinkable domains). Many of these narrow domains have been investigated to a greater or lesser extent, but they are far from trivial to integrate into a general framework, or a general line of thinking.

Although I am positive that a manifestation of intelligence is certainly not only possible in the way it works in humans, there is a lot that can be learned from taking a step back and considering the vast body of work contributed by the cognitive sciences, the neurosciences, (psycho)linguistics and certain branches of philosophy. Of course many AI branches consider these fields, but mostly as a means to reach said narrow goals. One of the ways to glue together this body of research on human cognition with the path to AGI, can be considered to be the branch of cognitive modeling: a branch on the intersection of the cognitive sciences and AI. The goal of cognitive modeling is to understand the cognitive processes that go on within humans and then recreating those processes by a mechanical simulation (a computer, generally). Whilst the results may not always be as awe-inspiring as other AI feats, the methodology and careful examination of empirical intelligence, paves a durable path. Rather than staring towards a single goal application with no guarantee for generalization, or developing a neural network that struggles for explanatory value (Özesmi & Özesmi, 1999; Olden & Jackson, 2002, etc.), consequently potentially leading to a dead end, it is incredibly valuable to consider the workings of human intelligence on a processing level in the incremental build-what-we-know-and-verify-it-as-we-gain-more-understanding modus operandi of cognitive processing.

The building of such models not only is an attempt of moving towards the AGI on the horizon, its feedback loop back towards the understanding of cognition also is a great strength. Using empirical data, theories about the inner workings of the mind can be modeled and then assessed. This two-fold validation is a way of keeping both the engineering and the reverse engineering side in check. The quality of proposed models can be assessed in a rich way, and it is possible for multiple theories and models to be bundled together. Multiple such bundles, or *cognitive architectures* have been proposed, most notably ACT-R (Anderson, 1996). Although cognitive modeling inhibits a sense of an overarching potential for unification, it is necessary to focus on small subsets of cognition at the time.

1.2 The aim of this thesis

In this thesis we will focus on an aspect of natural language: sentence processing. Expanding on the work by Nicenboim & Vasishth (2018), we will implement two Bayesian models to look at the speed/accuracy trade-offs that occur during this cognitive process and at how aphasia may or may not influence it. We will first provide the necessary background information on sentence parsing and the relevance of this research within the context of the current state of the field. After that we will unroll the incremental steps of creating such models. Then we will end by analyzing the models and by reflecting on what the results say about the speed/accuracy trade-offs in sentence processing.

1.3 Sentence processing

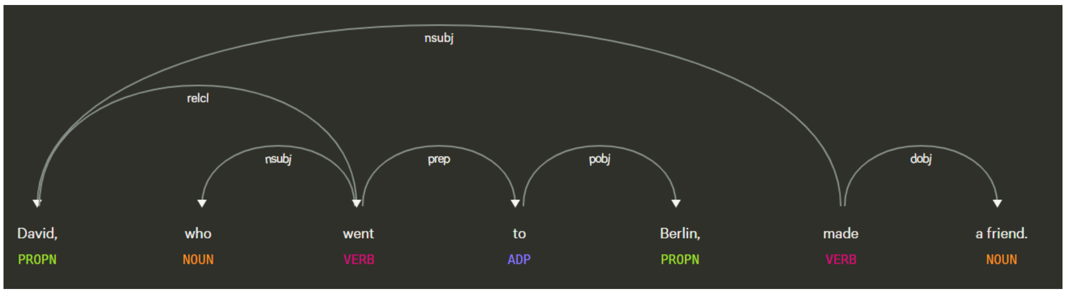


Figure 1: An example dependency parse of the sentence "David, who went to Berlin, made a friend". The main verb "made" is the root here, towards which all other words directly or indirectly point. Most dependencies are adjacent, but the one between "David" and "made" is not. The models attempt to explain what happens in resolving these dependencies. Software: (*Explosion.ai dependency parser*, n.d.)

In order to understand a sentence, we as humans, need to make connections between its words and figure out its underlying structure. This process is called sentence parsing and it gives us an idea about the functional task of each word and its relation with respect to the other words in the sentence. We humans do it constantly and mostly rather unconsciously, but it certainly is a non-trivial task, making it an interesting topic for research. We will reason from the idea that when syntactically parsing the structure of a sentence, the key is to form a notion of *dependency*. A listener (or reader, or ...) is to correctly form these dependencies among the words in order to correctly parse and understand a sentence. Given the notion of dependencies, this final parse has a tree-like structure, with directed dependencies between the nodes (the words), as shown in figure 1. Generally, the main verb is chosen as the root of the tree, with every other word being dependant on either this root or on any other node. Such a dependency may arise from the child's role as an argument (e.g.: nominal subject, indirect object) or as a modifier (e.g.: temporal modifier, determiner) (Jurafsky & Martin, 2009) of its dependant. When a new word is attended, the agent looks for certain traits that the dependant should share. For dependencies that are to be laid in a non-adjacent manner, the working memory serves as a kind of "sketchpad" through which the agent may search, in order to assess the candidate dependency resolvers (McElree, 2000). This notion of working memory is supported empirically, in that data shows that the larger the distance between two co-dependants is, the more difficult it is to process the sentence. Similarly, due to what is called interference, the more candidate co-dependants of a certain goal dependant share similar traits, the more difficult it is to lay the correct link, or to *retrieve* the correct dependant- that is, to re-access the correct part of information that has already been processed in the past (Nicenboim & Vasishth, 2018; Lewis, Vasishth, & Van Dyke, 2006). We assume that previously attended words and phrases are encoded as feature bundles: syntactic key/value pairs (e.g.: [number: singular, tense: past, etc.]). When a new word is attended and a dependency needs to be resolved, it will evoke retrieval cues: expectations of specific values of certain keys. Nicenboim & Vasishth (2018) noticed that how exactly this retrieval process works, is often under specified. The problem with this under specification is that different implementations will say different things about the speed/accuracy trade-off in empirical data. They developed two possible models coming from slightly different assumptions. We will also use these models, but fit them on a richer dataset: one that includes individuals with aphasia (IWA's, described in section 1.6). This, to explore how these models behave when such a language deficit comes into play.

1.4 The models

The first model, the *activation-based race model*, bases the coupling of retrieval cues and the correct previously retrieved feature bundle, or *chunk*, on the notion of *activation*. Previously attended chunks all have a certain activation level that depends on three factors: its retrieval history, the measure in which the features match the retrieval cues and the measure in which competing chunks inhibit similar features (Lewis & Vasishth, 2005). The speed and accuracy of retrievals in turn depend on this level of activation. Like Nicenboim & Vasishth (2018), in order to simplify this theory into a workable Bayesian model, we will use a log-normal race model as proposed by Rouder et al. (2015). In this simplification, items in memory accumulate activation over time. The item associated with the accumulator that first reaches a certain threshold is then retrieved. The accumulation times are log-normally distributed, where the correct interpretation is expected to have a lower μ than the incorrect one. Incorrect responses may occur when the accumulator of the incorrect interpretation is faster due to the random nature of the log-normal function. This means for the speed/accuracy trade-off that this model assumes that incorrect answers will generally be associated with longer listening times than correct ones.

The second model is called the *direct access model* and it is, in contrast to the activation-based race model, based on to the assumption that retrieval of previously accessed chunks is instant, regardless of their retrieval history or cue matching. To account for differences in listening times and comprehension accuracies, the direct access model assumes relatively fixed times for when the correct item is retrieved, as well as the possibility for a backtrack and repair process when an incorrect item is retrieved. If an incorrect item is retrieved and no reanalysis has taken place, it has occurred in a faster total time (one without extra reanalysis time), but the comprehension of the sentence will be wrong. If an incorrect item is retrieved and there *has* reanalysis taken place, the eventual response will assumed to be correct, and the total time will be longer. Because our data is based on sentences normalized to have at most one dependency that is difficult to resolve, we assume that this reanalysis time is distinguishable in the data. The result of this model for the speed/accuracy trade-off thus is that incorrect answers generally have faster response times than correct ones.

Both of these models (described in more detail in sections 3 and 4) will be fitted on the empirical dataset that partially consists of individuals with aphasia (IWA's). Both the dataset itself and the notion of aphasia are described in the following sections.

1.5 The dataset

The dataset we will fit our models on comes from an experiment conducted by David Caplan et al. (2015). The subjects were exposed to three experiments: object manipulation, picture matching, and self-paced listening with picture matching. In this thesis, we are interested in the task that combined self-paced listening task with picture matching. A total of 13 sentence types were tested, but we will only focus on two of those: subject-subject relative clauses, (SS) and subject-object relative clauses (SO). In both sentence types a noun is modified by a relative clause. In the case of an SS-type sentence, the word referring to said noun is the subject, in SS-type sentences that referral word is an object:

1. **SS**: David, who missed his band, went to the United States
2. **SO**: David, who his band missed, went to the United States

In case 1, David is the one who misses his band, whereas in case 2 it is the band that misses David. Intuitively, one might expect the latter to be more difficult to read. Unsurprisingly, it is widely established that this is indeed so: in English, the processing of SO type sentences leads to significantly more problems than SS type sentences (Traxler, Morris, & Seely, 2002; Wanner & Maratsos, 1978, and many others). The advantage of such a clear distinction is that we may assume our models capture this *relative clause type effect*. For both of these sentence types, 20 trials were conducted for each of the 56 IWA's and for the 46 controls. Both the listening times of every chunk and the correctness of the subsequent comprehension question were stored. These comprehension questions tested whether the correct interpretation or its 'counterpart' - passive vs. active / etc. - was chosen. The control subjects that were selected were matched for age and education.

1.6 Aphasia

The richness of the dataset lies in the fact that it includes individuals with aphasia (IWA's). Aphasia is not a single isolated condition, but rather a collection of language deficits that depend on the location and the severity of the brain damage. As Damasio (1992) puts it: *"Aphasia is a disturbance of the comprehension and formulation of language caused by dysfunction in specific brain regions"*. IWA's thus often experience problems when processing sentences. Many possibilities have been posed to explain the nature of this inability, such as slowed processing (Burkhardt, Piñango, & Wong, 2003), intermittent deficiency (Caplan et al., 2015), and resource reduction (Caplan, 2012), all three of which have been modeled in the Lewis-Vasishth model by by Mätzig et al. (2018). However, we will not go into the underlying foundations of this deficit. Instead, we will look how the models are able to deal with the speed/accuracy trade-off in this specific group. Because of the difficulty that this group experiences, we expect the models to show that the performance of this group is worse in at least one of the two measures (comprehension accuracy and total listening times).

In the sections to come we will elaborate on these models and their respective power in capturing specific aspects of the dataset. But since the models are implemented in a Bayesian way we will start off by explaining the mechanics of Bayesian modeling and how this method is implemented in the framework we will be using: Stan (Carpenter et al., 2017).

2 Bayesian modeling using Stan

We use the R implementation of the probabilistic modeling framework Stan (Carpenter et al., 2017). This open source project, named after Stanislaw Ulam, one of the founders of the Monte Carlo method (Metropolis & Ulam, 1949), is rapidly gaining ground among Bayesian statisticians and continues to expand its audience. Below is explained how it works.

2.1 Bayesian inference

In order to fit the models on the data, we first need to create the functional foundation of the models. After this, we have to fine-tune specific parameters of these models to allow them to best align with the data. The first step, the creation of these models is explained extensively in sections 3 and 4. For now, it is enough to know that these models only describe the functional mechanics of the theories. One difficulty is that the theories that they are built upon usually don't specify the expected values for most of the numerical aspect of the models. These values often depend on the details of the empirical set-up that collected the data. What Bayesian modeling allows us to do, is to parameterize these numbers, and to find the values of these parameters that best match the data. Finding the best parameter values starts with a concept called *Bayesian inference*. A typical model thus has a set of parameters, and a set of proposed parameter values, one for every parameter, is called a *hypothesis*. The quality of such a hypothesis depends not only on the difference between the model given the parameter values of the hypothesis, and the real data, referred to as the *likelihood*, it also depends on the data agnostic initial belief on what parameter values should look like, called the *prior probability*. The prior probabilities may be based on empirical data, or on other research. They may also be non-informative, if there is no reasonable prior knowledge.

More formally, Bayesian inference is a way to compute the *posterior probability*: the probability that the hypothesis holds given the likelihood function and the prior probability. A set of these probabilities consequently is a probability distribution. To explain how exactly Bayesian inference works, we need to establish Bayes' theorem to establish a posterior probability distribution:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \tag{1}$$

where D stands for the collected data and H for the hypothesis. In this explanation, we are mostly interested in the terms in the numerator on the right-hand side. First, $P(H)$ is the prior probability: what is the probability of the hypothesis being true given our knowledge of the world? This is a delicate part, because it gives one the chance of adding a very useful complementary element to the data, but if the prior is not chosen carefully, the convergence of the model may be steered into the wrong direction. In this thesis, most priors are either based upon previous research, or rather conservative. The second term is the marginal likelihood $P(D|H)$. This is the probability of the data given the hypothesis. In other words: what is the difference between the model under a given hypothesis and the real data? This is computed by first determining for every data point d the probability that the given hypothesis would be true and then multiplying these probabilities with each other. In Bayesian modeling, this is where the structure of the model is defined. The denominator $P(D)$ is the probability of the data being observed. Because it has the same value for every hypothesis, this normalizing constant can be ignored. We can therefore also state:

$$P(H|D) \propto P(D|H)P(H) \tag{2}$$

It is computationally impossible to use Bayesian inference with multiple parameters (or sets of hypotheses) simultaneously for every possible hypothesis: there needs to be heuristics in place to cleverly pick suitable candidate hypotheses. For this reason, sampling heuristics are developed to 'walk' between hypotheses. This is explained below.

2.1.1 Markov chain Monte Carlo sampling

Recall that that the parameter space to explore consists of all possible parameter values for all parameters. For every set of specific values we can compute the posterior probability using formula 2: we know the prior probability and we can compute the likelihood. Sampling can be regarded as a means to 'walk' over this parameter space. There are multiple different sampling algorithms, but most of them, as well as the ones we will discuss are based on the notion of getting from a sampled hypothesis x to a new sample hypothesis x'. The formula that does it can then be called $Q(x'|x)$. Sampling while using the current position on the parameter space to find the next is called Markov chain Monte Carlo (MCMC) sampling. Stan uses a certain type of MCMC - so a certain type for formula Q - called a No-U-Turn sampler (NUTS), which in turn is based on Hamiltonian Monte Carlo (HMC) sampling. However, in order to develop a sense of feeling for these algorithms, it is best to start off by talking about the simpler Metropolis-Hastings algorithm. This algorithm is a generalization by Wilfred K. Hastings (1970) of the landmark paper by Nicholas C. Metropolis (1953), wherein MCMC is first proposed. The description of this generalization is described rather technically in the paper, but this is simplified below.

2.1.2 Metropolis-Hastings

To initialize the algorithm we start in a random position x on the parameter space (that is, for every parameter we randomly select a possible value, 'we construct a random hypothesis') and calculate its posterior probability $P(x)$. Then for every iteration t, a candidate x' is selected using a random walk function, and its posterior probability is computed: $P(x')$. If $P(x`) > P(x)$ then we accept the sample. If not, we generate a random number $\alpha$ between 0 and 1, and we only accept it if $\alpha > \frac{P(x`)}{P(x)}$. If the new sample is accepted, we store this new sample, else we store the old sample again. This way, samples closer to the optimum will always be favoured, but samples further away can still be selected, depending on the values of α generated. If all went well, after many iterations, the sampling distribution now has a shape very similar to the true distribution. A problem however, is that this method is very biased towards local optima: once the sampler approaches a certain optimum, it is very unlikely for it to traverse out of it, travel through a valley, and reach the next optimum. This can be solved by using multiple *chains*: every chain (or every complete run of the algorithm) assumes a different initial location in the parameter space. If these chains all reasonably converge to the same optimum, we may assume that the optimum we are interested in is found. If not, a so-called multimodal distribution is found: multiple different sets of parameter values reach similar outcomes. If this happens, this may mean that we need to resort to stricter priors, for one of the distributions will most likely align more with the theory than the other(s). Because we started on a random place on the parameter space, the first part of the samples may all be quite far from the real distribution, and, depending on the traits of the data and the random walk formula, take quite some iterations before converging to the interesting part of the parameter space. This section of the iterations is often referred to as the 'burn-in phase'. Its outcomes are often ignored in the analysis process.

2.1.3 Hamiltonian Monte Carlo

In 1987, Simon Duane et al. presented a hybrid Monte Carlo system. It was initially coined as hybrid Monte Carlo (HMC), but it is now better known as Hamiltonian Monte Carlo (not coincidentally, also abbreviated to HMC) (Duane, Kennedy, Pendleton, & Roweth, 1987). This technique has been described more aimed towards our purposes by Radford M. Neal (2011) and by someone who is one of the driving forces behind Stan: Michael Betancourt (2017).

The difference between HMC and the Metropolis-Hastings algorithm lies in the way in which it samples the next sample x' given a current sample x. In order to gain an intuitive sense of the algorithm, one has to imagine the parameter space and its corresponding values as its inverse: the higher the posterior probability a parameter combination will yield, the lower this is represented in this inverted space, as shown in figure 2

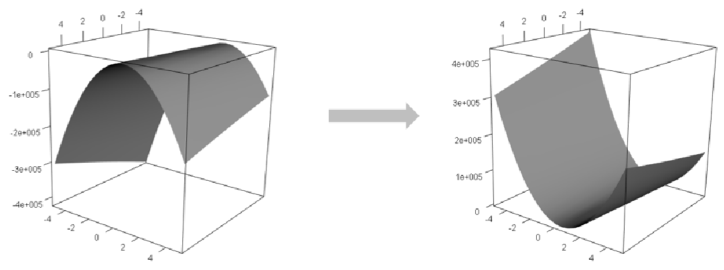


Figure 2: Flipping of the posterior probabilities of the parameter space for HMC.

Now, from a point x, we will get to point x' by choosing a velocity v, a direction d and a time t with which we fire an element from x. Imagine that element physically being sent off under those conditions. It would behave just like a real element would in the physical equivalent of the inverted parameter space. When traveling uphill, it will gradually lose velocity and possibly even stop and turn around. When traveling downhill, its speed increases. Similarly, bends in the parameter space may cause the element to take twists and turns. The velocity and direction will change dynamically according to the shape of the parameter landscape. The location of the element after time t is then taken as the new sample x'. A graphic illustration of an element traveling through such an inverted parameter space is shown in figure 3.

How every point in the parameter space affects the particle that travels through it (or, intuitively, how 'gravity' works on the parameter space), is laid out in a vector field: for every point in the parameter space, an adjustment for direction and velocity is taken into account.

The advantage of this technique is that it still allows for attaining samples distributed similarly to the true parameters, while at the same time taking larger steps at the time. This lowers the amount of samples required, which is a great computational advantage. The chance of it traversing to a second optimum (valley, in this inverted world) also is slightly higher, but still not something this algorithm excels at.

2.1.4 No-U-Turn sampling

The smooth path the particle takes is discretized into leapfrog steps. After each such leapfrog step the gradient in that point is calculated and the momentum from there onward adjusted accordingly. Because of this discretization, the particle may end up

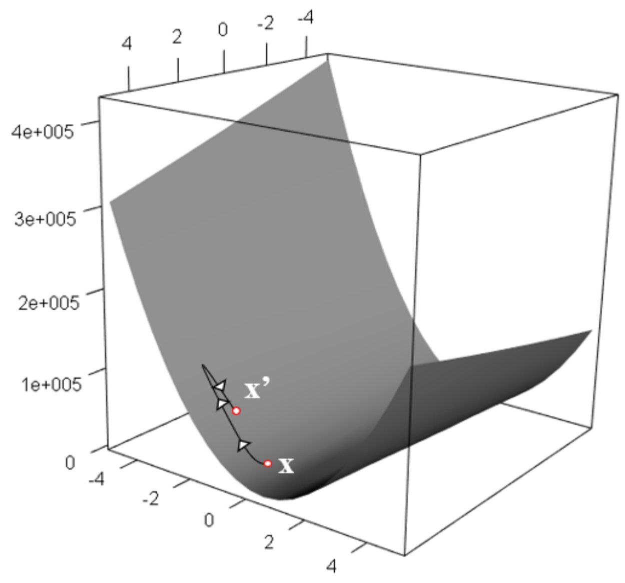


Figure 3: Example of a sampling step in HMC. Note the similarities the path shows with a physical 'kick' from point x in the direction upwards of the slope, during which the gravitational force makes it lose its velocity and causes it to drop down again. The location after time t is chosen as new sample point x'.

in undesirable places. For this reason, a Metropolis-Hastings-type accept/reject step is added, based on the negative log probability and the momentum of the particle. The difficulty is to determine the size and the amount of the leapfrog steps: if the step size ϵ is too small, the computation time can turn out overly high, but if ϵ is too big, many samples may be rejected. Also, if the amount of steps L is too low, the algorithm walks seemingly at random and is no better than the Metropolis-Hastings algorithm, whereas if L is too high, again, the computation time may be impractical. The sampling algorithm as implemented in Stan is built on top of HMC. Because it keeps track of where it has already been, and aims to avoid those places, it will choose ϵ and L dynamically. Because the element will not double back on itself, this final algorithm is called the No-U-Turn sampler (NUTS) (Hoffman & Gelman, 2014).

Now Bayesian interference and the sampling algorithm are explained we will look how this is all captured by our framework of choice: Stan (Carpenter et al., 2017).

2.2 Stan

Stan itself is a C++ framework, but in this thesis we connect to it through Rstan (Stan Development Team, 2018), the interface compatible with R (version 3.4.4) (R Core Team, 2013). The interface allows one to feed input data to a compiled Stan file from R, and receive the output back into R.

2.2.1 Setup

A Stan file is built up out of 7 components, or *blocks*, that all have a different function. They need to be defined in a particular order, and each block shares its scope with the subsequent ones. In order to give an idea about how Stan modeling works in practice, we'll go through them here.

In the **functions** block, functions may be defined. Its main use is to enhance readability and maintainability of the code. In the **data** component the data that will be fed from R is specified. In case of a model

$$Y = \alpha + \beta * X \tag{3}$$

vectors Y (the dependent variable) and X (an unmodeled effect parameter) are fed to the model. Note that in order to use such an unmodeled effect parameter, it has to be translated to numerical values. The cleanest way, and the way we will do it in this paper, is to translate the unmodeled effect parameters from binomial data into the values -1 and 1. This way, a modeled effect parameter (such as β in equation 3) may be attached to (multiplied with) this unmodeled effect parameter and lets one inspect the effect of this unmodeled parameter independently from the rest of the model. The **transformed data** block is used to declare constants. After the data is read, this block may use its values to compute auxiliary ones. In the **parameters** component the modeled parameters are defined. These parameters are eventually sampled by Stan. In equation 3, the unmodeled parameters are α and β. Similar to the transformed data block, the **transformed parameters** may be used to define auxiliary parameters. In the **model** part, the priors and the likelihood function are defined. The priors are defined in statistical notation:

$$\mu \sim Normal(0, 1) \tag{4}$$

This is read as μ is *distributed* as a standard normal distribution with a mean of 0 a standard deviation of 1. The likelihood function may also be written in such a form, e.g.:

$$Y \sim Normal(\mu, \sigma) \tag{5}$$

Alternatively, the `target += ...` notation evokes a similar behaviour: the *target* keyword represents the total log probability. In other words, equation 5 yields the same results as `target += normal_lpdf(Y | mu, sigma)`, albeit in the log space (The suffix `_lpdf` implies that the log probability density function is evaluated). The reason it may be convenient to transform to the log space is rooted in a computational limitations called *underflow*. Recall that the likelihood, $P(D|H)$ is the product of all of the probabilities of the data points d to be true, given a hypothesis. Now, with thousands of data points, all with values v where $0 < v < 1$, most of which values of v are very close to 0, it becomes impossible to plausibly represent those numbers in a computer system; the eventual numbers are microscopic. A property of logarithms is that $log(x * y) = log(x) + log(y)$. The `target` variable is the sum of all of these logs and it allows the algorithm to skip the impossible multiplication method and instead opt for the much easier addition. The final Stan component is the **generated quantities** block. Here it is possible to generate values as the model runs. This may be to disentangle mixes of different parameters, or to generate data to perform posterior predictive checks or leave-one-out-cross-validation with.

2.2.2 Runtime

Once the file is in place, calling the `stan` function will start the Stan process. First the data is read and if defined, the transformed data is computed. Then the initial parameter values are sampled uniformly from the interval (-2,2) and written down. If multiple chains are selected, the Stan makes sure that the distribution of these initial values is diffuse across chains. The sampler-element starts at the point in the parameter space as defined by its initial values and generates a random initial momentum for the Hamiltonian dynamics. Here the first iteration starts by calculating the transformed parameters. Then, using the NUTS algorithm, the particle traverses the parameter space in leapfrog steps, computing the gradient of the negative log probability and the particle's new momentum at every such step. This computation is what Stan spends most of its time doing. If after some leapfrog steps a sample is found, the new parameter values are written down. Finally, the generated quantities are calculated, also written down, and a new iteration is started. After the specified number of iterations, the first set of warm-up samples is thrown out and the `stan.fit` object is returned.

Now the required technical baggage is accounted for, we can take a look at the actual models themselves, which we will do in the following sections.

3 Activation-based race model

Our first model is based on the theory of Lewis & Vasishth (2005). Due to the promising results that cognitive architectures show, Lewis & Vasishth proposed a theory of sentence processing on the basis of such an architecture. The advantage is that this allows the theory to fit in the broader present-day understanding of cognitive processes and ideas about computational architectures. The main aim of the theory proposed by Lewis & Vasishth is to offer a procedural insight into the underlying step-by-step process in processing sentences. It does so computationally: there are mechanics and components to the define the broader computational architecture, such as memories, processes and a control structure, and functionally: specifically for the task of real-time sentence comprehension. The mold that was chosen to embed the theory in is ACT-R: the leading cognitive architecture that combines the consensual assumptions of cognitive processing today (Anderson, 1996).

Within ACT-R, the declarative memory consists of *chunks*: a chunk is a set of key/value pairs, with things the agent knows. In sentence processing, while understanding sentence 1 for instance, a chunk could be [category:NP, case:nominative, number:singular, head:*David*], holding the information about the word *David*.

1. **SS**: David, who missed his band, went to the United States

These chunks may held by buffers: cognitive components with certain roles. Every buffer holds up to one chunk. Every chunk not in a buffer has to be retrieved in order to access it. The Lewis-Vasishth model works with four such buffers: the control goal buffer, the problem state buffer, the retrieval buffer and the lexical buffer, but the main two of interest here are the control goal buffer and the retrieval buffer. When resolving a dependency, the control goal buffer evokes *cues*: expectations that the dependant ought to meet. The probability and latency to retrieve a chunk (to occupy the retrieval buffer) depends on its *activation*: a numeric value that depends on its previous retrievals, as well as on the measure in which the cues match the chunk's key/value pairs and on interference. The chunk of which the activation first exceeds a certain threshold is selected and put in the retrieval buffer, enabling the resolution. The base activation level B for a chunk i is calculated as follows (Lewis & Vasishth, 2005):

$$B_i = ln(\sum_{j=1}^{n} t_j^{-d}) \tag{6}$$

where t_j represents the time since its j'th retrieval, d stands for a decay rate (as the consensus suggests, set to 0.5 (Lewis & Vasishth, 2005; Anderson et al., 2004)), and n is the number of retrievals. The important trait of this formula is that it decays over time, with an sudden increase every time the chunk is retrieved again, after which it decays over time again, albeit slower each time.

The total activation A of chunk i does not only depend on its retrieval history (expressed as base activation B_i). Once a chunk is in the goal buffer and looks for a match, the activation of previous chunks is influenced by their features in relation to the cues that the chunk in the goal buffer evokes. Additionally, the measure in which cues that match the chunk's features also match other chunks' features is taken into account. This phenomenon, called the *fan effect*, makes retrieval more difficult and results in lower activation. This yields the complete activation formula as follows:

$$A_i = B_i + \sum_j W_j S_{ji} \tag{7}$$

Here, index j loops over the retrieval cues of the goal buffer that match with chunk i. The term W_j represents the weight of a particular cue j. In the Lewis-Vasishth model, this term is a normalization factor: $\frac{1}{g}$, where g is the total amount of cues evoked by the chunk in the goal buffer. The term S_{ji} then represents the association of retrieval cue j to chunk i. Because index j loops over cues that are matched in chunk i, association is present. The maximum value this term may assume is S, and it is reduced only by the number of occurrences of the same cue in competing chunks. This fan effect is written as fan(j):

$$S_{ji} = S - ln(fan(j)) \tag{8}$$

As mentioned before, the latency to retrieve a chunk depends on activation. Intuitively: the more active a chunk still is, the less effort it takes to retrieve it. The relation between activation A_i and latency T_i can be expressed as follows:

$$T_i = Fe^{-A_i} \tag{9}$$

where F is a scaling constant, estimated to be 0.14 by Lewis & Vasishth (2005). Recall that the activation of a chunk depends on its correspondence with the chunk in the goal buffer. Due to noise, it may occur that a wrong chunk is retrieved and an incorrect interpretation of the sentence is held.

Rouder et al (2015) propose a way of treating response choices and response times as log-normally distributed accumulators which race with each other. Nicenboim & Vasishth (2018) then proposed a way of connecting this mechanism to the Lewis-Vasishth model. This allows us to conveniently capture the listening times and comprehension task accuracies from our Caplan et al. dataset (2015), while maintaining the functional spirit of the Lewis-Vasishth model. The assumption is that chunks that are candidates for retrieval accumulate their activation over time until a threshold is hit. The chunk of which the accumulator first hits its threshold, is retrieved. The speed of the accumulation depends on the same processes as mentioned in equation 7. Because the sentences in our dataset, there are generally two main possible interpretations of a sentence. We therefore assume that there are two accumulators that race each other, one for SS-type sentences (such as 1), and one for SO-type sentences (such as 2). The difference between this activation-based race model and the Lewis-Vasishth model is that when an interpretation I is chosen in the activation-based race model, the other interpretation I' still accumulated its own activation, whereas in the Lewis-Vasishth model that would not necessarily be the case. However, as suggested by Nicenboim (2018), we can theorize about the retrieval time (equation 9) it *would have had*, if it were retrieved (7). Assuming the noise of activation A_c is normally distributed with a mean μ_c a standard deviation σ_c, we can rewrite equation 9 to:

$$T_c \sim Fe^{normal(-\mu_c,\sigma)} \propto T_c \sim e^{normal(-\mu_c,\sigma)} \tag{10}$$

$$\Rightarrow log(T_c) \sim normal(-\mu_c, \sigma) \tag{11}$$

$$\Leftrightarrow T_c \sim lognormal(-\mu_c, \sigma) \tag{12}$$

As suggested by Nicenboim (2018), the exact values of μ and σ are not of semantic importance. Instead, it is of more value to rewrite μ_c as follows: μ = b - α_c

Where b is an arbitrary constant set high enough to make sure α_c is positive. This way, a higher α_c means a higher accumulation rate.

For both models, we will incrementally build three Stan implementations. The first one, hereafter referred to as the "simple" implementation, is built to make sure the basic premise of the theory works. The second model, the "hierarchical" implementation takes into account random effects, or by-subject and by-item adjustments. The final implementation aims to find out what the role of aphasia is in sentence processing, given the respective models. It does that by adding a group slope parameter for the other parameters.

3.1 Simple implementation

The simplest version of the activation-based race model is listed in appendix A.1 In the data block, the `rctypes` are the relative clause types, mapped from "SS" and "SO" to -1 and 1, respectively. The `winner` vector is a mapping of the interpretations ("SS", "SO") to 1 and 2, respectively. For all trials, the race function is called (note that the input vectors `RT`, `winner` and `rctype` ought to have the same ordering with respect to the trials). This function checks which of the two interpretations has been chosen. To the log_likelihood it then first adds the probability of the winning RT, given the current parameter values, and then it adds the probability that the losing accumulator has *at least* a reaction time of RT, given the current parameter values. The suffixes `_lpdf` and `_lccdf` indicate whether the probability density function (which calculates the probability of certain value to be x) or the complementary cumulative distribution function (probability of a certain value to be at least x) are used.

The effect parameters `rctype` and `beta` are inserted to allow us to distinguish between the four permutations of the answer given in combination with the correct answer. The `beta` values can be regarded as sloped on the intercepts `alpha`:

$$T_c \sim lognormal(b - (\alpha_c + rc_{type} * \beta_c), \sigma) \tag{13}$$

Semantically, this means, that for accumulator α_c, there is an effect-adjustment β_c, for whether or not the answer is in fact true. Note that all data is used here, and the group-effect (IWA vs control) is not taken into consideration on the model level at this point.

We then let Stan fit the model on the dataset, that is, the sampler samples from the parameter space to give us a posterior distribution. Once this is done, it is required to assess the quality of the model: did the model do what we expected it to do? Does the model fit the data properly? How does the outcome say anything about our assumptions and research questions? Stan comes out-of-the-box with a handful of diagnostic tools for these purposes.

The first one we will discuss is called $\hat{R}$, or Rhat. For every parameter fitted, the Stan object will return, alongside its posterior values, a measure $\hat{R}$. This number gives an indication on whether or not the model has converged to a common distribution. Mathematically, it comes down to being the ratio of the between- and within-variance of the chains. This tells us if all chains show similar distributions and converged to the same maximum. As a rule of thumb, the posterior distribution of a parameter can be considered 'healthy' if its $\hat{R}$ value is $<$1.1 (Gelman, Rubin, et al., 1992; Carpenter et al., 2017). As shown in table 1, this is the case for all parameter values for this model.

Secondly, it is important to look at the trace-plot of the parameters, as shown in figure 4. Here, the parameter values of the three chains after warm-up are plotted against the iteration number. If the trace-plots look like straight "fat hairy caterpillars" (Lunn, Jackson, Best, Spiegelhalter, & Thomas, 2012), we assume that the model has converged (Sorensen & Vasishth, 2015). In our case, the trace-plots (in figure 4) look good, and considering the healthy Rhat values, we may assume that this

parameter	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
alpha[1]	19.26	0.00020	0.0104	19.24	19.28	2567	1
alpha[2]	19.18	0.00021	0.0112	19.15	19.20	2728	1
beta[1]	-0.20	0.00019	0.0102	-0.22	-0.18	3000	1
beta[2]	0.28	0.00022	0.0111	0.26	0.31	2634	1
sigma	0.51	0.00012	0.0059	0.50	0.53	2616	1

Table 1: Posterior distributions of the simple activation-based race model

model has converged. For the coming models the trace plots will not be displayed, but you may assume that they look good for our purposes.

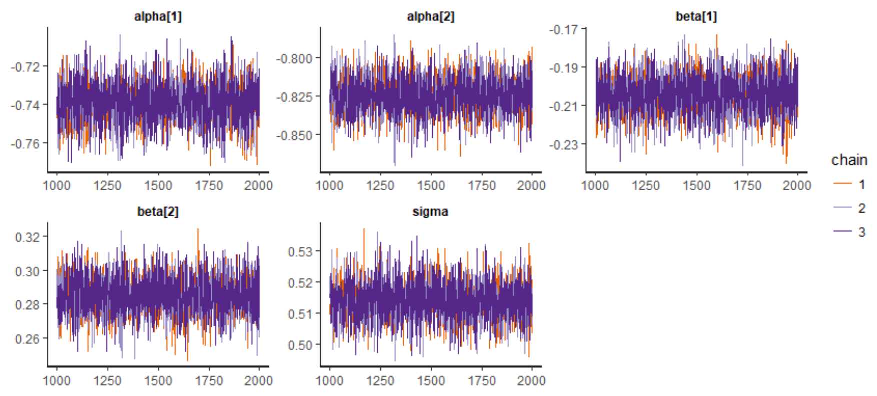


Figure 4: Trace-plot for the all three chains of the MCMC when fitting the simple version of the activation-based race model on the real data. The fact that the chains mostly overlap is an indicator of convergence.

The next step in determining the quality of the model is to generate simulated data using the same model principles and the means of the posteriors as parameters. We generate 6000 data points, equally divided among group type and relative clause type. For every data point, a listening time and accuracy is generated using the posterior means. Once this is done, we fit the model on this newly simulated dataset so that we can compare the originally retrieved posteriors Q against the posteriors from the simulated data Q'. The results are shown in figure 5. The fact that the discrepancies between the posteriors Q and Q' cross zero and are quite small indicates that Stan successfully recovers the true parameters (Furr, 2017).

3.2 Hierarchical implementation

The simple model does not really capture the richness of the data yet. It is likely that certain subjects are generally somewhat faster than others. There might also be a difference in the relative clause type effect (the difference between the *correct* answers, given the *chosen* answers) across subjects. Similarly, some items may have generally been interpreted faster than others. This iteration of the model is to take this into account: the random effects for subjects and items. We generate by-subject adjustments u and by-item adjustments w. The by-subject adjustments u are added to α and the relative clause type effect β, whereas the by-item adjustments are only added to α. This, because the relative clause type effect already is a between-items effect, removing the need for by-item adjustments here. The general effect of the relative clause type, β, would be redundant if by-item adjustments were to be added

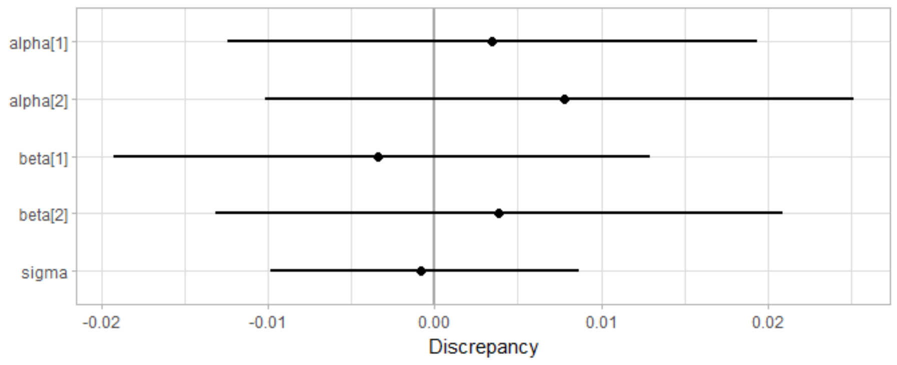


Figure 5: The scaled discrepancies between the posterior means found when fitting the simple activation-based race model on the real data and the posteriors found when fitting the same model on the simulated data. The points represent the simulated posterior means, the lines represent the middle 95% of the posteriors.

here. Given these adjustments, the accumulators look like this:

$$T_c \sim lognormal(b - (\alpha_c + u_{\alpha i} + w_{\alpha j} + rc_{type} * (\beta_c + u_{\beta i})), \sigma) \tag{14}$$

where i is the index of the subject, and j the index of the item. Note that because the two accumulators have different α's and β's, we assume that the adjustments may also differ. This leads to a total of 4 by-subject adjustments, and 2 by-item adjustments.

We assume that the u and w parameters have means of 0 and variances σ_u^2 and σ_w^2. The reason the means should be 0 is that it makes sure that u and w really are just adjustments, correcting for by-subject and by-item effects, and they keep the general model and its general parameter values in tact. To attain the correct values for u and w, we adopt a method as explained by Sorensen & Vasishth (2015). The process described below explains how to build the variance-covariance matrices Σ_u and Σ_w, used to generate the by-subject and by-item adjustment matrices u and w. These final matrices u and w have dimensions $N_{items} \times 4$ and $N_{subjects} \times 2$, one entry for every adjusted parameter for every subject/item, and are generated as follows:

$$u \sim N(\vec{0}, \Sigma_u)) \qquad\qquad w \sim N(\vec{0}, \Sigma_w)) \tag{15}$$

The first step to find Σ_u, is to take the Cholesky decomposition L_u of the correlation matrix C_u. The Cholesky decomposition is an algorithm to decompose a matrix M into a lower triangular matrix L and its transpose L^t. This decomposition has can be considered to be the square root: the main take away here is that $M = LL^t$.

$$C_u = \begin{pmatrix} 1 & \rho_{01} & \rho_{02} & \rho_{03} \\ \rho_{10} & 1 & \rho_{12} & \rho_{13} \\ \rho_{20} & \rho_{21} & 1 & \rho_{23} \\ \rho_{30} & \rho_{31} & \rho_{32} & 1 \end{pmatrix} = L_u L_u^t \tag{16}$$

where the values ρ_{ij} are the correlations between variables i and j (note that $\rho_{ij} = \rho_{ji}$). The prior most often used for this Cholesky factor correlation matrix is the LKJ prior: a prior for the entire matrix with parameter η. The value of η determines the values of the correlations ρ: because we have no real reason to assume any correlation between the different by-subject and by-item adjustments, we set it to 2.0, setting the prior values of ρ near zero.

The next step is to create the diagonal matrix τ of (sampled) standard deviations is created:

$$\tau_u = \begin{pmatrix} \sigma_{u,0} & 0 & 0 & 0 \\ 0 & \sigma_{u,2} & 0 & 0 \\ 0 & 0 & \sigma_{u,3} & 0 \\ 0 & 0 & 0 & \sigma_{u,4} \end{pmatrix} \tag{17}$$

Finally, the $N_{items} \times 4$ matrix z_u is sampled from independent distributions $N(0, 1)$:

$$z_u = \begin{pmatrix} z_{00} & z_{01} & \cdots & z_{0n} \\ z_{10} & z_{11} & \cdots & z_{1n} \\ z_{20} & z_{21} & \cdots & z_{2n} \\ z_{30} & z_{31} & \cdots & z_{3n} \end{pmatrix} \tag{18}$$

To obtain the final matrix with the actual by-subject adjustments u, we multiply the matrices in the following order:

$$u = (L_u \times \tau_u) \times z_u \qquad\qquad w = (L_w \times \tau_w) \times z_w \tag{19}$$

This gives us the by-items and by-subjects matrices, of which every i, j'th element is assigned to item (or subject) i, and parameter j (so either α_1, α_2, β_1 or β_2).

The hierarchical model from formula 14 is now built up and we fit it on the empirical data. The posteriors that Stan returns are presented in table 2. Most parameter values are very close to their counter parts in the simple model, except for `sigma_e` (which is simply called `sigma` in the simple model). This is a sign that the random noise from earlier can now be attributed to by-subject and by-item adjustments, and that the hierarchical model is an improvement over the simple model.

parameter	**mean**	**se_mean**	**sd**	**2.5%**	**97.5%**	**n_eff**	**Rhat**
alpha[1]	19.236	1.1e-03	0.0279	19.1802	19.289	697	1
alpha[2]	19.140	1.0e-03	0.0304	19.0786	19.197	855	1
beta[1]	-0.239	7.5e-04	0.0227	-0.2848	-0.197	925	1
beta[2]	0.336	8.3e-04	0.0272	0.2840	0.390	1070	1
tau_u[1]	0.261	7.5e-04	0.0208	0.2242	0.304	768	1
tau_u[2]	0.250	7.0e-04	0.0217	0.2095	0.294	961	1
tau_u[3]	0.204	5.4e-04	0.0180	0.1703	0.242	1121	1
tau_u[4]	0.235	6.8e-04	0.0211	0.1966	0.279	973	1
tau_w[1]	0.023	4.4e-04	0.0135	0.0013	0.052	963	1
tau_w[2]	0.049	4.0e-04	0.0151	0.0227	0.083	1418	1
sigma_e	0.435	9.6e-05	0.0053	0.4250	0.446	3000	1

Table 2: Posterior distributions of the hierarchical activation-based race model

3.3 Final implementation

For the final implementation, in order to distinguish between the groups, an extra slope parameter for group type is added to every existing parameter. Furthermore, we parameterize the interaction between the group type and the relative clause type. The final accumulators have the following shape:

$$\begin{aligned} T_c \sim lognormal(b - (\alpha_c + u_{\alpha i} + w_{\alpha j} + group * \beta_{c1} + rc_{type} * (\beta_{c2} + u_{\beta i}) \\ + group * rc_{type} * \beta_{c3}), \sigma + group * \beta_{c4}) \end{aligned} \tag{20}$$

The posteriors are shown in table 3. The beta values are all fairly small, except for `beta[3]` and `beta[4]`. These correspond with the rctype however (the model uses a slightly different 'paired' numbering than equation 20). This could indicate that in this model there is no apparent single identifiable effect of aphasia.

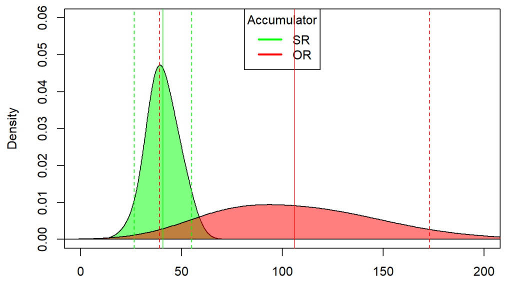
parameter	**mean**	**se_mean**	**sd**	**2.5%**	**97.5%**	**n_eff**	**Rhat**
alpha[1]	19.2278	0.00096	0.0253	19.1779	19.278	696	1
alpha[2]	19.1319	0.00107	0.0296	19.0727	19.188	761	1
beta[1]	0.1129	0.00102	0.0250	0.0643	0.158	599	1
beta[2]	0.0873	0.00097	0.0261	0.0348	0.140	730	1
beta[3]	-0.2455	0.00054	0.0186	-0.2817	-0.209	1182	1
beta[4]	0.3434	0.00073	0.0251	0.2958	0.393	1169	1
beta[5]	0.1266	0.00050	0.0187	0.0903	0.164	1382	1
beta[6]	-0.1149	0.00065	0.0241	-0.1634	-0.067	1356	1
beta[7]	0.0229	0.00014	0.0078	0.0073	0.038	3000	1
beta[8]	0.0093	0.00013	0.0070	-0.0048	0.023	3000	1
tau_u[1]	0.2319	0.00065	0.0193	0.1976	0.274	870	1
tau_u[2]	0.2340	0.00068	0.0209	0.1948	0.277	950	1
tau_u[3]	0.1531	0.00044	0.0159	0.1247	0.186	1289	1
tau_u[4]	0.2064	0.00072	0.0210	0.1676	0.251	850	1
tau_w[1]	0.0204	0.00039	0.0131	0.0009	0.049	1126	1
tau_w[2]	0.0496	0.00044	0.0158	0.0209	0.085	1320	1
sigma_e	0.4336	0.00010	0.0053	0.4235	0.444	2718	1

Table 3: Posterior distributions of the final activation-based race model

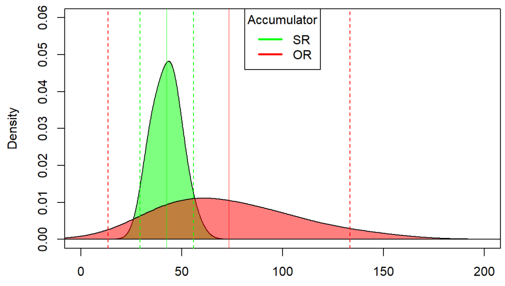
What is interesting now is to visualize the estimated activation values. As mentioned in the beginning of this chapter, the model assumes that there is activation accumulating even for the not chosen interpretations. Using the tie from activation to resolution times (and consequently, listening times), we can then state that this not chosen accumulation gives us a not used resolution time. Because we fitted both accumulators, (the not chosen one with the log complementary cumulative distribution function), and because in the generated quantities field we randomly generated a listening time given the parameter estimations, we are able to plot the set of listening times over the not used listening times. This is done in figure 6.

The shorter the listening times, the faster the accumulator has gotten its activity. The way to look at the figures is by assuming that when parsing a sentence, a listening time from both of the interpretations, so from both 'blobs', is chosen, and the smallest one wins the race. The lack of overlap for example in figure 6a makes for a low chance of the OR accumulator to draw a faster listening time than the SS accumulator. The consequence is that the accuracy will most likely be high. What becomes clear is that the IWA's show a lot more overlap between the accumulators. This indicates that for this group it is much harder to differentiate between the interpretations than for the controls. Note that in any of the cases the eventual resolution time would not be much higher than in any of the others: the problem for the IWA group seems to be that the wrong interpretation accumulates just as quick as the correct one. Additionally, both groups struggle more with SO-type sentences than with SS-type sentences, confirming our initial expectations.

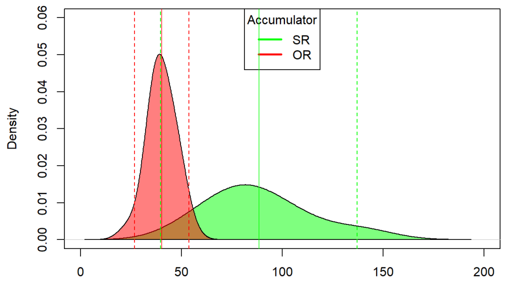
(a) Group - controls, Rctype - SS



(b) Group - IWA's, Rctype - SS



(c) Group - controls, Rctype - SO



(d) Group - IWA, Rctype - SO

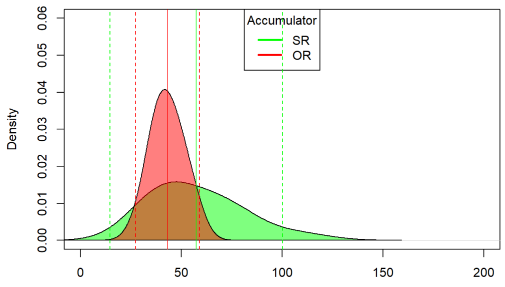


Figure 6: Density plots of the derived activation races, split out per cross section of the relative clause type (that is, the correct interpretation) and group type. We used the model to generate listening times given the μ and σ values that were sampled in that iteration. This allows us to see the potential listening times on the x-axes for each case, even if they weren't selected at that time.

4 Direct Access model

The second model we will discuss will be referred to as the direct access model. The roots of this model can be traced back to McElree (1993). In order to assign a grammatical role to a word, syntactic information is necessary. Because of the ambivalence of the English language however, this syntactic information is not always clear, and often ambiguous. Connine et al. (1984) showed that through this ambiguity, for most cases, a certain structural relation is often preferred over others, often related to its relative occurrence in natural language. An example given is the verb "*watch*" - its preferred form is transitive (e.g.: "Alice watched the cat") rather than intransitive ("Alice watched with the cat"). McElree proceeds by showing that one of the empirical signals of this idea is that experimental reading times are faster when the preferred form is used over the non-preferred form. Additionally, locally, the reading times increase at the critical point that resolves the preceding ambiguous part, if that critical point resolves to the *non-preferred* option. This indicates that a certain preferred way of interpreting a sentence is expected, until there is more information to come to resolution. When such a situation occurs, it leads to a moment of reanalysis, which will be the basis of the direct access model.

This model, like the activation-based race model, also uses the notion of cue based feature bundles that may or may not match with some goal bundle. However, instead of activation determining retrieval time, the assumption is that retrieval is instant: there is *direct access* to the representations (Nicenboim & Vasishth, 2018). Consequently, in contrast to the activation-based race model, the direct access model doesn't draw a dependency between the retrieval times and the probability of the items being retrieved. Instead, in cases where a misretrieval happens (or, in the spirit of the paragraph above, where the correct interpretation is not the initially preferred one), the agent may backtrack and reanalyze the representation to get to the correct interpretation. This backtracking takes extra time. The total processing time then is a mixture of two cases: one with a 'base' processing time wherein no reanalysis is done (where the interpretation may or may not be correct), and one with said 'base' processing time as well as some reanalysis time wherein the reanalysis is done (and we assume the interpretation then is correct) (Nicenboim & Vasishth, 2018).

So computationally, the direct access model explains retrieval time (consequently in our case, listening times) as a mixture of two possible events: one wherein the target is retrieved incorrectly, and one wherein the target is retrieved correctly, and a reanalysis may or may not occur. It is assumed that the probability of correct initial retrieval θ differs per sentence type, and the probability of performing a reanalysis once an incorrect initial retrieval has been made, P_b, is similar within subjects across sentence types. With our data in mind, we draw a distinction between three cases: a correct answer given by the participant may either be due to a correct initial retrieval (1) or by a reanalysis (2). An incorrect answer is coupled to an incorrect initial retrieval and no reanalysis (3). Given a sentence type x, we can then see that the probability of reanalysis is $(1 - \theta_x) * P_b$):

Answer is correct:

1. Initial retrieval correct: $lognormal(\mu, \sigma)$ with probability θ_s (21)
2. Reanalysis done: $lognormal(\mu + \delta, \sigma)$ with probability $(1 - \theta_s) * P_b$ (22)

Answer is incorrect:

1. No reanalysis done: $lognormal(\mu, \sigma)$ with probability $(1 - \theta_s) * (1 - P_b)$ (23)

where μ is the log mean of the listening times without reanalysis, and σ is the log standard deviation. Note that for the strength of the model it doesn't matter whether wrong answers arise from gambling or from a wrong conviction. If there would be a portion of correctly gambled answers that would skew the interpretation of θ, but not its mechanics. For the sake of this thesis, however, we assume that wrong answers are truly believed to be correct by the participants.

4.1 Simple implementation

First we establish a simple, non-hierarchical model, as listed in B.1. We add the relative clause type effect only to the θ parameter. This, because the model assumes that both (controlled for length) sentence types can be read just as quickly if no reanalysis is required. The difference in listening times is explained by the fraction of initially incorrect retrievals, which depend on the nature of the sentence types (subscript s in equations 21, 22 and 23). From an implementation standpoint, when adding this relative clause type effect to θ a problem arises: we want Stan to freely discover the parameter space without constraints and we want to avoid taking the log of numbers $\theta + rc_{type} * \beta \leq 0$. Therefore, we convert this parameter with the sigmoid function:

$$sigmoid(x) = \frac{e^x}{e^x - 1} \tag{24}$$

This function has a domain from $-\infty$ to ∞ on the x-axis, and squeezes all corresponding y-values in the $[-1, 1]$ range. This makes sure that no problems occur if $\theta + rc_{type} * \beta \leq 0$, while shifts in values make for shifts in the same direction for the eventual probabilities.

In order to capture in the likelihood function the three described probabilities from equations 21, 22 and 23, these probabilities themselves are taken as an increment on the log likelihood alongside the log-normal evaluations. The first step is to convert the probabilities into the log space like this:

$$P(acc = 1) = \theta + (1 - \theta) * P_b \Rightarrow LSE(\theta, log(P_b) + log(1 - \theta)) \tag{25}$$

$$P(init = 1) = \frac{\theta}{\theta + (1 - \theta) * P_b} = \frac{\theta}{P(acc = 1)} \Rightarrow log(\theta) - log(P(acc = 1)) \tag{26}$$

$$P(reanalysis) = \frac{(1 - \theta) * P_b}{\theta + (1 - \theta) * P_b} = \frac{(1 - \theta) * P_b}{P(acc = 1)} \Rightarrow log(1 - \theta) + log(P_b) - log(P(acc = 1)) \tag{27}$$

$$P(acc = 0) = (1 - \theta)(1 - P_b) \Rightarrow log(1 - \theta) + log(1 - P_b) \tag{28}$$

Note that LSE stands for the *log_sum_exp* function which adds up the exponents in the log space, which is analogous to normal addition in the linear space.

The second step is to add these probabilities to the likelihoods evoked by the log-normal distributions of the listening times. If the accuracy is equal to 1, the total likelihood function then looks as follows:

$$\begin{aligned} &P(acc = 1) + LSE(P(init = 1) + lognormal(LT|\mu, \sigma_e) \\ &\qquad P(reanalysis) + lognormal(LT|\mu + \delta, \sigma_e)) \end{aligned} \tag{29}$$

For an accuracy of 0 the total likelihood function is:

$$P(acc = 0) + lognormal(LT|\mu, \sigma_e) \tag{30}$$

Running the complete model on the dataset, the posterior distributions as shown in table 4 are found. Note that parameter `alpha` stands for the θ before its transposition to the sigmoid function.

parameter	**mean**	**se_mean**	**sd**	**2.5%**	**97.5%**	**n_eff**	**Rhat**
mu	10.38	0.00056	0.0179	10.342	10.41	1017	1
alpha	-0.13	0.01210	0.3653	-0.806	0.65	912	1
beta	-0.38	0.00229	0.0850	-0.566	-0.23	1373	1
sigma	0.44	0.00013	0.0053	0.434	0.45	1822	1
delta	0.15	0.00087	0.0337	0.081	0.21	1507	1
P_b	0.61	0.00291	0.0775	0.421	0.70	707	1

Table 4: Posterior distributions of the simple direct access model

It becomes noticeable that the beta is negative, which indicates that $\theta_{SS} > \theta_{SO}$. This is in line with the consensus that SS-type sentences are generally easier to process than SO-type sentences. Furthermore, the values for delta seem very low. However, if we convert the values back to linear space, we can calculate the actual effect of δ as follows: $e^{\mu+\delta} - e^{\mu}$. This gives us a reanalysis time of over 2 seconds. Another point of attention is that the values for alpha seem on the low end: plugging in the mean value in the inversion of the sigmoid function yields an average θ of around 53%. On the other hand, `P_b` seems to be on the high end with a mean of 0.61. We expect the `P_b` to be more around 0.3, bumping up the values of θ in the process (Nicenboim & Vasishth, 2018). It is possible for the probability-related parameters to be 'flipped', and still get relatively good predictions for accuracy.

When generating the simulated data, we assume $\sigma = 0.3$, to somewhat reduce the noise (the posterior mean is 0.44). Apart from this σ value, the discrepancies between the retrieved posteriors from the simulated data and the retrieved posteriors from the real data are close or through zero (figure 7. This again indicates that the model retrieves the true parameters.

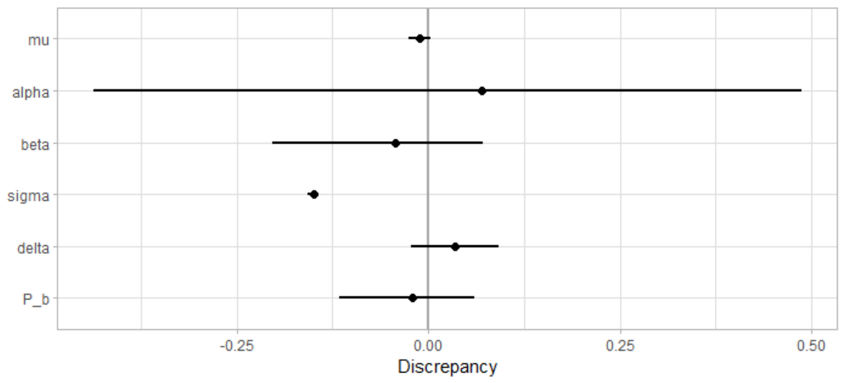


Figure 7: The scaled discrepancies between the posterior means found when fitting the simple direct access model on the real data and the posteriors found when fitting the same model on the simulated data. The points represent the simulated posterior means, the lines represent the middle 95% of the posteriors

4.2 Hierarchical implementation

For the hierarchical implementation the same technique is used as in the activation-based race model (section 3.2). By-subject and by-item adjustments are attached to μ and θ:

$$\mu' = \mu + u_\mu + w_\mu \qquad \theta = sigmoid(\alpha + rc_{type} * \beta + u_\theta + w_\theta) \tag{31}$$

parameter	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
mu	10.422	1.0e-03	0.0217	10.37990	10.465	466	1
alpha	1.278	7.5e-03	0.2593	0.75305	1.775	1185	1
beta	-0.471	1.4e-03	0.0749	-0.62221	-0.325	3000	1
tau_u[1]	0.204	5.6e-04	0.0159	0.17527	0.239	822	1
tau_u[2]	2.014	7.3e-03	0.2480	1.55116	2.537	1148	1
tau_w[1]	0.025	3.7e-04	0.0112	0.00323	0.047	904	1
tau_w[2]	0.670	4.0e-03	0.1523	0.42456	1.018	1440	1
sigma_e	0.402	8.5e-05	0.0047	0.39337	0.412	3000	1
delta	0.028	4.7e-04	0.0255	0.00065	0.094	3000	1
P_b	0.343	1.6e-03	0.0558	0.21544	0.432	1243	1

Table 5: Posterior distributions of the hierarchical direct access model

The results of this model are presented in table 5. The `sigma_[e]` here is slightly lower than the `sigma` in the simple model, like in the activation-based race model indicating that the adjustments were able to localize and explain some of the noise. In addition, the values for `P_b` (0.34) and `alpha` (1.28, corresponding to a mean θ of 0.78) now seem to be aligned in the expected configuration, in line with previous work (Nicenboim & Vasishth, 2018). The value of δ has dropped significantly however, to around 600ms in the linear scale. The model could be having problems with the difference in initial retrieval correctness between groups, leaving little room for the reanalysis time to do its work. We will explore this in the next section.

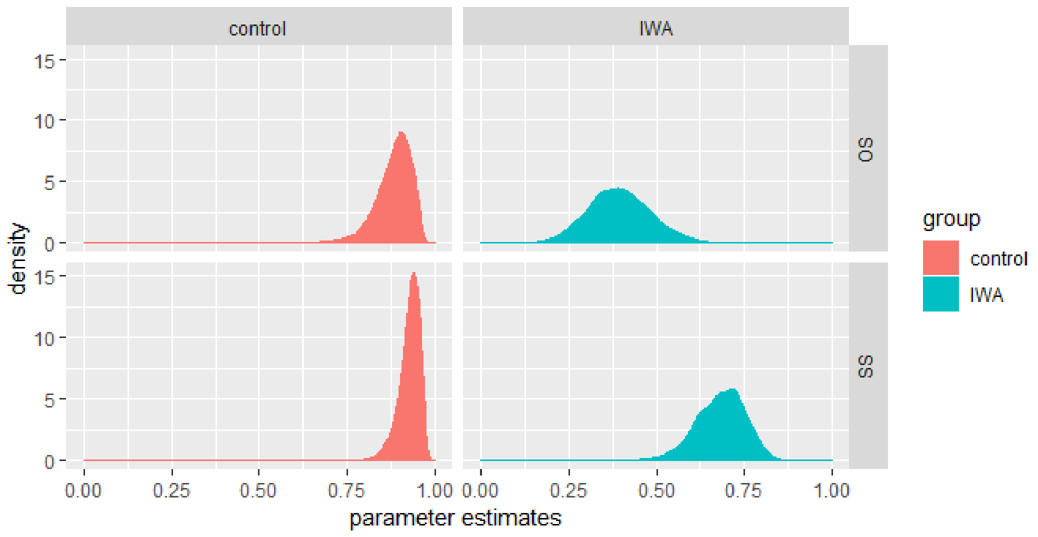
4.3 Final implementation

The final implementation (appendix B.3) takes the hierarchical model and adds fixed effects for group types (IWA's vs controls) to parameters μ, δ, P_b, σ_e and θ, and adds the interaction between the relative clause type and the group type to parameter θ.

The results in table 6 show that this approach indeed allows for larger reanalysis times `delta` (around 1.5 second on average). As shown in section B.3, the associated group effect parameter for δ is β_5. Recall that controls are represented as -1 whereas IWA's are represented as 1. The values of `beta[5]` then indicate that the reanalysis time for IWA's is shorter than that for controls.

When recomputing the θ values from `alpha` and its respective `beta`'s, we see in figure 8a that controls have a much easier time initially retrieving the correct parameter values, whereas IWA's struggle with this. The effect of relative clause type is also very noticeable and as expected: SS sentence types are gotten right on first analysis more often than SO sentence types. Interestingly, the posteriors of the P_b parameters are somewhat closer between groups, as shown in figure 8b, but the controls still seem to perform reanalysis more often. This is under the assumption that the fat tail on the lower end of the probability scale is due to the difficulty to estimate something sensible for subjects with close to perfect accuracies.

(a) Values for θ



(b) Values for P_b

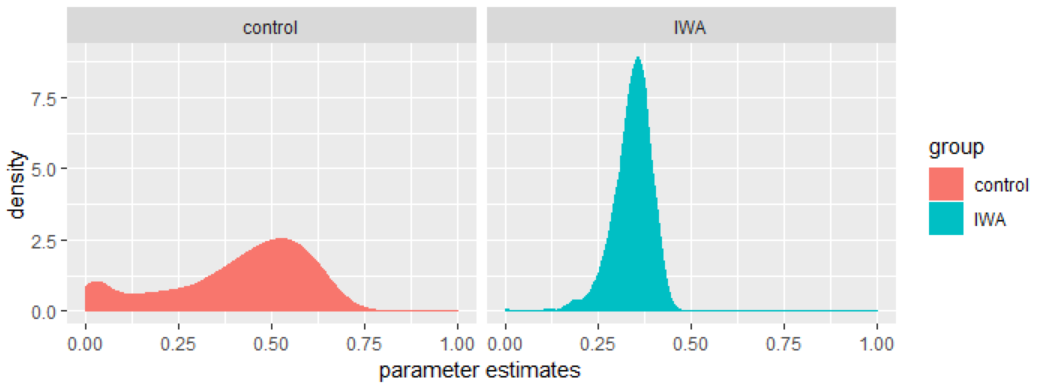


Figure 8: Density plots of the posterior distributions of θ and P_b per cross section. Note that the θ values are quite pronounced, which indicate that the initial retrieval is where IWA's experience the most problems. The lower 'bump' at the controls may be explained by the difficulty to properly estimate it for participants with (close to) 100% accuracy.

parameter	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
mu_0	10.42270	1.1e-03	0.0229	10.3788	10.4673	418	1
alpha	1.27376	8.1e-03	0.2684	0.7366	1.7830	1105	1
beta[1]	-0.44035	1.6e-03	0.0860	-0.6170	-0.2824	3000	1
beta[2]	-1.10275	1.0e-02	0.2840	-1.6745	-0.5513	778	1
beta[3]	-0.17485	1.5e-03	0.0796	-0.3259	-0.0165	3000	1
beta[4]	-0.03979	1.2e-03	0.0206	-0.0813	0.0013	280	1
beta[5]	-0.04307	6.8e-04	0.0371	-0.1247	0.0132	3000	1
beta[6]	-0.01284	3.6e-02	0.7549	-0.7757	2.1736	435	1
beta[7]	0.00066	8.3e-05	0.0046	-0.0084	0.0095	3000	1
tau_u[1]	0.20039	6.0e-04	0.0156	0.1721	0.2328	686	1
tau_u[2]	1.61792	6.0e-03	0.2143	1.2384	2.0695	1261	1
tau_w[1]	0.02493	3.7e-04	0.0110	0.0039	0.0474	870	1
tau_w[2]	0.68257	4.4e-03	0.1591	0.4323	1.0637	1297	1
sigma_e_0	0.40220	8.5e-05	0.0046	0.3932	0.4116	3000	1
delta_0	0.06415	6.7e-04	0.0366	0.0100	0.1481	3000	1
gamma	-0.67351	3.7e-02	0.7673	-2.9405	0.0877	438	1

Table 6: Posterior distributions of the final direct access model

5 Model comparison

In order to see if the models make sense, we run posterior predictive checks: we simulate data from the posteriors and compare that to the real data. In the world of Stan, this means that for every iteration, in the generated quantities field, we use random number generators wherein we plug the parameters of the current iteration to generate for every permutation of fixed effects we find in the real dataset, an accuracy and a listening time. We thus have $\#_{iterations} * \#_{real\_datapoints} = 11859000$ generated data points in total. If the simulated data is very similar to the real data, it does not necessarily imply that the model is adequate, but when the simulated data is very different from the real data it means something is going wrong. For this reason, these posterior predictive checks can be considered sanity checks (Shiffrin, Lee, Kim, & Wagenmakers, 2008; Nicenboim & Vasishth, 2018). The accuracies are plotted in figure 9, and figure 10 shows the listening times. The violins represent the densities of the generated data, and the crosses represent the means of the real data. The violin plots in figure 9 show the average accuracies for the 3000 samples for every subject/item(/accuracy) combination. If the data could have been generated by the model, we expect the crosses to be inside the violins.

For the accuracies (figure 9), both models do a good job. The activation-based race model estimates the controls slightly on the lower side, and the direct access model is on point. This can be explained by the fact that the direct access model explicitly models the probabilities that determine the accuracy. The activation-based race model does this implicitly, by comparing two randomly generated numbers representing the listening times associated with the accumulators with each other, yielding a more noisier trade-off between fitting on the listening times and accuracy.

The listening times are plotted in figure 10. This part clearly shows that neither model performs very well. Recall that the differences of the two models in part lied in their ways of handling the speed/accuracy trade-off. The activation-based race model assumes that incorrect responses are slower than correct ones, while the direct access model allows the correct answers to take on longer listening times. The means of the real data (the crosses) indicate that the direct access model should have an edge here, because the incorrect responses correspond with faster listening times across the board. The activation-based race model fits pretty well on the correct listening times, but then fails at properly accounting for the incorrect listening times. Interestingly, in the IWA/SO cross section of the simulated data, the incorrect responses are about as fast as the correct ones. This could be explained by the noise that comes with sampling and the low accuracy of this cross section that enforces a lot of overlap between the accumulators (and consequently, listening times). The direct access model performs reasonably well for the IWA/SS cross section of the data, but overestimates the listening times in the other cross sections. Furthermore, all cases show a very long tail that the activation-based race model does not have.

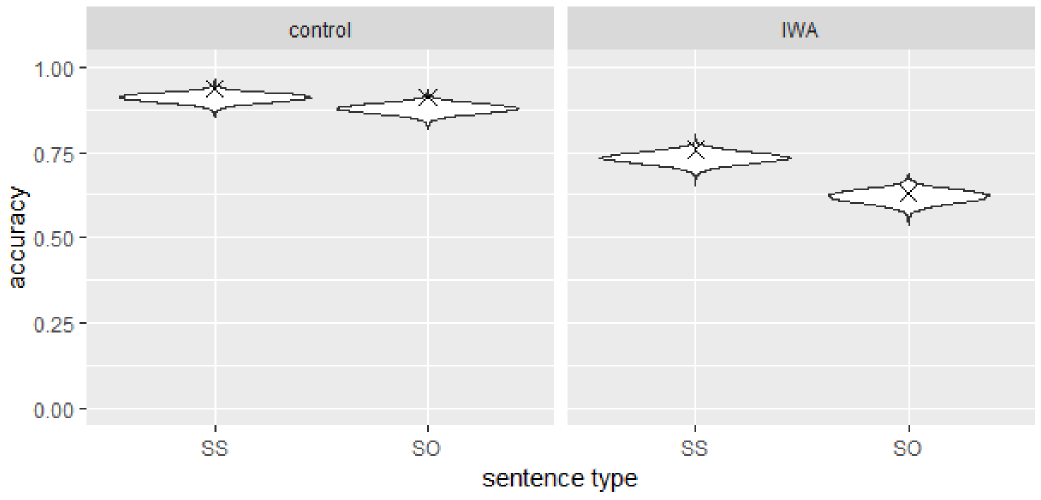
In order to compare the models with each other in a more quantitative manner, we will perform leave-one-out-cross-validation on the log likelihoods of the samples (Vehtari, Gelman, & Gabry, 2016). The values for the pointwise out-of-sample prediction accuracies $\hat{elpd}$ are -44848.3 (SE=66.5) for the activation-based race model, and -44835.8 (SE=69.4) of the direct access model. This yields a difference $\hat{elpd}$ of 12.5 (SE=22.2). We can therefore state that no model is clearly better than the other.

We can look further into the $\hat{elpd}$ values to see how the different models compare in capturing certain cross sections of the data. This is visualized in figure 11. For every data point from the empirical data both models have a $\hat{elpd}$ value that explains the capacity of the model to capture the observation. The difference between these $\hat{elpd}$ values is plotted on the y-axis for every listening time in the real data on the x-axis. Values close to zero indicate no or a very small difference between the two models. Points greater than zero indicate that the direct access model performed better for this observation, and points smaller than zero indicate that the activation-based race model did a better job.

For all the incorrect answers (except for maybe the less outspoken IWA/SO cross section) the direct access model performs better than the activation-based race model the shorter the listening times are. When the listening times are longer, the activation-based race model starts to show its strength. This is in line with the conclusions drawn earlier from figure 10, as well as with the findings of Nicenboim & Vasishth (2018). Regarding the correct answers, there is less discrepancy between the models in SS-type sentences. The models both generally perform quite similarly. For the SO-type sentences however, the models clearly show different performances. The activation-based race model scores much better on the shorter listening times. For the extremely long sentences, the direct access seems to do slightly better, which could indicate that this is where the long tail is used.

Accuracies

(a) Activation-based race model



(b) Direct access model

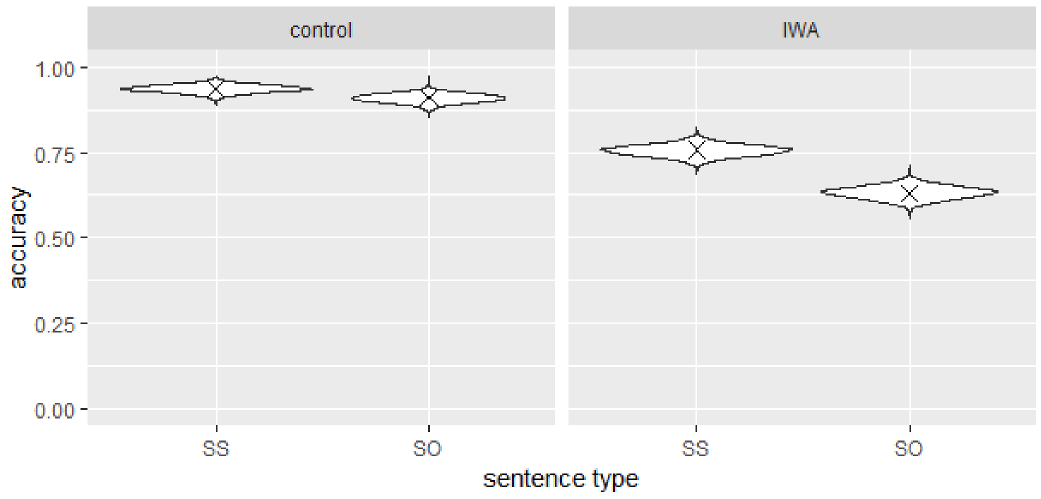
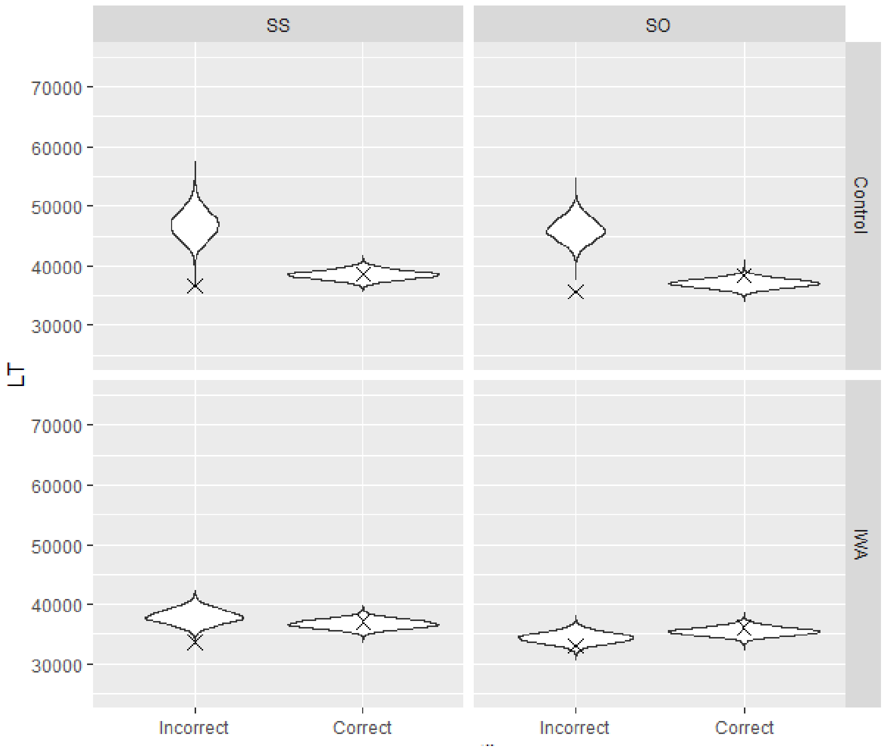


Figure 9: Violin plots of the accuracies as predicted from the posteriors, split by group and sentence type. The width of the plots stand for the density of the predicted accuracies and listening times, the crosses for the respective means of the real data.

Listening times

(a) Activation-based race model



(b) Direct access model

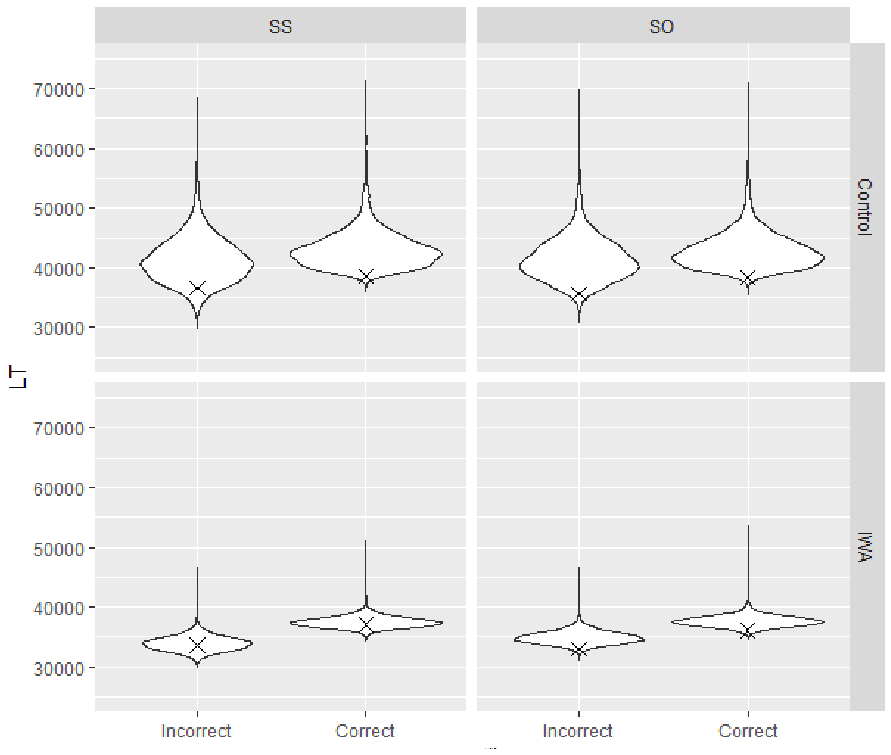
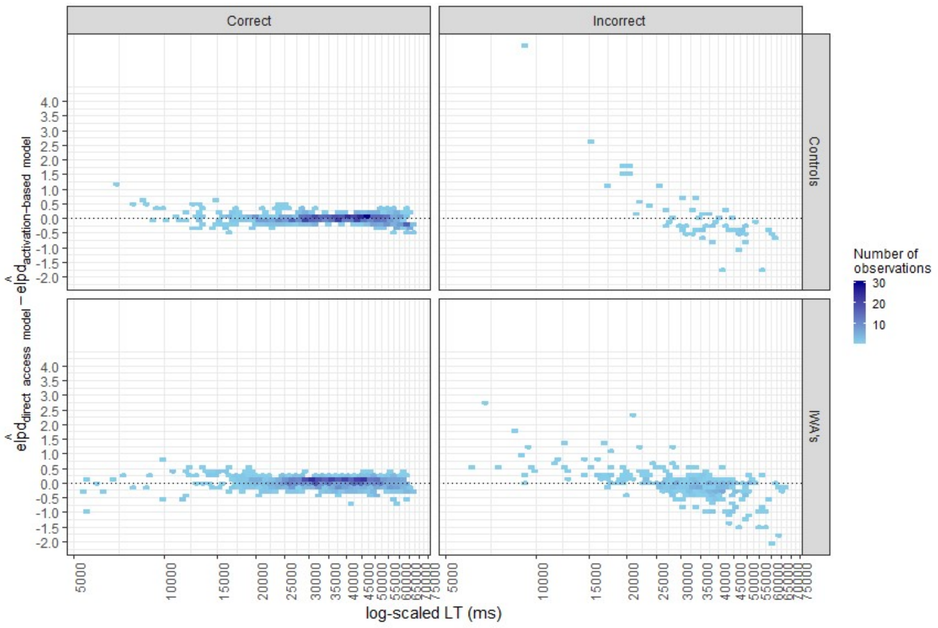


Figure 10: Violin plots of the listening times gained from the posteriors, split by group and sentence type. The width of the plots stand for the density of the predicted listening times, the crosses for the respective means of the real data.

Model comparison

(a) Relative clause type: SS



(b) Relative clause type: SO

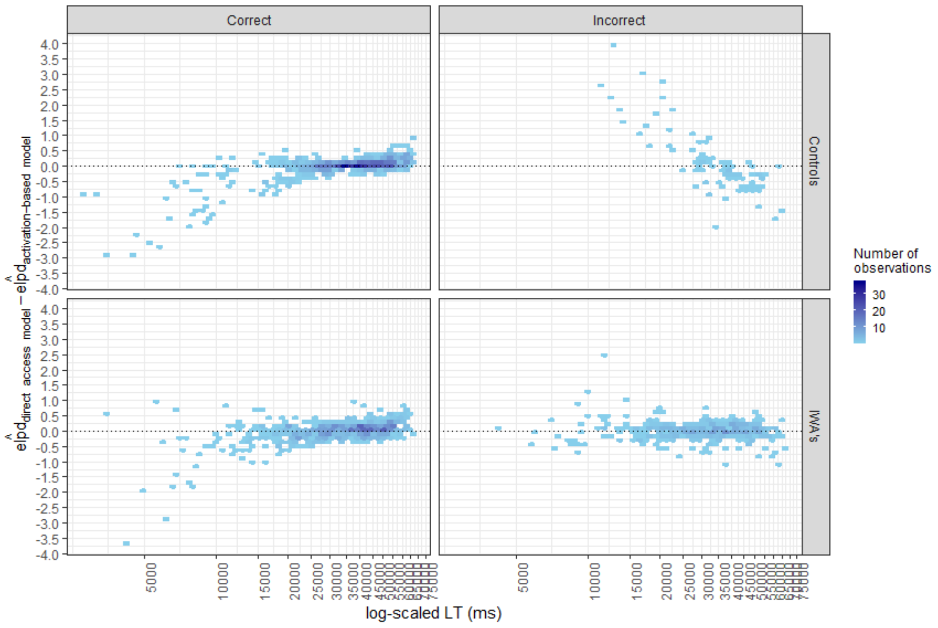


Figure 11: The comparison of the $\hat{elpd}$ values of the two models for each observation. The y-axis shows the difference in the expected log pointwise predictive density for that particular observation. Positive values indicate better performance by the direct access model, and negative values better performance by the activation-based race model. When multiple data points are close together in both dimensions, this is indicated by a darker color.

6 Discussion

The goal of this thesis was to see how the models as proposed by Nicenboim & Vasishth (2018) handle the speed/accuracy trade-off in a dataset that is enriched with data from IWA's. Additionally, the incremental way of implementing these models is unrolled and supported mathematically.

The results of the accumulators of the activation-based model make sense: when presented with SO-type sentences, the accumulators show more overlap than when the subjects were presented SS-type sentences. This is in line with the general consensus that SO-type sentences pose more processing difficulty. Also, the accumulators for IWA's showed more overlap in both relative clause types. This also makes sense given the difficulties these individuals have shown to have regarding sentence processing. Furthermore, because on average, listening times are faster when an incorrect answer is given, this model has a difficult time handling the associated listening times of these inaccurate trials. It is only at the higher end of the listening times of the incorrect responses that the model does better than the direct access model.

The direct access model has shown the ability to make sensible predictions about the parameters for the correctness of initial retrieval θ and about its closely associated parameter P_b that stands for the probability of reanalysis given an initial incorrect retrieval. If we assume the model to be correct, we can state that the big problem for IWA's is that the initial retrieval is more difficult for them. Once a mistake has been made, they will try and repair it only slightly less often than controls do. The final accuracies that arise from these parameters also match the observations very closely. Like the activation-based race model, the direct access model generally also has a difficult time fitting the listening times. It works well for the listening times associated with incorrect responses that lie on the shorter end of the scale. For longer listening times, the activation-based race model performs better.

The different levels of performance of the respective models in the different segments and listening times could indicate that even though the models are simplifications of the full theories, the speed/accuracy trade-off is more delicate than either of these two proposals. The findings of Nicenboim & Vasishth (2018) are similar in how the different models handled the listening times for the respective accuracies. Because the data shows that incorrect answers are on average associated with faster listening times, a case could be made choose to elaborate on the direct access model rather than on the activation-based race model. The bulk of the direct access model consists of a simple log normal distribution function, so there is still a lot to be added in terms of complexity. However, the fact that the direct access model still does not show a clear advantage in the leave-one-out-cross-validation also indicates the strength of the activation-based race model. A possible enhancement to the activation-based race model would be to allow two kinds of parses: a more superficial parse, having less accurate (noisier) but faster accumulators to race each other, and a more elaborate one, with longer listening times but also with a higher accuracy. This would likely allow the model to show the speed/accuracy trade-off as found in the real data. This effect can also be attained by dynamically changing the thresholds of the accumulators. When the threshold of an accumulator is lower (and the listening time shorter), the relatively higher amount of noise will cause the model to more often opt for a wrong answer.

This exploratory thesis, as well as the work by Nicenboim & Vasishth (2018), show that both of the functional retrieval theories have their shortcomings, and that it is not a trivial aspect of sentence processing. It is an aspect that should not be under specified, but instead be researched and explicitly elaborated on. Within the broader scope of AI, we indulged ourselves in a functional aspect of the cognitive process of sentence parsing. We examined it and conclude that there is more to it than first meets the eye. This encourages us to look more closely at this facet and to come up with a better theory. The goal is to develop *the* theory that could account for the fine-grained delicacies that we find in our observations. In the meanwhile, all steps in the direction of this theory are also tiny steps towards AGI, and we hope to have taken such an ever so small step by means of this thesis.

A Activation based model Stan files

A.1 Simple model

```
functions {
  real race(int winner, real RT, real[] alpha, int rctype, real[] beta, real b, real sigma){

    real log_lik;
    log_lik = 0;

    if(winner==1){
      log_lik += lognormal_lpdf(RT|b - (alpha[1] + rctype*beta[1]), sigma);
      log_lik += lognormal_lccdf(RT|b -(alpha[2] + rctype*beta[2]), sigma);
    }
    else {
      log_lik += lognormal_lpdf(RT|b - (alpha[2] + rctype*beta[2]), sigma);
      log_lik += lognormal_lccdf(RT|b -(alpha[1] + rctype*beta[1]), sigma);
    }
    return(log_lik);
  }
}
data {
  int<lower = 0> N_obs;
  int<lower = 1> N_choices;
  int<lower =-1, upper = 1> rctype[N_obs];
  int<lower = 1, upper = N_choices> winner[N_obs];
  vector<lower = 0>[N_obs] RT;
}
transformed data {
  real b; //arbitrary threshold
  b = 30;
}
parameters{
  real beta[N_choices];
  real alpha[N_choices];
  real<lower=0> sigma;
}
model {
  alpha ~ normal(20,2);
  sigma ~ normal(0,2);
  for (n in 1:N_obs) {
     target += race(winner[n], RT[n], alpha, rctype[n], beta, b, sigma);
  }
}
```

A.2 Hierarchical model

```
functions {
  real race(int winner, real RT, real[] alpha, int rctype, real[] beta, real b, int N_choices, vector u,
      vector w, real sigma_e){

    real log_lik;
    log_lik = 0;

    if(winner==1){
      log_lik += lognormal_lpdf(RT|b - (alpha[1]+u[1]+w[1] + rctype*(beta[1]+u[3])), sigma_e);
      log_lik += lognormal_lccdf(RT|b -(alpha[2]+u[2]+w[2] + rctype*(beta[2]+u[4])), sigma_e);
    }
    else {
      log_lik += lognormal_lpdf(RT|b - (alpha[2]+u[2]+w[2] + rctype*(beta[2]+u[4])), sigma_e);
      log_lik += lognormal_lccdf(RT|b -(alpha[1]+u[1]+w[1] + rctype*(beta[1]+u[3])), sigma_e);
    }
    return(log_lik);
  }
}
data {
  int<lower = 0> N_obs;
  int<lower = 1> N_choices;
  int<lower = 1> n_u;
  int<lower = 1> n_w;
  int<lower =-1, upper = 1> rctype[N_obs];
  int<lower = 1, upper = N_choices> winner[N_obs];
  vector<lower = 0>[N_obs] RT;

  int<lower = 1> N_subj;
  int<lower = 1> N_item;
  int<lower=1> subj[N_obs];
  int<lower=1> item[N_obs];
}
transformed data {
  real b; //arbitrary threshold
  real min_RT;
  b = 30;
  min_RT = min(RT);
}
parameters{
  real beta[N_choices];
  real alpha[N_choices];
  real<lower=0> sigma_e;

  cholesky_factor_corr[n_u] L_u; // equation (16)
  cholesky_factor_corr[n_w] L_w; // equation (16)
  vector<lower=0>[n_u] tau_u; // equation (17)
  vector<lower=0>[n_w] tau_u; // equation (17)
  vector[n_u] z_u[N_subj]; // equation (18)
  vector[n_w] z_w[N_item]; // equation (18)
}
```

```
transformed parameters {
  // construction of (19)
  vector[n_u] u[N_subj];
  vector[n_w] w[N_item];
  {
    matrix[n_u,n_u] Sigma_u;
    matrix[n_w,n_w] Sigma_w;
    Sigma_u = diag_pre_multiply(tau_u,L_u);
    Sigma_w = diag_pre_multiply(tau_u,L_w);
    for(j in 1:N_subj)
      u[j] = Sigma_u * z_u[j];
    for(k in 1:N_item)
      w[k] = Sigma_w * z_w[k];
  }
}
model {
  //priors
  alpha ~ normal(0,10);
  beta ~ normal(0,1);
  sigma_e ~ normal(0,2);
  tau_u ~ normal(0,1);
  tau_u ~ normal(0,1);

  L_u ~ lkj_corr_cholesky(2.0);
  L_w ~ lkj_corr_cholesky(2.0);
  for (s in 1:N_subj)
    z_u[s] ~ normal(0,1);
  for (i in 1:N_item)
    z_w[i] ~ normal(0,1);

  for (n in 1:N_obs) {
     target += race(winner[n], RT[n], alpha, rctype[n], beta, b, N_choices, u[subj[n]], w[item[n]], sigma_e)
          ;
  }
}
```

A.3 Final model

```
functions {
  real race(int winner, real RT, real accum_1_mu, real accum_1_sig, real accum_2_mu, real
      accum_2_sig){

    real log_lik;
    log_lik = 0;

    if(winner==1){
      log_lik += lognormal_lpdf(RT| accum_1_mu, accum_1_sig);
      log_lik += lognormal_lccdf(RT|accum_2_mu, accum_2_sig);
    }
    else {
      log_lik += lognormal_lpdf(RT| accum_2_mu, accum_2_sig);
      log_lik += lognormal_lccdf(RT|accum_1_mu, accum_1_sig);
    }
    return(log_lik);
  }

  // RTs for ppc
  vector race_rng(real mu_1, real sig_1, real mu_2, real sig_2, int rctype){
    vector[2] gen;
    real accum_1_RT = lognormal_rng(mu_1, sig_1);
    real accum_2_RT = lognormal_rng(mu_2, sig_2);

    if(accum_1_RT < accum_2_RT){
      gen[1] = accum_1_RT;
      if(rctype == -1){
        gen[2] = 1;
      }
      else {
        gen[2] = 0;
      }
    }
    else {
      gen[1] = accum_2_RT;
      if(rctype == 1){
        gen[2] = 1;
      }
      else {
        gen[2] = 0;
      }
    }
    return(gen);
  }
}
data {
  int<lower = 0> N_obs;
  int<lower = 1> N_choices;
  int<lower = 1> n_u;
  int<lower = 1> n_w;
  int<lower =-1, upper = 1> rctype[N_obs];
  int<lower =-1, upper = 1> group[N_obs];
  int<lower = 1, upper = N_choices> winner[N_obs];
  vector<lower = 0>[N_obs] RT;

  int<lower = 1> N_subj;
  int<lower = 1> N_item;
  int<lower=1> subj[N_obs];
  int<lower=1> item[N_obs];
}
transformed data {
  real b; //arbitrary threshold
  real min_RT;
  b = 30;
  min_RT = min(RT);
}
parameters{
  vector[8] beta;
  real alpha[N_choices];

  real<lower=fmax(fabs(beta[7]),fabs(beta[8]))> sigma_e;

  cholesky_factor_corr[n_u] L_u;
  cholesky_factor_corr[n_w] L_w;
  vector<lower=0>[n_u] tau_u;
  vector<lower=0>[n_w] tau_w;
  vector[n_u] z_u[N_subj];
  vector[n_w] z_w[N_item];
}
transformed parameters {
  vector[n_u] u[N_subj];
  vector[n_w] w[N_item];
  {
    matrix[n_u,n_u] Sigma_u;
    matrix[n_w,n_w] Sigma_w;
    Sigma_u = diag_pre_multiply(tau_u,L_u);
    Sigma_w = diag_pre_multiply(tau_w,L_w);
    for(j in 1:N_subj)
      u[j] = Sigma_u * z_u[j];
    for(k in 1:N_item)
      w[k] = Sigma_w * z_w[k];
  }
}
model {
  alpha ~ normal(0,10);
  beta ~ normal(0,1);
  sigma_e ~ normal(0,2);
  tau_u ~ normal(0,1);
  tau_w ~ normal(0,1);

  //priors
  L_u ~ lkj_corr_cholesky(2.0);
  L_w ~ lkj_corr_cholesky(2.0);
  for (s in 1:N_subj)
    z_u[s] ~ normal(0,1);
  for (i in 1:N_item)
    z_w[i] ~ normal(0,1);




  for (n in 1:N_obs) {
    real accum_1_mu = b - (alpha[1] + u[subj[n],1] + w[item[n],1] + beta[1]*group[n] + rctype[n]*(beta[3]+u[subj[n],3]) + group[n]*rctype[n]*beta[5]);
    real accum_2_mu = b - (alpha[2] + u[subj[n],2] + w[item[n],2] + beta[2]*group[n] + rctype[n]*(beta[4]+u[subj[n],4]) + group[n]*rctype[n]*beta[6]);

    real accum_1_sig = sigma_e + group[n]*beta[7];
    real accum_2_sig = sigma_e + group[n]*beta[8];

    target += race(winner[n], RT[n], accum_1_mu, accum_1_sig, accum_2_mu, accum_2_sig);
  }
}



generated quantities {
  vector[N_obs] rt_1;
        vector[N_obs] rt_2;
  vector[N_obs] gen_acc;
  vector[N_obs] gen_rctype;
  vector[N_obs] gen_RT;
  vector[N_obs] gen_group;

  vector[N_obs] log_lik;

  for (n in 1:N_obs){
    vector[2] gen;
    real mu_1;
    real mu_2;
    real sig_1;
    real sig_2;

    mu_1 = b - (alpha[1] + u[subj[n],1] + w[item[n],1] + beta[1]*group[n] + rctype[n]*(beta[3]+u[subj[n],3]) + group[n]*rctype[n]*beta[5]);
    mu_2 = b - (alpha[2] + u[subj[n],2] + w[item[n],2] + beta[2]*group[n] + rctype[n]*(beta[4]+u[subj[n],4]) + group[n]*rctype[n]*beta[6]);

    sig_1 = sigma_e + group[n]*beta[7];
    sig_2 = sigma_e + group[n]*beta[8];

    gen = race_rng(mu_1, sig_1, mu_2, sig_2, rctype[n]);
    gen_RT[n] = gen[1];
    gen_acc[n] = gen[2];
    gen_rctype[n] = rctype[n];
    gen_group[n] = group[n];

    rt_1[n] = lognormal_rng(mu_1, sig_1)/1000;
            rt_2[n] = lognormal_rng(mu_2, sig_2)/1000;

    log_lik[n] = race(winner[n], RT[n], mu_1, sig_1, mu_2, sig_2);
  }
}
```

B Direct Access model Stan files

B.1 Simple model

```
functions {
  real direct_access(int accuracy, real RT, real theta, real P_b, real mu, real delta, real sigma){

    real p_answer_correct;
    real p_answer_correct_direct_access;
    real p_answer_correct_reanalysis;
    real p_answer_incorrect;

    // theta * (P_b * 1-theta)
    // - Combination of equation (20) and (21) in log
    p_answer_correct = log_sum_exp(log(theta), log(P_b) + log1m(theta));
    // theta / p_answer_correct
    // - Proportion initial correct, equation (20) in log
    p_answer_correct_direct_access = log(theta) - p_answer_correct;
    // (P_b * 1-theta) / p_answer_correct
    // - Proportion reanalysis, equation (21) in log
    p_answer_correct_reanalysis = log(P_b) + log1m(theta) - p_answer_correct;

    // (1-theta) * (1-P_b)
    // - Equation (22) in log
    p_answer_incorrect = log1m(P_b) + log1m(theta);

    if(accuracy==1) {
            //
      return (p_answer_correct +
            // Increment on likelihood due to RT:
            log_sum_exp(
              p_answer_correct_direct_access + lognormal_lpdf(RT| mu, sigma),
              p_answer_correct_reanalysis + lognormal_lpdf(RT| mu + delta, sigma) ));
    } else {
      return (p_answer_incorrect +
              lognormal_lpdf(RT| mu, sigma));
    }
  }
}
data {
  int<lower=1> N_obs;
  real RT[N_obs];
  int<lower=0,upper=1> accuracy[N_obs];
  int<lower=-1,upper=1> rctype[N_obs];
}

parameters {
  real mu; // meanlog
  real beta; // effect sentence type
  real<lower=0> sigma; // sdlog
  real alpha; // probability of reanalysis
  real<lower=0> delta; // effect of reanalysis
  real<lower=0,upper=1> P_b; // probability of backtracking
}

model {
  //priors
  alpha ~ normal(1,.5);
  beta ~ normal(0,2);
  delta ~ normal(0,2);
  mu ~ normal(10,2);
  sigma ~ normal(0,1);
        P_b ~ normal(0,1);

  // log likelihood
  for (i in 1:N_obs){
    // compute theta here
    real theta = inv_logit(alpha + rctype[i]*beta);
    target += direct_access(accuracy[i], RT[i], theta, P_b, mu, delta, sigma);
  }
}

generated quantities {
  real prob_sr;
  real prob_or;
  prob_or = inv_logit(alpha + beta);
  prob_sr = inv_logit(alpha - beta);
}
```

B.2 Hierarchical model

```
functions {
  real direct_access(int accuracy, real RT, real theta, real P_b, real mu,
                     real delta, real sigma_e){

    real p_answer_correct;
    real p_answer_correct_direct_access;
    real p_answer_correct_reanalysis;
    real p_answer_incorrect;

    // theta * (P_b * 1-theta)
    // -- Combination of equation (20) and (21) in log
    p_answer_correct = log_sum_exp(log(theta), log(P_b) + log1m(theta));
    // theta / p_answer_correct
    // -- Proportion initial correct, equation (20) in log
    p_answer_correct_direct_access = log(theta) - p_answer_correct;
    // (P_b * 1-theta) / p_answer_correct
    // -- Proportion reanalysis, equation (21) in log
    p_answer_correct_reanalysis = log(P_b) + log1m(theta) - p_answer_correct;

    // (1-theta) * (1-P_b)
    // -- Equation (22) in log
    p_answer_incorrect = log1m(P_b) + log1m(theta);

    if(accuracy==1) {
              // Increment on likelihood if accuracy=1
      return (p_answer_correct +
             // Increment on likelihood due to RT:
             log_sum_exp(
               p_answer_correct_direct_access + lognormal_lpdf(RT| mu, sigma_e),
               p_answer_correct_reanalysis + lognormal_lpdf(RT| mu + delta, sigma_e) ));
    } else {
      return (p_answer_incorrect +
               lognormal_lpdf(RT| mu, sigma_e));
    }
  }

  vector direct_access_rng(real theta, real P_b, real mu, real delta, real sigma_e){
    int init_acc;
    int backtrack;
    vector[2] gen;

    init_acc = bernoulli_rng(theta);
    backtrack = 0;
    if (init_acc!=1){
      backtrack = bernoulli_rng(P_b);
    }
    // Change the answer to 1 if there was backtracking:
    if(backtrack){
      gen[1] = lognormal_rng(mu, sigma_e);
      gen[2] = 1;
    }
    else {
      gen[1] = lognormal_rng(mu + delta, sigma_e);
      gen[2] = init_acc;
    }
    return(gen);
  }
}
data {
  int<lower=1> N_obs;
  real RT[N_obs];
  int<lower=0,upper=1> accuracy[N_obs];
  int<lower=-1,upper=1> rctype[N_obs];

  int<lower=1> N_subj;
  int<lower=1> N_item;
  int<lower=1> subj[N_obs];
  int<lower=1> item[N_obs];
}

parameters {
  real mu; //logmean
  real beta; //effect per sentence type
  real<lower=0> sigma_e; //logsd
  real alpha; //probability of reanalysis
  real<lower=0> delta; //effect of reanalysis
  real<lower=0,upper=1> P_b; //probability of backtracking (if a mistake has been made, similar across
      sentence types)

  cholesky_factor_corr[2] L_u;
  cholesky_factor_corr[2] L_w;
  vector<lower=0>[2] tau_u;
  vector<lower=0>[2] tau_w;
  vector[2] z_u[N_subj];
  vector[2] z_w[N_item];
}

transformed parameters {
  vector[2] u[N_subj];
  vector[2] w[N_item];
  {
    matrix[2,2] Sigma_u;
    matrix[2,2] Sigma_w;
    Sigma_u = diag_pre_multiply(tau_u,L_u);
    Sigma_w = diag_pre_multiply(tau_w,L_w);
    for(j in 1:N_subj)
      u[j] = Sigma_u * z_u[j];
    for(k in 1:N_item)
      w[k] = Sigma_w * z_w[k];
  }
}
```

```

model {
  //priors
  alpha ~ normal(1,.5);
  beta ~ normal(0,2);
  delta ~ normal(0,2);
  mu ~ normal(20,2);
  sigma_e ~ normal(0,1);
  tau_u ~ normal(0,1);
        tau_w ~ normal(0,1);
        P_b ~ normal(0,1);

  L_u ~ lkj_corr_cholesky(2.0);
  L_w ~ lkj_corr_cholesky(2.0);
  for (j in 1:N_subj)
    z_u[j] ~ normal(0,1);
  for (k in 1:N_item)
    z_w[k] ~ normal(0,1);

  // log likelihood
  for (i in 1:N_obs){
    real mu_adj = mu+ u[subj[i],1] + w[item[i],1];
    real theta = inv_logit(alpha + rctype[i]*beta + u[subj[i],2] + w[item[i],2]);
    target += direct_access(accuracy[i], RT[i], theta, P_b, mu_adj, delta, sigma_e);
  }
}

generated quantities {
  real prob_sr;
  real prob_or;

  vector[2] gen;
  vector[N_obs] gen_acc;
  vector[N_obs] gen_rctype;
  vector[N_obs] gen_RT;

  prob_or = inv_logit(alpha + beta);
  prob_sr = inv_logit(alpha - beta);

  for (i in 1:N_obs){
    real mu_adj = mu+ u[subj[i],1] + w[item[i],1];
    real theta = inv_logit(alpha + rctype[i]*beta + u[subj[i],2] + w[item[i],2]);
   // Generate data from the sampled parameters
    gen = direct_access_rng(theta, P_b, mu_adj, delta, sigma_e);
    gen_RT[i] = gen[1];
    gen_acc[i] = gen[2];
    gen_rctype[i] = rctype[i];
  }
}
```

B.3 Final model

```
functions {
  real direct_access(int accuracy, real RT, real theta, real P_b, real mu, real delta, real sigma_e){

    real p_answer_correct;
    real p_answer_correct_direct_access;
    real p_answer_correct_reanalysis;
    real p_answer_incorrect;

    // theta * (P_b * 1-theta)
    // -- Combination of equation (20) and (21) in log
    p_answer_correct = log_sum_exp(log(theta), log(P_b) + log1m(theta));
    // theta / p_answer_correct
    // -- Proportion initial correct, equation (20) in log
    p_answer_correct_direct_access = log(theta) - p_answer_correct;
    // (P_b * 1-theta) / p_answer_correct
    // -- Proportion reanalysis, equation (21) in log
    p_answer_correct_reanalysis = log(P_b) + log1m(theta) - p_answer_correct;

    // (1-theta) * (1-P_b)
    // -- Equation (22) in log
    p_answer_incorrect = log1m(P_b) + log1m(theta);

    if(accuracy==1) {
              // Increment on likelihood if accuracy=1
      return (p_answer_correct +
             // Increment on likelihood due to RT:
             log_sum_exp(
              p_answer_correct_direct_access + lognormal_lpdf(RT| mu, sigma_e),
              p_answer_correct_reanalysis + lognormal_lpdf(RT| mu + delta, sigma_e) ));
    } else {
      return (p_answer_incorrect +
              lognormal_lpdf(RT| mu, sigma_e));
    }
  }

  vector direct_access_rng(real theta, real P_b, real mu, real delta, real sigma_e){
    int init_acc;
    int backtrack;
    vector[2] gen;

    init_acc = bernoulli_rng(theta);
    backtrack = 0;
    if (init_acc!=1){
      backtrack = bernoulli_rng(P_b);
    }
    // Change the answer to 1 if there was backtracking:
    if(backtrack){
      gen[1] = lognormal_rng(mu, sigma_e);
      gen[2] = 1;
    }
    else {
      gen[1] = lognormal_rng(mu + delta, sigma_e);
      gen[2] = init_acc;
    }
    return(gen);
  }

}
data {
  int<lower=1> N_obs;
  real RT[N_obs];
  int<lower=0,upper=1> accuracy[N_obs];
  int<lower=-1,upper=1> rctype[N_obs];
  int<lower=-1,upper=1> group[N_obs];

  int<lower=1> N_subj;
  int<lower=1> N_item;
  int<lower=1> subj[N_obs];
  int<lower=1> item[N_obs];
}

parameters {
  vector[7] beta; //slopes per main effect
  real mu_0; //logmean
  real<lower=fabs(beta[5])> delta_0; //effect of reanalysis
  real gamma; //probability of backtracking (if a mistake has been made, similar across sentence types) in
      logit space
  real alpha; //probability of reanalysis in logit space
  real<lower=fabs(beta[7])> sigma_e_0; //logsd

  cholesky_factor_corr[2] L_u;
  cholesky_factor_corr[2] L_w;
  vector<lower=0>[2] tau_u;
  vector<lower=0>[2] tau_w;
  vector[2] z_u[N_subj];
  vector[2] z_w[N_item];
}

transformed parameters {
  vector[2] u[N_subj];
  vector[2] w[N_item];
  {
    matrix[2,2] Sigma_u;
    matrix[2,2] Sigma_w;
    Sigma_u = diag_pre_multiply(tau_u,L_u);
    Sigma_w = diag_pre_multiply(tau_w,L_w);
    for(j in 1:N_subj)
      u[j] = Sigma_u * z_u[j];
    for(k in 1:N_item)
      w[k] = Sigma_w * z_w[k];
  }
```

```
}

model {
  //priors
  alpha ~ normal(1,.4);
  beta ~ normal(0,2);
  delta_0 ~ normal(0,0.1);
  mu_0 ~ normal(20,4);
  sigma_e_0 ~ normal(0,1);
  tau_u ~ normal(0,1);
  tau_w ~ normal(0,1);
  gamma ~ normal(-2,1.5);

  L_u ~ lkj_corr_cholesky(2.0);
  L_w ~ lkj_corr_cholesky(2.0);
  for (j in 1:N_subj)
    z_u[j] ~ normal(0,1);
  for (k in 1:N_item)
    z_w[k] ~ normal(0,1);

  // log likelihood
  for (i in 1:N_obs){
    // Adjust parameters with random and fixed effects,
    real theta = inv_logit(alpha + rctype[i]*beta[1] + group[i]*beta[2] + rctype[i]*group[i]*beta[3] + u[
        subj[i],2] + w[item[i],2]);
    real mu = mu_0 + group[i]*beta[4] + u[subj[i],1] + w[item[i],1];
    real delta = delta_0 + group[i]*beta[5];
    real P_b = inv_logit(gamma + group[i]*beta[6]);
    real sigma_e = sigma_e_0 + group[i]*beta[7];

    target += direct_access(accuracy[i], RT[i], theta, P_b, mu, delta, sigma_e);
  }
}

generated quantities {
  real prob_or_i;
  real prob_or_c;
  real prob_sr_i;
  real prob_sr_c;

  real mu_i;
  real mu_c;

  real delta_i;
  real delta_c;

  real P_b_i;
  real P_b_c;

  real sigma_e_i;
  real sigma_e_c;
  vector[N_obs] log_lik;

  vector[2] gen;
  vector[N_obs] gen_acc;
  vector[N_obs] gen_rctype;
  vector[N_obs] gen_RT;
  vector[N_obs] gen_group;

  prob_or_i = inv_logit(alpha + beta[1] + beta[2] + beta[3]);
  prob_or_c = inv_logit(alpha + beta[1] - beta[2] - beta[3]);
  prob_sr_i = inv_logit(alpha - beta[1] + beta[2] - beta[3]);
  prob_sr_c = inv_logit(alpha - beta[1] - beta[2] + beta[3]);

  mu_i = mu_0 + beta[4];
  mu_c = mu_0 - beta[4];

  delta_i = delta_0 + beta[5];
  delta_c = delta_0 - beta[5];

  P_b_i = inv_logit(gamma + beta[6]);
  P_b_c = inv_logit(gamma - beta[6]);

  sigma_e_i = sigma_e_0 + beta[7];
  sigma_e_c = sigma_e_0 - beta[7];


  for (i in 1:N_obs){
    // Adjust parameters with random and fixed effects,
    real theta = inv_logit(alpha + rctype[i]*beta[1] + group[i]*beta[2] + rctype[i]*group[i]*beta[3] + u[subj[i],2] + w[item[i],2]);
    real mu = mu_0 + group[i]*beta[4] + u[subj[i],1] + w[item[i],1];
    real delta = delta_0 + group[i]*beta[5];
    real P_b = inv_logit(gamma + group[i]*beta[6]);
    real sigma_e = sigma_e_0 + group[i]*beta[7];
    // Add this for loo comparison later
    log_lik[i] = direct_access(accuracy[i], RT[i], theta, P_b, mu, delta, sigma_e);
    // Generate data from the sampled parameters
    gen = direct_access_rng(theta, P_b, mu, delta, sigma_e);
    gen_RT[i] = gen[1];
    gen_acc[i] = gen[2];
    gen_rctype[i] = rctype[i];
    gen_group[i] = group[i];
  }
}
```

References

Anderson, J. R. (1996). Act: A simple theory of complex cognition. *American Psychologist*, *51*(4), 355.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, *111*(4), 1036.

Betancourt, M. (2017). A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*.

Burkhardt, P., Piñango, M. M., & Wong, K. (2003). The role of the anterior left hemisphere in real-time sentence comprehension: Evidence from split intransitivity. *Brain and language*, *86*(1), 9–22.

Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, *134*(1-2), 57–83.

Caplan, D. (2012). Resource reduction accounts of syntactically based comprehension disorders. In *Perspectives on agrammatism* (pp. 48–62). Psychology Press.

Caplan, D., Michaud, J., & Hufford, R. (2015). Mechanisms underlying syntactic comprehension deficits in vascular aphasia: new evidence from self-paced listening. *Cognitive neuropsychology*, *32*(5), 283–313.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., ... Riddell, A. (2017, 1). Stan : A probabilistic programming language. *Journal of Statistical Software*, *76*(1). doi: 10.18637/jss.v076.i01

Connine, C., Ferreira, F., Jones, C., Clifton, C., & Frazier, L. (1984). Verb frame preferences: Descriptive norms. *Journal of Psycholinguistic Research*, *13*(4), 307–319.

Damasio, A. R. (1992). Aphasia. *New England Journal of Medicine*, *326*(8), 531–539.

Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, *195*(2), 216–222.

Explosion.ai dependency parser. (n.d.). https://explosion.ai/demos/displacy. (Accessed: 2018-12-12)

Furr, D. (2017). Rasch and two-parameter logistic item response models with latent regression.

Gelman, A., Rubin, D. B., et al. (1992). Inference from iterative simulation using multiple sequences. *Statistical science*, *7*(4), 457–472.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, *57*(1), 97–109.

Hoffman, M. D., & Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, *15*(1), 1593–1623.

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing* (Vol. 2). Pearson London.

Lewis, R. L., & Vasishth, S. (2005). An activation-based model of sentence processing as skilled memory retrieval. *Cognitive science*, *29*(3), 375–419.

Lewis, R. L., Vasishth, S., & Van Dyke, J. A. (2006). Computational principles of working memory in sentence comprehension. *Trends in cognitive sciences*, *10*(10), 447–454.

Lunn, D., Jackson, C., Best, N., Spiegelhalter, D., & Thomas, A. (2012). *The bugs book: A practical introduction to bayesian analysis*. Chapman and Hall/CRC.

Mätzig, P., Vasishth, S., Engelmann, F., Caplan, D., & Burchert, F. (2018). A computational investigation of sources of variability in sentence comprehension difficulty in aphasia. *Topics in cognitive science*, *10*(1), 161–174.

McElree, B. (1993). The locus of lexical preference effects in sentence comprehension: A time-course analysis. *Journal of Memory and Language*, *32*(4), 536.

McElree, B. (2000). Sentence comprehension is mediated by content-addressable memory structures. *Journal of psycholinguistic research*, *29*(2), 111–123.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*(6), 1087–1092.

Metropolis, N., & Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, *44*(247), 335–341.

Neal, R. M., et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, *2*(11), 2.

Nicenboim, B., & Vasishth, S. (2018). Models of retrieval in sentence comprehension: A computational evaluation using bayesian hierarchical modeling. *Journal of Memory and Language*, *99*, 1–34.

Olden, J. D., & Jackson, D. A. (2002). Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, *154*(1-2), 135–150.

Özesmi, S. L., & Özesmi, U. (1999). An artificial neural network approach to spatial habitat modelling with interspecific interaction. *Ecological modelling*, *116*(1), 15–31.

R Core Team. (2013). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from http://www.R-project.org/

Rouder, J. N., Province, J. M., Morey, R. D., Gomez, P., & Heathcote, A. (2015). The lognormal race: A cognitive-process model of choice and latency with desirable psychometric properties. *Psychometrika*, *80*(2), 491–513.

Russell, S. J., & Norvig, P. (1995). *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.

Shiffrin, R. M., Lee, M. D., Kim, W., & Wagenmakers, E.-J. (2008). A survey of model evaluation approaches with a tutorial on hierarchical bayesian methods. *Cognitive Science*, *32*(8), 1248–1284.

Sorensen, T., & Vasishth, S. (2015). Bayesian linear mixed models using stan: A tutorial for psychologists, linguists, and cognitive scientists. *arXiv preprint arXiv:1506.06201*.

Stan Development Team. (2018). *RStan: the R interface to Stan.* Retrieved from http://mc-stan.org/ (R package version 2.17.3)

Traxler, M. J., Morris, R. K., & Seely, R. E. (2002). Processing subject and object relative clauses: Evidence from eye movements. *Journal of Memory and Language*, *47*(1), 69–90.

Vehtari, A., Gelman, A., & Gabry, J. (2016). Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and Computing*, *27*(5), 1–20.

Wanner, E., & Maratsos, M. (1978). *An atn approach to comprehension. linguistic theory and psychological reality, ed. by morris halle, joan bresnan and george miller, 119–61.* Cambridge, MA: MIT Press.