Parameterization of connectionist models

Rafal Bogacz & Jonathan D. Cohen

Center for the Study of Brain, Mind and Behavior

Princeton University

Technical Report, no. 1

Abstract

This report presents a method for finding parameters of connectionist models that allows the behavior of the model to be fit as closely as possible to empirical data regarding the behavior of human subjects in psychological experiments. The method is based on minimization of a cost function that expresses how different the statistics describing behavior of the model are from the statistics of the subjects' performance in the experiment. An optimization algorithm is used to find the values of the parameters for which the value of the cost function is the smallest. The cost function also indicates whether the model's statistics are significantly different from those obtained in the experiment. In some cases, the method can find the required parameters automatically. In other cases it may help in and accelerate the process of manual parameterization. The method has been implemented in Matlab, is fully documented and is available for free download.

1. Introduction

Connectionist models are abstract models of how information processing occurs in the brain during performance of psychological tasks (Rumelhart, McClelland & the PDP Research Group, 1986). In these models, task performance is simulated as the flow of activity between sets of processing units that comprise processing pathways. Connectionist models have been successfully used to explain a variety of effects observed in psychological experiments (e.g. Rumelhart, McClelland & the PDP Research Group, 1986; Cohen et al., 1990; Cohen et al., 1992; Spencer & Coles, 1999; Usher & McClelland, 2001; Yeung et al. 2002; Holroyd and Coles, 2002). Furthermore, they provide a natural bridge between theories about cognitive processes and their implementation in the brain.

An example of a very simple connectionist network is shown in Figure 1. It is a model of the behavior of subjects in a two-choice experiment (e.g. subjects have to indicate whether a letter on the screen is A or B by pressing a corresponding button; simplified from Usher & McClelland, 2001). In psychological experiments, subjects repeat the task a large number of times, and various statistics can be computed describing subjects' performance, e.g. error rate, mean reaction time for correct and error trials, standard deviation of the reaction times, etc.. Similarly, the connectionist model of a task can be executed many times, and the same statistics can be calculated for the model. Such a repeated execution of the model in order to evaluate the statistic will be termed "run of the model" in the remainder of the article. The behavior of the connectionist model and thus the values of the model's statistics are controlled by a set of parameters. For example, the network in Figure 1 is controlled by 5 parameters: the weights of connections from correct and incorrect inputs, the weight of the inhibitory connection between the response units, a decision threshold, and the magnitude of noise.

Usually, in order to claim that a connectionist model can account for human behavior in a given task, researchers show that the statistics describing the model performance have the same or very similar values to those for subjects in the experiment (e.g. Usher & McClelland, 2001; Holroyd and Coles, 2002; Botvinick et al., 2001). However, in order to show this, it is necessary to find the values of model parameters that result in the

required behavior of the model. This parameterization of the model is usually done manually (i.e. researchers execute models with different sets of parameters and search for the set resulting in the closest match), and may be extremely time consuming. This may be due, in part, to the computational demands of wide-scale automated parameter search procedures for models that are, themselves, computationally demanding.

With continued improvements in computational power, some have begun to use automated procedures for parameterization of connectionist models. As yet, however, these have not typically been reported (to our knowledge, there are no published reports on universal tools for parameterization of connectionist models). Furthermore, these procedures may not be optimal; they may also be designed for a particular model and may not easily generalize to work for other models.

For example, Usher & McClelland (2001) automated parameterization of their two-choice model. However, the optimization algorithm they used was based on random search. Hence, although it was appropriate for the simple two-choice model, it is likely to be too slow for larger models (i.e. models of more complex tasks with larger number of parameters). Furthermore, the cost function they minimized was chosen arbitrarily for their model, thus may not generalize to other models.

Ratcliff et al. (1999) automated parameterization of their diffusion model of two-choice tasks. They used the Simplex optimization algorithm (Nedler & Mead, 1965) – a type of algorithm that we have also found to be useful for parameterization of connectionist models. Ratcliff et al. (1999) found that automatic parameterization was very useful: "… it was only when the parameter space could be automatically searched […] allowed the model to fit the different patterns of error reaction times" (Ratcliff et al., 1999, p. 268). This gives hope that automatic parameterization procedures would similarly allow the finding of closer fits of other psychological models to experimental data.

Although there are well-described methods for optimizing certain parameters of connectionist networks (such as the back-propagation algorithm for setting connection strengths – e.g. Rumerhalt et al., 1986), setting parameters for other parts of the model is more challenging for several reasons. First, the analytic expressions describing how the model's statistics depend on the parameters are not fully described, and can be very

complex (Brown & Holmes, 2001). For example, while back-propagation can be used to find connection weights that minimize error, it has not been specified how to do so for a particular mean or distribution of reaction times.

Furthermore, connectionist models often incorporate noise, so that the statistics calculated during two different runs of the model with the same set of parameters differ from each other. This demands the model be run multiple times for each set of parameters to be tested, as well as methods of comparing the statistics generated by one run with another.

This report describes an algorithm for parameterization of connectionist models that seeks to address the challenges outlined above. It can be used for any connectionist model implemented as a computer program. In some cases, the algorithm can find the required parameters automatically. In other cases it may help in and accelerate the process of manual parameterization. The algorithm employs a statistically motivated cost function, which indicates whether the model's statistics are significantly different from those obtained in the empirical experiment.

The algorithm has been implemented in Matlab and is free for download at: http://www.math.princeton.edu/~rbogacz/autofit. It requires a user only to write a short Matlab script that executes the model and returns a vector containing the model's statistics as outputs.

The algorithm is described in Section 2. Section 3 demonstrates how the method performs in the parameterization of a sample connectionist model. Section 4 discusses various issues related to the usage of the method. A brief user's manual for the program implementing the algorithm is contained in the Appendix.

2. Parameterization algorithm

The process of parameterization of connectionist models described in this report is based on minimization of a cost function that is sensitive to differences between the statistics describing the behavior of the model and those derived from empirical data concerning human subject performance. An optimization algorithm is used to find the values of the model parameters for which the value of the cost function is least.

The optimization procedure involves repeating the following steps:

- Run of the model with a current set of parameters
- Regression between the performance statistics of the model and the empirical experiment
- Computing the cost function
- Choosing a new set of parameters by the optimization algorithm

Upon completion of the parameterization session, the value of the cost function indicates whether the model's statistics are significantly different from those obtained in the experiment.

Here we describe the parameterization algorithm. Section 2.1 describes how units of time in the model are converted to the time units of the experiment, which must be done before the cost function can be defined. Section 2.2 introduces a general form of the cost function. Section 2.3 describes the motivation for our choice of a particular optimization algorithm, and Section 2.4 details the parameterization procedure.

2.1. Matching time units of model and experiment

Statistics describing the reaction time of the model are expressed in time units of the model (e.g. number of iterations of updating units' activations required for a decision unit to cross a given threshold), while in the experiment they are expressed in milliseconds. In some models the relation between model time and real time is an explicit assumption of the model (e.g. it is assumed that one step of the model corresponds to 50 ms; Anderson, 1993). However, in the majority of connectionist models such explicit assumptions are not made, and the reaction time statistics must be converted from model time units to milliseconds. Such conversion must be performed after each run of the model (i.e. repeated execution of the model in order to compute statistics) before the cost function may be evaluated, because the cost function includes terms expressing the difference between reaction times of the model and the experiment. In order to do so, they need to be in the same units. This conversion is described below.

It is usually assumed that a connectionist model does not capture all information processing during a task (e.g. it does not capture processing in the early visual pathway or in the motor pathway). Hence the relationship between the reaction time in model units RT^{units} and in milliseconds RT^{ms} can be expressed by the following equation:

$$RT^{ms} = a\, RT^{units} + b \tag{1}$$

In Equation 1, b denotes the duration of processing not captured by the model (in milliseconds; we assume for simplicity that it is constant on every trial) and a denotes how many millisecond correspond to one time unit of the model. Our goal is to find the values of a and b such that the differences between the model statistics converted to millisecond and the statistics describing human behavior in the psychological experiment are minimized. We can find such values of a and b by performing a standard linear regression.

Some statistics of the model may correspond to differences between reaction times in different experimental conditions or standard deviations of reaction times. The relationship between such statistics in model units V^{units} and in milliseconds V^{ms} can be expressed by the following equation:

$$V^{ms} = a\, V^{units} \tag{2}$$

In Equation 2, the same conversion factor a is used as in Equation 1 but there is no b term since the statistics are differences between reaction times and thus do not include the period of processing not captured by the model.

If the reaction time distribution is described by an ex-Gaussian function with mean μ and standard deviation σ of Gaussian and exponential time course τ, then the relation between the statistics μ in model units and in milliseconds is expressed by Equation 1, while the relation between the statistics σ and τ in model units and in milliseconds is expressed by Equation 2. For simplicity we will refer to all statistics whose relation of units is described by Equation 2 as statistics concerning variations of reaction times.

If we try to match also statistics concerning variations of reaction times between model and experiment, we choose the values of a and b by the following extended version of the regression. First, let us denote the statistics from the experiment concerning reaction

times by $RT^{exp}{}_i$, statistics concerning variations of reaction times by $V^{exp}{}_j$, and the numbers of these statistics being matched by N_{RT} and N_V respectively. We then choose the values of a and b that minimize the following function:

$$E = \sum_{i=1}^{N_{RT}} \left(a\, RT_i^{units} + b - RT_i^{\exp} \right)^2 + \sum_{j=1}^{N_V} \left(a V_j^{units} - V_j^{\exp} \right)^2 \tag{3}$$

In order to find the minimum for Equation 3 we have to find values of a and b for which derivatives of E are equal to 0:

$$\begin{cases} \dfrac{\partial E}{\partial a} = 0 \\ \dfrac{\partial E}{\partial b} = 0 \end{cases} \tag{4}$$

Solving the set of Equations 4 the following expressions are obtained:

$$\begin{cases} a = \dfrac{\sum_{i=1}^{N_{RT}} RT_i^{units} RT_i^{\exp} + \sum_{j=1}^{N_V} V_j^{units} V_j^{\exp} - \dfrac{1}{N_{RT}} \sum_{i=1}^{N_{RT}} RT_i^{units} \sum_{i=1}^{N_{RT}} RT_i^{\exp}}{\sum_{i=1}^{N_{RT}} \left(RT_i^{units} \right)^2 + \sum_{j=1}^{N_V} \left(V_j^{units} \right)^2 - \dfrac{1}{N_{RT}} \left(\sum_{i=1}^{N_{RT}} RT_i^{units} \right)^2} \\ b = \dfrac{\sum_{i=1}^{N_{RT}} RT_i^{\exp} - a \sum_{i=1}^{N_{RT}} RT_i^{units}}{N_{RT}} \end{cases} \tag{5}$$

Note that when we do not fit statistics concerning variations of reaction times (i.e. N_V = 0), Equations 5 reduces to the standard expression for regression coefficients.

Considering fits of the model to experimental data more generally, each statistic can be assigned to one of three types: 1. those that do not require the change of units described in this Section (e.g. error rates), 2. those being fit by regression with intercept (e.g. reaction times), 3 those being fit by regression without intercept (e.g. variations of reaction times). Let us denote the type of statistic i by c_i. After every run of the model, the regression coefficients are calculated according to Equation 5. If any of the coefficients is negative, it is made equal to 0 (negative values of a and b do not make sense). Then the model statistics of types 2 and 3 are converted to milliseconds according to Equations 1 and 2. After the conversion, the cost function may be calculated.

2.2. Cost function

Let us denote the statistics of the model by m_i and the statistics obtained in the experiment by e_i. Let us denote the number of statistics being fit by N. The cost function describes how different m_i are from e_i:

$$Cost = \sum_{i=1}^{N} \left(\frac{e_i - m_i}{n_i} \right)^2 \quad (6)$$

In Equation 6, n_i denotes the normalization factor for statistic i. It must be introduced to ensure that each statistic contributes equally to the cost function, because different statistics may have values with different orders of magnitude (e.g. error rate may be equal to 0.1 and reaction time to 400). A natural choice for the normalization factor is $n_i = e_i$, and such factors were used in the initial simulations described in the following Section 2.3. However, we found that a somewhat different set of values is superior, as described in Section 2.4.

2.3. Optimization algorithm

We compared a number of different optimization algorithms in simulations. Specifically, we used them to parameterize the simple two-choice model shown in Figure 1. The best performance was achieved by an algorithm called Subplex (Rowan, 1990).

Subplex is a modification of the simplex algorithm (Nedler & Mead, 1965). Subplex is specifically designed to minimize noisy functions, i.e. functions that may have different values when called two times with the same arguments. Hence the choice we made is well suited to connectionist models, as many connectionist models incorporate noise into processing.

In the Simplex algorithm, a simplex in n-dimensional parameter space is a set of n+1 points that bound part of the space. For example on a plane (i.e. 2-dimensional space), a simplex is a triangle. The simplex defines the region in which the algorithm looks for the minimum of the cost function. The Simplex algorithm starts by defining a simplex in the parameter space of a given size centered in a given point (i.e. the starting point of optimization). It evaluates the cost function in every point of the simplex (by running the

model). Then in the following steps, the set of parameters (i.e. a point of the simplex) which results in the highest value of the cost function is replaced by a new set of parameters and cost function for it is evaluated (for details see Nedler & Mead, 1965). The Subplex method (Rowan, 1990) is a generalization of the Simplex algorithm. It works by decomposing high-dimensioned problems into low-dimensioned sub-spaces which are easily handled by Simplex method (for details see Rowan, 1990).

In the remainder of this Section, the comparison of the performance of different optimization algorithms in parameterization of connectionist models is described, which motivated our choice of Subplex as our optimization algorithm. However, for readers not interested in this comparison, the rest of this Section may be skipped without loss in understanding of our parameterization procedure.

An important constraint for the optimization algorithm is that it must be fast, i.e. find the minimum of the cost function with as few runs of the model (i.e. sampling the parameter space) as possible. This feature is critical as one run of a complex connectionist model may take a significant period of time. Therefore, we did not test optimization algorithms that involve extensive random search such as genetic algorithms and simulated annealing.

The optimization algorithms were tested on parameterization of the two-choice model shown in Figure 1. We wanted the model to produce the following values of the statistics we chose arbitrarily: error rate = 20%, mean reaction time = 10 model units of time, standard deviation of reaction time = 2 model units of time. We specified the statistics concerning time in model units for simplicity, so the conversion of units described in Section 2.1 did not need to be performed for the immediate purposes.

We fixed the threshold parameter (see legend of Figure 1) at 1, and seek the value of 4 remaining parameters: w_{cor}, w_{inc}, w_{inh}, w_{noise} that would result in the values of the statistics specified in the previous paragraph (i.e. in the smallest value of the cost function from Equation 6 with $n_i = g_i$).

Each algorithm was tested during 100 optimization sessions (i.e. the optimization procedure was repeated 100 times). For each session, we chose the starting values of the parameters as random numbers from range [0, 0.2]. The starting points of optimization were the same for each algorithm, i.e. each algorithm started 100 sessions from the same

100 sets of random values of parameters. During each optimization session, the algorithms were allowed to execute 100 runs of the model (i.e. check 100 values of parameters). In the end of each optimization session we checked the value of the cost function for the solution found by the algorithm. For each algorithm we counted how many times during the 100 optimization sessions the algorithm found a set of parameters resulting in the value of the cost function below 0.1 and how many times below 0.02. The results for the five algorithms tested are presented in Figure 2.

First, we tested the standard Matlab built-in optimization functions: gradient based (fminunc) and simplex based (fminsearch, Lagarias, 1998). They seldom found parameters resulting in low values of the cost function. The reason for their poor performance was that they were not able to deal with the fact that the cost function is a noisy function. For example, the gradient based algorithm tried to calculate the gradient at the starting point of optimization numerically by running the model for the values of the parameters in different directions in the parameter space. However, this estimate of the gradient was imprecise as the cost function is noisy, and the optimization algorithms did not take it into account.

We tested three algorithms implemented in Matlab that are specifically designed to deal with noisy functions: Subplex (Rowan, 1990), implicit filtering (Kelley, 1999) and implementation of Simplex by Hooke-Jeeves (Kelley, 1999). Their performance is compared in Figure 2 – the best performance we observed was achieved by Subplex. We did not test all possible optimization algorithms, so it is possible that some other algorithms are more suitable for parameterization of connectionist models. We chose Subplex for further examination, because its performance was the best among the algorithms we tested.

We also assessed the impact of the choice of the optimization starting point. We evaluated the performance of the Subplex algorithm under conditions in which during each session the algorithm was allowed to run the model only 50 times, but the starting point of the optimization was chosen as follows. The model was run 50 times for 50 sets of parameters whose values were randomly chosen from range [0, 0.2]. For each set of parameters the cost function was evaluated, and the set of parameters resulting in the

lowest value of the cost function was chosen as the optimization starting point. The performance of Subplex with starting point chosen as described above is presented in the right-most bar of Figure 2. It is better than the performance of Subplex with random starting point. This suggests that it is worthwhile to spend some computational resources on searching for a good optimization starting point.

2.4. Parameterization procedure

This section describes in detail the parameterization procedure implemented in our tool. Each optimization session consists of three phases: finding a starting point of optimization, optimizing model parameters, and tuning the parameters.

1. Finding a starting point of optimization

The starting point of optimization may be either specified by the user or found during random search. In the second case, the user specifies the number of search iterations S, and values p_i for each parameter i which define the range of search. Namely, the model is run S times for S sets of parameters generated randomly such that each parameter i is chosen as a random number from range $[0, 2p_i]$. The set of parameters which resulted in the lowest value of the cost function is taken as the starting point of optimization.

During evaluations of the cost function the normalization factors of Equation 6 are different from $n_i = e_i$. If a simple normalization $n_i = e_i$ were used, then statistics that had values very close to zero (e.g. an error rate for a given condition may be smaller than 1%), could contribute to the cost function much more than others, because for them the denominator of Equation 6 would be very close to 0. To avoid this problem, during this phase, the normalization factors n_i of the cost functions are taken in the following way. For each type j of the statistics (defined in the last paragraph of Section 2.1) the average value E_j of the empirical statistics e_i belonging to that type is calculated, i.e. $E_j = \text{mean}\{e_i : c_i = j\}$. That is, the normalization factor for a given statistic is taken as the average goal statistics for its type, i.e. $n_i = E_{c_i}$. This choice was motivated by the fact that the statistics belonging to the same type usually have values relatively close to one another.

2. Optimization of parameters

Subplex searches for the set of parameters minimizing the cost function. The set of parameters found in the previous phase is taken as a starting point for Subplex. The initial size of the simplex in dimension i in the parameter space is taken as 30% of p_i (we found empirically that this value resulted in the highest performance of Subplex during parameterization of the model shown in Figure 1). The number of model runs executed by Subplex is specified by the user. The same normalization factors of the cost function are used as in phase 1.

3. Tuning of parameters

In the second phase, the optimization algorithm may find a set of parameters resulting in a quite close match between the statistics of the model and of the experiment. However, typically, these parameters can be further tuned which is attempted in phase 3.

Some statistics of the model may differ substantially from run to run with the same parameters, while other statistics may be more stable, i.e. have very similar values for different runs of the model with the same parameters. It may prove impractical to further optimize parameters to fit statistics which differ substantially from run to run with the same parameters, so it is better to focus on fitting the more stable statistics. Such emphasis can be achieved by setting the normalization factors n_i of the cost function to the standard deviations of the statistics of the model.

Hence, at the beginning of the tuning phase, the model is run 10 times for the parameters found in phase 2. For each of the statistics m_i of the model, the standard deviation across the 10 runs is calculated and it is taken as the normalization factor of this statistics n_i in the cost function during the optimization in this phase. The solution found in phase 2 is taken as a starting point of Subplex. The initial size of the simplex in dimension i in the parameter space is taken as 15% of p_i (value found empirically). The number of model runs allowed to be executed by Subplex in this phase is also specified by the user. The set of parameters found in this phase is the final solution. At the end of this phase the model is run 10 times for the final solution. The mean values of the model statistics and their standard deviations are calculated and used to compute the final value of the cost function.

The reason why normalization factors were not equal to the standard deviation in phase 2 is that during the optimization in second phase the solution (or the simplex) moves substantially in parameter space. Hence the cost function is being evaluated for very different values of parameters, so the model statistics may have very different values and standard deviations within phase 2.

This Section has described a single session of parameterization. Since the optimization process may find only a local minimum of the cost function, the optimization sessions may be repeated. The number of such repetitions is also specified by the user.

3. Case study

To verify how our parameterization procedure performs on more complex models, it has been used to parameterize a connectionist model of the Eriksen flankers task (Eriksen & Eriksen, 1974).

In one version of the Eriksen task, subjects are presented with one of the following visual stimuli: "<<<<<", ">>>>>", "<<><<", ">><>>", and have to indicate the direction of the middle arrow by pressing the left or the right button. Stimuli in which the direction of the middle arrow is the same as the "flanker" arrows are called compatible, and the other are called incompatible. Cohen et al. (1992) proposed a connectionist model that simulated many of the behavioral effects observed in this experiment. Recently, Yeung et al. (2002) conducted an empirical study using this task. We tried to fit the model to the values of the statistics describing the mean performance of subjects in that experiment using the procedures described above.

The architecture of the version of the Eriksen task model we used is shown in Figure 3 (simplified slightly from Cohen et al., 1992). The inputs to the model include three pairs of units. The middle pair provides information about the direction of the middle arrow in the stimulus. If on a given simulated trial the middle arrow is pointing left, then the activation of the left input in the pair is equal to 1 and of the right input is equal to 0. If on the trial the middle arrow is pointing right, then the activation of the right input is equal to 1. The two other pairs of inputs indicate the direction of the left and right "flanker" arrows in the stimulus in a similar way.

The model consists of two layers of units: a stimulus layer and a response layer. The model "makes a response" when the activity of one of the response units exceed a decision threshold. The middle units in the stimulus layer also receive input from a unit representing attentional bias (the unit with an exclamation mark in Figure 3), which is constantly equal to 1. The model has five parameters: *a* – the weight of excitatory connections, *b* – the attentional bias, *c* – magnitude of noise, *d* – weight of mutual inhibitory connections, *thr* – threshold of response units. Seven statistics describing performance of the model were fit to the experimental data; they are listed in the left column of Table 1. Note that since the number of parameters is fewer than the number of statistics being fit, there is not a guaranteed solution.

During a single run, the model was executed for 13056 trials, which is equal to the number of stimuli (across subjects) in the Yeung et al. (2002) experiment. The order of stimuli was also the same in the simulations as in the experiment. During each optimization session the model was run 50 times during the phase of searching for optimization starting point, 150 times during the optimization phase and 100 times during the tuning phase. The whole optimization session was repeated 200 times, which took about one day of computation using a Pentium II computer.

The value of the cost function for the best solution found by the parameterization algorithm was equal to 27.6. The values of the individual statistics are also compared in Table 1.

4. Discussion

This report describes an algorithm for parameterization of connectionist models and an example of its application to a particular model. The method can be used for parameterization of any connectionist model implemented as a computer program. The optimization procedure uses a statistically motivated cost function which indicates whether the statistics of the model differ significantly from those obtained in the experiment.

Since the algorithm may find a local minimum of the cost function, it is important to perform a suitable number of optimization sessions. For example, in the simulations

described in Section 3, the parameterization procedure found sets of parameters resulting in a low value of the cost function in only about 10% of the sessions.

We have noticed that the quality of the solution found by the algorithm strongly depends on the initial range of parameters p_i from which starting points of optimization are taken. If the method cannot find good solutions, it may mean that it is searching in the wrong part of the parameter space and getting stuck in local minima. Therefore, in such situations, it may be sometimes worthy to try to run the model with different parameters, and get some intuitions how the parameters effect the behavior of the model. This may help in choosing the appropriate range of parameters p_i from which starting points of optimization are taken.

The use of this optimization procedure may also indicate problems with the model and thus help in improving its design. For example, if during many sessions of parameterization with different starting points, particular statistics of the model are consistently different from the statistics in the experiment, it may indicate that a change in the architecture of the model is necessary.

When a run of a connectionist model takes a long time, then it may take many days for our algorithm to parameterize the model. In this case it is worth trying to speed the computer program implementing the model (e.g. if the model is implemented in Matlab, it could be translated into C). In order to find an approximation of parameters, one can modify the way the model is executed such that it reduces the execution time. For example, one can try to decrease the number of model executions in one run, or if the model is described by continuous differential equations, one can also try to make its discrete approximation coarser. Although this may reduce the precision of the estimate of the model's statistics, or make them noisier, the result may provide useful guidance in the choice of a starting point for optimization of the full but slowly executing model.

The cost function used in the algorithm has statistical meaning, and one may be tempted to use it to compare different models. For example, if two models of a given task are compared, and the parameters of the first model result in a lower value of the cost function than the second, one could claim that the first model fits the data better than the second. However, it should be remembered that the value of the cost function depends on

the number of simulated trials in a run, because the number of trials influences the normalization factors n_i of the cost function. Namely, n_i are equal to the standard deviations of the statistics across trials for a given set of parameters, and the more simulated trials, the more similar are the values of statistics across trials. Hence, if the cost function is used to compare the models, it must be insured that the models were executed with the same number of simulated trials per run.

Finally, in the simulations described in Section 3 we fit the model to mean experimental statistics averaged across subjects. But one could also use the algorithm to fit the model to behavior of each of the subjects, and then explore individual differences between subjects using the model. Of course, these methods can be also used to fit neurophysiological statistics such as the amplitude or latency of ERPs or hemodynamic response measurements (such as fMRI).

Appendix. User manual

The parameterization algorithm has been implemented in Matlab and can be downloaded from http://www.math.princeton.edu/~rbogacz/autofit

The compressed file available to download includes a number of Matlab files. The main function is "fitparam". The first part of the Appendix describes the usage of this function and the second part shows an example how the function is called.

The usage of fitparam function

The function has the following format:

function [bestpar, bestval, bestat, step, delay, p] = fitparam (model, startpar, goalstat, typestat, randiter, optiter, tuneiter, nosession, statweight, parange)

The following inputs must be specified:

model - string containing a name of Matlab function implementing the connectionist model. This function should get as input the vector of parameters and return the values of statistics, i.e. function statistics = model (parameters)

startpar - starting values of parameters (p_i in Section 2)

goalstat - the values of the statistics obtained in the experiment (e_i in Section 2)

typestat - vector of the same length as goalstat describing types of the statistics (c_i in Section 2), for each statistics, c_i should have value 1, 2 or 3 as described in Section 2.

The following inputs are optional:

randiter - when = 0, the provided parameters p_i are used as starting point for optimization; when > 0, number of runs during random search for starting point (number of runs in phase 1 described in Section 2.4) ; default = 0

optiter - number of runs during optimization (i.e. phase 2); default = 70

tuneiter - number of runs during tuning (i.e. phase 3); default = 50

nosession - number of session of repetition of the whole process; default = 1

One needs to be careful with changing these parameters - they should depend on the speed of the function model, since the model will be called: nosession * (randiter + optiter + tuneiter + 20) times

statweight - vector describing how much emphasis should be made on each statistic during optimization; default: all the statistics are equally emphasized.

parange - allowed range of parameters (it should be specified if a user does not want the parameters to take values outside a particular range), matrix containing 2 columns with length equal to the number of parameters. The first column contains minimal, the second maximal values of the parameters; default: no constraints.

The function returns the following outputs:

bestpar - best sets of parameters found during each session

bestval - error for the corresponding set of parameters

bestat - values of the statistics for the corresponding set of parameters

step, delay - slopes and intercepts of reaction time regression (i.e. parameters a and b in Section 2.1)

p - the significance of the difference between experimental and model statistics.

Since the optimization may take long time, at the end of every session the outputs are written to a Matlab data file: bestmy.mat.

Example of usage

The package downloadable includes a function "twodecision" which implements a simplified version of the model from Figure 1, that does not have a mutual inhibition between the decision units (i.e. $w_{inh} = 0$).

The model (and function twodecision) has three parameters: param(1) = w_{cor}, param(2) = w_{inc} , param(3) = w_{inh}. Function twodecision runs model 1000 times and calculates the following statistics: error rate, mean reaction time for correct, standard deviation of reaction times for correct, mean reaction time for incorrect, standard deviation of reaction times for incorrect. Assume that we want to fit the following values of the above statistics respectively: 10%, 280ms, 70ms, 300ms, 70ms. Let us start the optimization from all the parameters equal to 0.1.

To run just a single optimization session, one has to type:

[par, val, stat] = fitparam ('twodecision', [0.1 0.1 0.1], [0.1 280 70 300 70], [1 2 3 2 3])

To do an extensive search with 100 sessions, one has to type:

[par, val, stat] = fitparam ('twodecision', [0.1 0.1 0.1], [0.1 280 70 300 70], [1 2 3 2 3], 50, 70, 50, 100)

If a user wants to fit error rate (i.e. the first statistic) with particularly high precision, one has to increase its weight by typing:

[par, val, stat] = fitparam ('twodecision', [0.1 0.1 0.1], [0.1 280 70 300 70], [1 2 3 2 3], 50, 70, 50, 100, [10, 1, 1, 1, 1])

If a user wants the first parameter to be larger than 0.05 and smaller than 0.12 in all the solutions found and two other parameters to be in the range from 0 to 1, then one has to type:

[par, val, stat] = fitparam ('twodecision', [0.1 0.1 0.1], [0.1 280 70 300 70], [1 2 3 2 3], 50, 70, 50, 100, [1, 1, 1, 1, 1], [0.05, 0.12; 0, 1; 0, 1])

Acknowledgements

This research was supported by a grant from the National Institutes of Health (P50-MH62196). The authors are grateful to Afsheen Afshar for discussion, Sander Nieuwenhuis for testing the tool and useful comments, and to Peter Hu for reading the manuscript and useful comments.

References

Anderson, J.R. (1993). Rules of the Mind. Hillsdale, NJ: Erlbaum.

Botvinick, M.M., Braver, T.S., Barch, D.M., Carter, C.S., & Cohen, J.D. (2001). Evaluating the demands for control: anterior cingulate cortex and conflict monitoring. Psychological Review, 108, 624-652.

Brown, E., & Holmes, P. (2001). Modeling a simple choice task: stochastic dynamics of mutually inhibitory neural groups. Stochastics and Dynamics, 1, 159-191.

Cohen, J.D., Dunbar, K., & McClelland, J.L. (1990). On the control of automatic processes. Psychological Review, 97, 332-361.

Cohen, J.D., Servan-Schreiber, D., & McClelland, J.L. (1992). A parallel distributed processing approach to automaticity. American Journal of Psychology, 105, 239-269.

Eriksen, B.A., & Eriksen, C.W. (1974). Effects of noise letters upon the identification of target letters in a non-search task. Perception and Psychophysics, 16, 143-149.

Holroyd, C.B., & Coles, G.H. (in press). The neural basis of human error processing: reinforcement learning, dopamine and the Error related negativity.

Kelley, C.T. (1999). Iterative methods for optimization. SIAM Frontiers in Applied Mathematics, 18.

Lagarias, J.C., Reeds, J.A., Wright, M.H., & Wright, P.E. (1998). Convergence properties of the Nedler-Mead simplex algorithm in low dimensions. SIAM Journal of Optimization, 9, 112-147.

Nedler, J.A., & Mead, R. (1965). A simple method for function minimization. Computer Journal, 7, 308-313.

Ratcliff, R., Van Zandt, T., & McKoon, G. (1999). Connectionist and diffusion models of reaction time. Psychological Review, 106, 261-300.

Rowan, T. (1990). Functional Stability Analysis of Numerical Algorithms. PhD thesis, University of Texas at Austin.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning representations by back-propagating errors. Nature, 323, 533-536.

Rumelhart, D.L., McClelland, J.L., & the PDP Research Group (1986). Parallel Distributed Processing: Explorations in the microstructure of cognition. Cambridge, MA: MIT Press.

Spencer, K.M., & Coles, M.G.H. (1999). The lateralized readiness potential: Relationship between human data and activation in a connectionist model. Psychophysiology, 36, 364-370.

Usher, M., & McClelland, J.L. (2001). On the time course of perceptual choice: the leaky competing accumulator model. Psychological Review, 108, 550-592.

Yeung, N., Botvinick, M.M., & Cohen, J.D. (in press). The neural basis of error detection: conflict monitoring and the error-related negativity.

Figure 1. Example of a simple connectionist model of two-choice task (simplified from Usher & McClelland, 2001). Panel a) shows the architecture of the model. Arrows denote excitatory connections, line with circles denotes inhibitory connections. During model execution x_A is equal to 1 if stimulus A is presented, otherwise it is equal to 0; x_B – analogously. At the beginning of each simulated trial y_A and y_B are set up to 0. Then they are iteratively modified according to equations in panel b); η denote independent sources of Gaussian noise. Model "makes a response" when y_A or y_B exceeds a threshold (another parameter of the model).

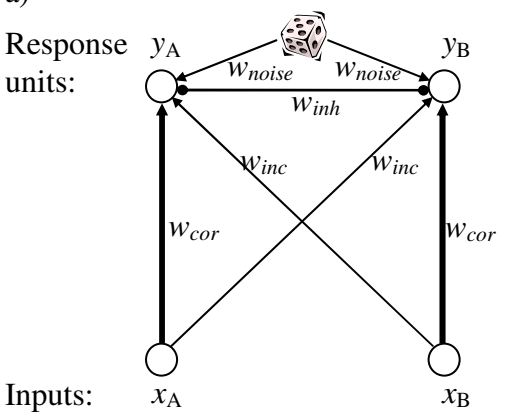
Figure 2. Comparison of the performance of various optimization algorithms in the parameterization of the two-choice model from Figure 1. The dark bars denote how many times during 100 optimization sessions particular algorithm found a set of parameters resulting in a cost function below 0.1, and the bright bars – how many times below 0.02.

Figure 3. Simplified version of the Eriksen task (Eriksen & Eriksen, 1974) model (simplified from Cohen et al., 1992). Arrows denote excitatory connections, arches with circles at the end indicated that all the units in a given layer mutually inhibit one another.

Table 1. Comparison of the values of the statistics describing the behavior of subjects in Yeung et al's (2002) experiment and the simplified connectionist model for the best solution found by the parameterization algorithm.

Figure 1

a)



b)

$$\Delta y_A = w_{cor}\, x_A + w_{inc}\, x_B - w_{inh}\, y_B + w_{noise}\, \eta$$

$$\Delta y_B = w_{cor}\, x_B + w_{inc}\, x_A - w_{inh}\, y_A + w_{noise}\, \eta$$

Figure 2

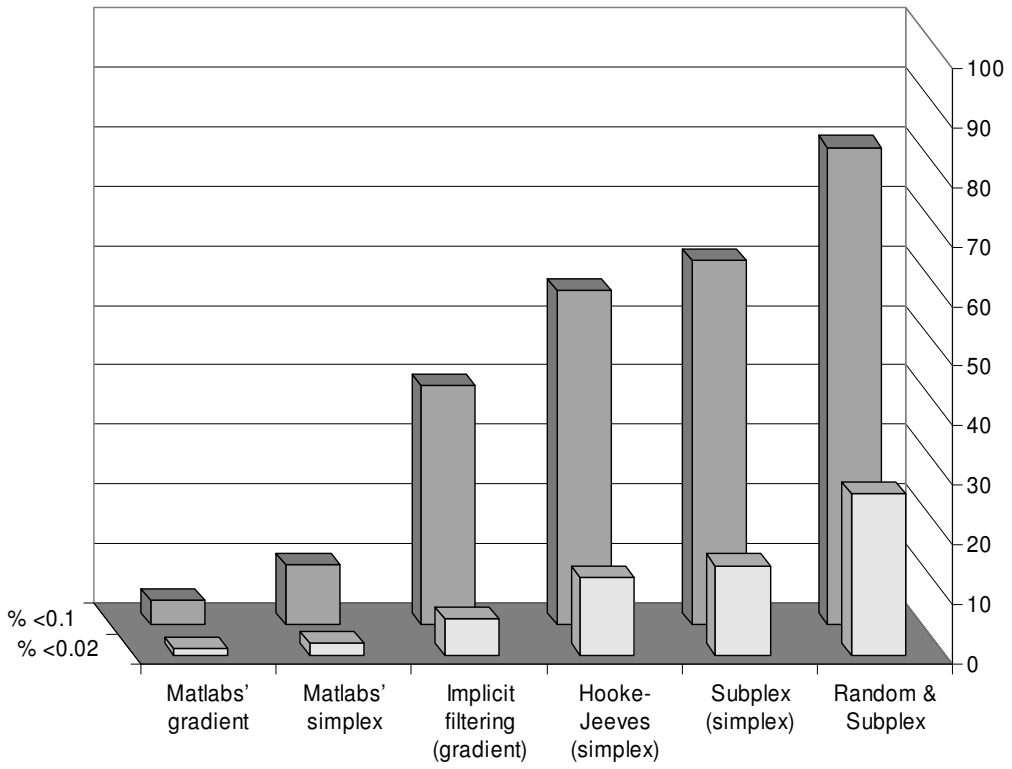

100
90
80
70
60
50
40
30
20
10
0
% <0.1
% <0.02
Matlabs' gradient
Matlabs' simplex
Implicit filtering (gradient)
Hooke-Jeeves (simplex)
Subplex (simplex)
Random & Subplex

Figure 3

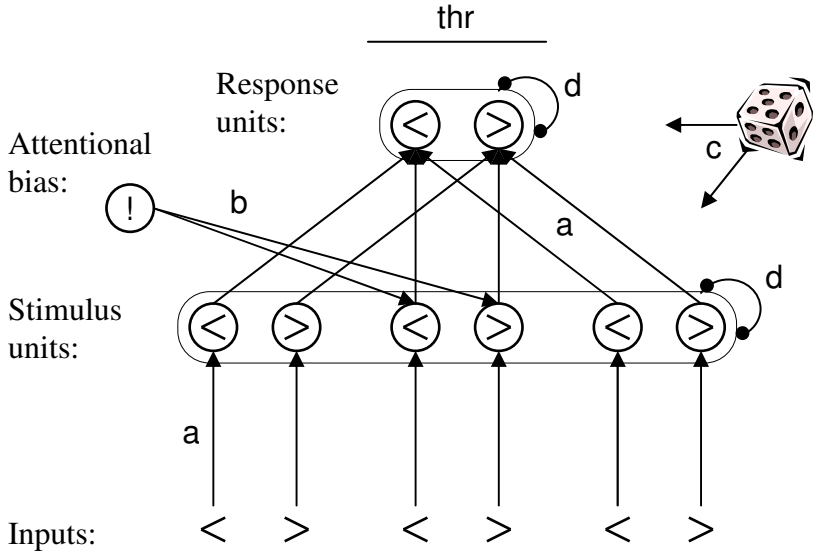

thr
Response units:
d
Attentional bias:
!
b
a
c
d
Stimulus units:
a
Inputs:
< > < > < >

Table 1

Statistics	Experiment	Model
Error rate for compatible stimuli	1.9%	1.3%
Error rate for incompatible stimuli	18.7%	19.1%
Mean reaction time for compatible stimuli	351.3	352.2
Mean reaction time for incompatible stimuli	403.2	402.3
Difference between mean reaction times for correct and incorrect responses	67.4	66.5
Standard deviation of reaction times for compatible stimuli	73.9	77.3
Standard deviation of reaction times for incompatible stimuli	100.9	99.3