U-Learn: *User Manual*

For a brief introduction to the theoretical underpinnings:
Perruchet, P., Robinet, V. & Lemaire, B. (submitted). U-Learn : Finding optimal coding units from an unsegmented sequential database
Last update: November 4, 2012
Please contact Pierre Perruchet, pierre.perruchet@u-bourgogne.fr, for any questions or problems.

Introduction

Let us consider the sequence:
ABCFGDEABCDEABCFGDEFGABCDEFG
This sequence is composed from the random concatenation of 3 units: ABC, DE, and FG. U-learn is aimed at (1) generating sequences of various levels of complexity, from the random concatenation of a few units of equal frequency, as above, to very complex sequences reflecting much better the variability of real-world situations, and (2) to test the ability of different computational models to find the units composing these sequences.

In the example above, the letters stand for any element that may be considered as a (undividable) primitive for a given subject at a given moment: a phoneme, a grapheme, a syllable, a note of music, a spatial location, and so on. However, given the focus of the literature on artificial languages since the seminal studies of Saffran et al. (1996), the terminology used below is borrowed to this research domain. As a consequence, the syllables are taken as primitives, the words as the relevant units, and the whole corpus may be composed of one or several sentences, each sentence comprising a variable number of words.

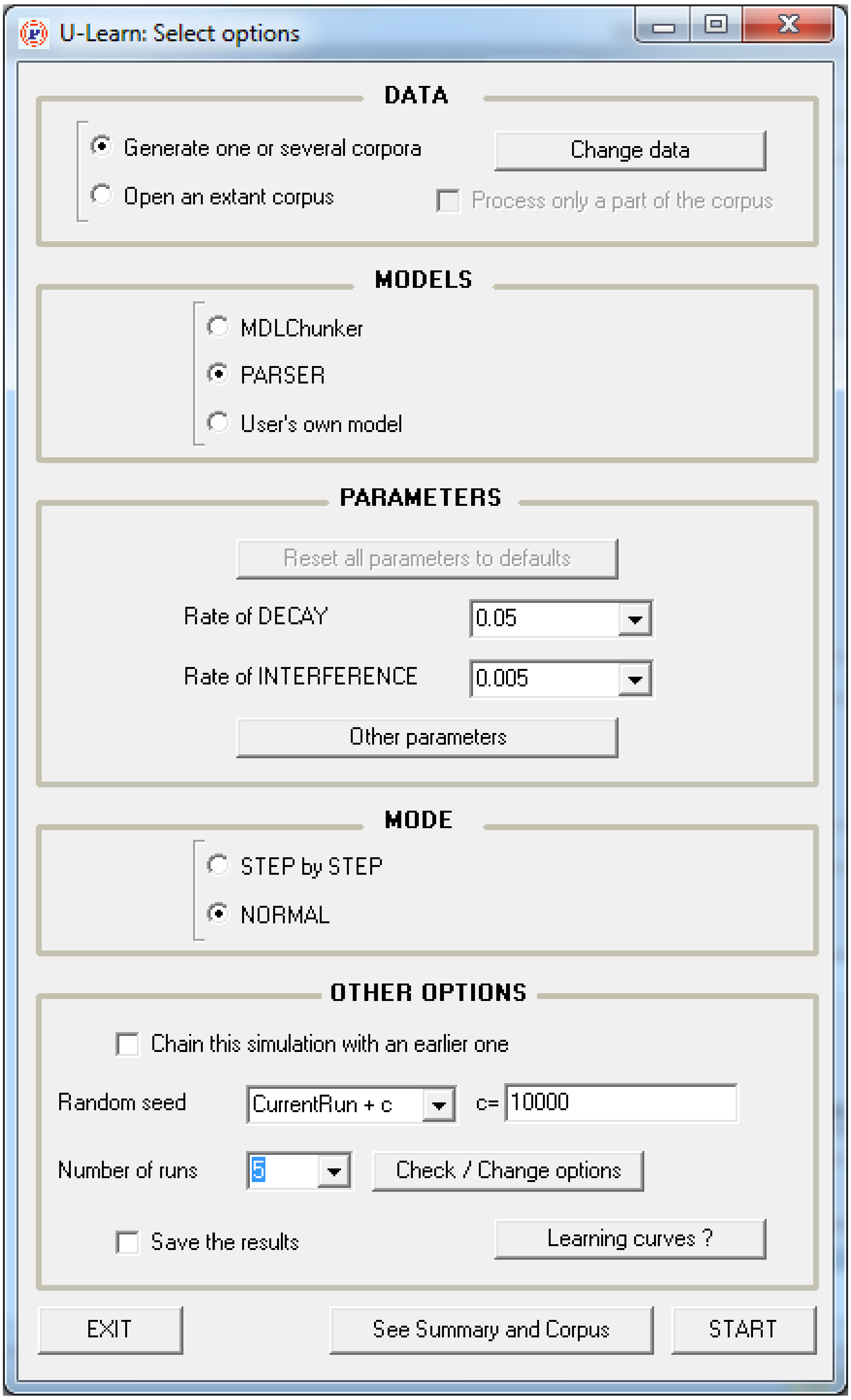
U-Learn is currently composed of 3 files, which may be freely downloaded at http://leadserv.u-bourgogne.fr/~perruchet/ The program should work with any version of Windows, at least from Windows XP to Windows7 64-bit.

1- U-Learn.exe. This is the main interface, and the only program that the user has to launch.
2- Parser.exe
3- MDLCh.exe.

Programs #2 and #3 are modules, with each module implementing a different model. The current models are Parser (Perruchet & Vinter, 1998) and the MDLChunker (Robinet, Lemaire, & Gordon, 2011), but the conception of the program makes it easy to add other models (see Appendix 2: Expanding U-Learn with a new model). It is advisable, although not mandatory, to *locate all three files in the same folder*. If the program doesn't find the appropriate .exe file, the user is required to indicate the path of this file through a standard Windows dialog box.

For a first appraisal

To have a quick overview of the program, start '*U-Learn.exe*'. The window that appears on the screen is shown below, except that several parts of this window are surrounded with a red line when the program starts. In U-Learn, the events circled with a red line (they may be a whole window, an option, a button box, etc.) are those that require an action from the user before the program can go ahead (which means: before the "*START*" or "*OK*" button is made active).



Three actions are required on the main window.

In the upper option box, click on '*generate one or several corpora*'. This opens a new window that is designed to enter the materials required to generate a corpus. Select one of the *Ready-to-use configurations* (the illustrations below display the data for Saffran et al. 1996) and the program will fill the form for you. The left-hand panels comprise the items that will be concatenated to build the corpus with their respective frequency. The other (right-hand) panels comprise the items used for the test.

After an example has been selected, the next choice is relative to the model, MDLChunker or Parser (the illustrations below display the data for Parser). The last mandatory option is relative to the Mode.

‘*Step-by-step*’ provides a detailed analysis on a single run, while ‘*normal*’ only provides the final state of the system (the illustrations below display the data for ‘*normal*’).

As indicated by the fact that all red lines have now disappeared, all the other options can be left at their default values. You may ask to see a summary of the current set-up and the corpus that the program has generated by clicking on the appropriate button. Clicking on ‘*START*’ opens the results window.

The results window comprises two main frames, the use of which substantially differs as a function of the selected mode (‘*Step-by-step*’ vs. ‘*normal*’), as detailed below. Learning curves are also displayed. Going back and forth from the ‘*Select options*’ to the ‘*results*’ windows allows to explore the main possibilities of the program. Note that by default, all user’s input and selections (i.e., the entered items and all options) are maintained until they are changed by subsequent actions. It is possible to change individual values, and to re-initialize all the values at once (‘clear *all*’ button in the ‘*enter the items’* window, and “*Reset all parameters to defaults*” button on the ‘*Select options’* window).

Which options are active at a given moment depends on earlier choices. Most options are inactive until a model (*MDLChunker* or *Parser*) and a mode (*Step-by-step* vs. *Normal*) have been set. The *Normal* mode makes more options available than the *Step by Step* mode. The most useful may be the *number of runs*. Entering any value >1, say 5, in the “*Number of runs*” combo box, allows to select a couple of options regarding the mode of processing of successive runs. Click on "*START*" and let the results scroll up on the screen. Standard media buttons allows to go ahead either run-by-run, or in a running way (note that the program waits for a few seconds after Run 1, to make it possible to stop on this run). The right-hand panel of the results window displays a summary of the results for the 5 runs. The program is described in more details below.

1 - The *Generate* vs. *Open a file* option

There are two options to enter the data: either (1) the basic components are provided to the program (i.e., at a minimum, a list of words and the number of occurrences for each word) via the keyboard, by copying/pasting the list from Word or Excel, or still by selecting a "*ready to use configuration*") and the program will generate one or several corpora, or (2) a ready-to-use corpus that has been previously created and saved as a text file (with this or another software) is loaded.

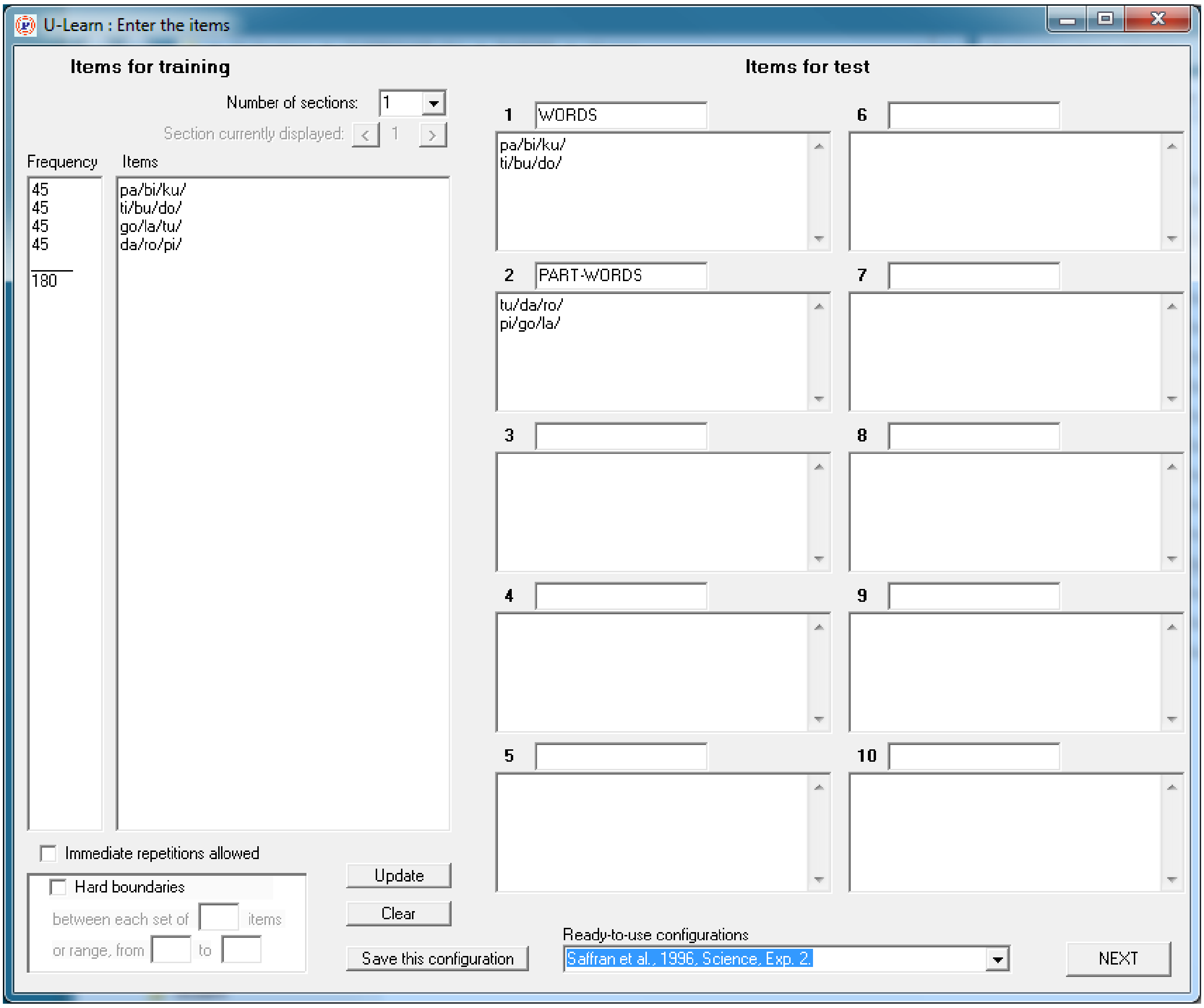
Data coding in U-Learn: General principles

To process a string of characters, the program needs to know the boundaries of the primitives, i.e., of the set of characters that are considered as indivisible units. In U-Learn, ‘/’ serves as a separator between primitives For instance, if syllables are considered as primitives for a given analysis, the word *baby* must be written "/*ba*/*by*/). The number of characters of a primitive is not limited, but using long primitives slows down the execution.

In many cases, the primitives can be coded with a single letter or digit, and this coding ensures maximal speed. For convenience, a general convention is that if there is no slash in a string (e.g., in the whole corpus, or in a test item), then the individual characters composing the string are taken as primitives.
In a nutshell: NO SLASH = SLASHES ANYWHERE. The user may easily check whether the data have been correctly coded by using the *See Summary and Corpus* option (see below).

Generate one or several corpora

One needs to enter the ***items for training*** in the left panel of the window below. The corpus can be divided into several (up to 10) sections. This may allow to investigate the influence of learning a first language on a second language, to vary the relative frequency of certain words along training, to introduce some words progressively during training, and so on. The items and all the parameters can be changed from one section to the next. Because the procedure for entering multiple sections is fairly obvious, and because most studies in the literature use a single, homogeneous corpus, the description below is limited to the case where a single, homogeneous language is created.



An item may comprise a variable number of primitives, and a primitive may comprise a variable number of characters. E.g., '1/', '1/6/3/', 'b/u/p/a/d/a', 'bu/pa/da/..' , 'par/ti/ci/pant/' are legal items.

The frequency of each item must be entered in the leftmost column. If the value is common to all items (or at least to a subset of successive items), it suffices to enter the target value at the top of the column (or at the top of the subset of items sharing the same frequency). Clicking on '*update*' completes the list automatically (and in addition indicates the total number of items). Updating is optional, but ensures that the frequencies have been set appropriately. As you may observe if you scan through the *ready-to-use examples*, the number of repetitions is constant in most studies, but it differs in others.

As a default, the items will be concatenated randomly without immediate repetition to form a continuous corpus. However, it is possible to authorize *immediate repetitions* by checking the appropriate box. As you may see in the *ready-to-use configurations*, some studies prohibited immediate repetitions while others allowed them. When no immediate repetition is allowed and the frequency of the items differs, the usual algorithms of randomization provide a flawed outcome (see French & Perruchet, Behavior Research & Method, 2009). U-Learn uses an algorithm derived from the one proposed by French & Perruchet, which ensures that each item occurs exactly the number of times that has been required, with an homogeneous distribution throughout the corpus.

It is also possible to generate the corpus as a succession of separate sentences, hence making language exposure a bit more natural. To do that, check the box '*Hard boundaries'*, and complete the edit boxes with either a fixed value (i.e. the number of words that will be comprised within two hard boundaries), or a range of values. The units created by the program cannot jump over a hard boundary.

Entering the ***items for test*** (in the small right-hand boxes) is optional. When test lists are provided, the program returns various scores (see below, "*analyze the results*" section) and learning curves. It is possible to enter ten different sets of test items, each set being composed of one or several items. In many cases, the relevant comparison is between words and part-words, so only two lists have to be completed, as in the example above. However, more detailed information is often useful. Consider for instance the study by Perruchet & Vinter (1998) in the *Ready-to-use configurations*. The question here is whether Parser is able to learn words of different length when they are mixed in a given language. Accordingly, the lists are composed of words of 1, 2, ...,5 syllables. Each list can be designated by a label; if no label is provided, the list will be designated in the result sheet by its number (from 1 to 10) and the first item of the list. Note that entering non-words (i.e., a sequence of syllables never displayed in the corpus) is objectless: Chunk-based models cannot create non-words.

The test items must be in the same format as the language, i.e. written with a '/' after each primitive (but see the foreword above). The items are usually only a few, so entering the data via the keyboard for each set of simulations should be a manageable task. However, there are two other possibilities. First, the items can be copied/pasted from a word processor or a worksheet in which they have been previously saved. A second possibility consists in loading a configuration that has been previously saved thanks to the "*save this configuration*" button. Click on the first option ("*load a previously saved configuration*") in the *ready-to-use configurations* combo box.

Although updating is optional, it is recommended to click on '*update*' before clicking on '*NEXT*', to ensure that the data have been correctly entered.

Open an extant corpus

Although the '*Generate*' option allows to create a large diversity of corpuses, there are also obvious limits. For instance, one may hope to analyze an artificial language following some syntactic rules, or a part of natural language (e.g., child-directed languages). This is not possible under the '*Generate*' mode (the only available syntactic constraint is the prohibition of immediate repetition...).

Before selecting the '*Open an extant corpus*' option, you have to prepare a text file containing the entire corpus. The text must be segmented into primitives, which are separated by '/' (optional if all the primitives are single characters). The program also needs to know whether the corpus can be considered as a continuous sequence of primitives, or if there are hard boundaries. Hard boundaries separate physically discontinuous utterances – No unit straddling over a hard boundary will be created. In U-Learn, the hard boundaries are coded with '//'.

Paragraph marks, spaces, and the following punctuation symbols: . , ! and ? can be included for user's convenience, but they have no function at all: only '/' and '//' are recognized as separators. Any other character (more precisely: any character the ASCII code is comprised between 34 and 255, with the exception of the punctuation marks listed above, and, of course, '/') is coded as an element of a primitive.

For example:

(1) this/is/the/first/sen/tence//this/is/the/se/cond/sen/tence/
(2) this / is / the / first / sen/tence,/
/this / is / the / se/cond / sen/tence/.

and

(3) this/
is/
the/
first/
sen/
tence//
this/
....

are equivalent: in all cases, syllables are primitives and the two sentences are separate utterances.

Given that the '/' is optional if individual characters are taken as primitives,

(1) t/h/i/s/i/s/

and

(2) this is

are also equivalent (provided there is no "/' in the whole string in which this excerpt is embedded. Note that in these two examples, the data will be automatically recoded under the format displayed in (1) before being sent for analyzes. The recoded data can be seen with the *see summary and corpus* option.

By contrast:

(1) this/

and

(2) This/

are different primitives, because ‘t’ and ‘T’ are coded as different characters. Although this choice appears to be rather inappropriate in this specific case, lower-case and upper-case letters are considered as different characters due to their distinctive function in the phonetic code.

> It is worth stressing that coding any existing text files for U-Learn may just require a few back-and-forth with a word processor. Let us suppose that you have a child-directed database coded as a sequence of phonemes, and that you wish to use phonemes as primitives. You may first replace any punctuation marks (at least the dots) with "//" under the word processor (given that boundaries between sentences have high chance to be perceptually salient), and save the corpus as a text file. Then you may load the text with U-Learn (don't forget to close the file under the word processor before this operation), and U-Learn will insert a "/" after each character (e.g., *babe* will be rewritten *b/a/b/e/*). In a last step, you may return to the word processor to remove the slash between phoneme codes involving two characters (e.g., replace all /o/~/ with /o~/)

After having loaded a file, the user is offered the possibility of entering the words of the language and/or test words. This information is obviously ignored by the program during the extraction process, but, if provided, it is exploited for analyzing the results and drawing learning curves. The procedure is the same as the one described above (‘*Generate one or several corpora*’), except that a few irrelevant options (e.g., the frequency of items) are made inactive.

Process only a part of the corpus

(Available only when the corpus has been loaded from a file –The number of repetitions is a parameter in the ‘*Generate*’ mode). If you have prepared a file with a long corpus, and that you wonder about the model’s performance with a smaller corpus, you don’t need to prepare and save a new file. When the ‘*process only a part of the corpus*’ box has been checked, you are asked how many primitives you wish to keep for the next analysis.

2 - Selecting a Model

If ' MDLChunker ' has been selected, the subprogram 'MDLCh.exe' is expected to be in the same folder as the main program, and likewise for the other models. If U-Learn doesn't find the appropriate .exe file, you will be required to indicate the path of this file through a standard Windows dialog box.

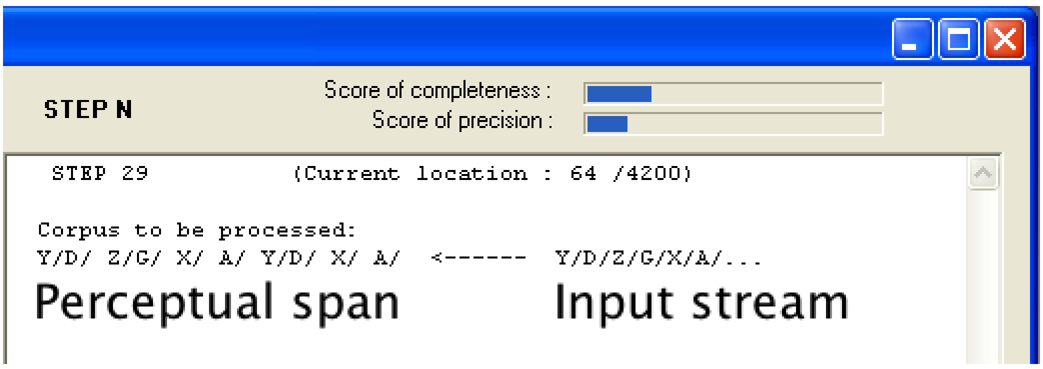
3 - Setting the parameters

MDLChunker

Two parameters have to be set before running a simulation : *memory size* and *perceptual span size*. Both sizes are expressed in bits.

The *memory size* parameter specifies the maximal memory load in terms of amount of information. Since units exceeding this value are forgotten (removed from memory), the smaller the size, the slower the learning. An infinite *memory size* would allow MDLChunker to use all information available to create new units. If this value is set below a critical threshold (depending on the dataset), no learning occurs. A default value of 150 bits allows MDLChunker to reproduce the vanishing sub-unit effect described in Giroux & Rey (2009). This parameter is of central importance to change the MDLChunker learning rate.

As opposed to the previous one, the *perceptual span size* has a limited impact on learning. It is the amount of information perceived from the input stream. It affects how existing units shape perception. It could be set to an arbitrary high value. Twenty-five bits are sufficient for the ready-to-use examples. MDLChunker cannot create a unit whose component codelengths exceed the *perceptual span size*. The perceptual span is displayed in the step-by-step mode (see figure below) which would help the user to adjust this parameter.

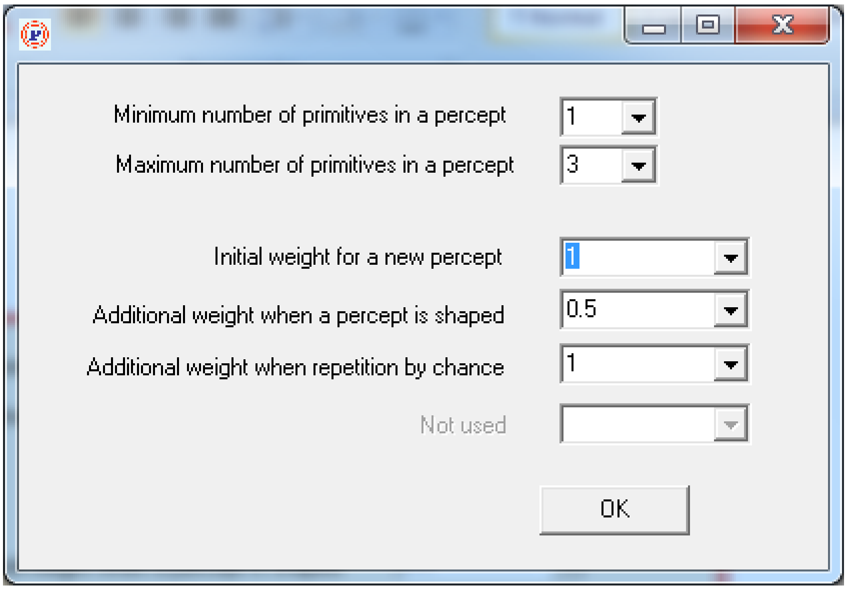


PARSER

For Parser, the more important parameters are the *rate of decay* and the *rate of interference*. Two main guidelines have to be kept in mind when these parameters are modified. First, the rate of forgetting (decay/interference) needs to be set at an intermediary value. If forgetting is too strong, the program fails to build any units, hence generating a low score of completeness. If forgetting is too low, the program stores a very large number of units, hence generating a low score of precision. Usually, running the *step-by-step mode* allows to find appropriate values without running complete simulations.

Second, manipulating forgetting through the decay parameter makes the model essentially sensitive to frequency, while manipulating forgetting through the interference parameter makes the model essentially sensitive to transitional probability and contingency (the *ready-to-use configuration* of Perruchet & Peereman, 2004 are specially well-suited to illustrate this claim, and for an explanation, see for instance Perruchet & Pacton, 2006).

The other parameters for Parser are displayed in the window below. The number of primitives composing a single percept may be conceived of as something like a working memory span, and it may make sense to adjust the values in some studies (e.g. in developmental investigations). The other values define the way the weight of the units are incremented. Admittedly, these values are set arbitrarily, but the problem is more apparent than real. Indeed, what is relevant is the ratio between the increments (due to the on-line processing of the units) and the decrements (due to forgetting). For the sake of between-studies comparisons, it is advisable to leave these values unchanged, and to manipulate the relevant ratio by changing what has been coined here as the main parameters, namely the rates of decay and interference.



The parameters that have been set as a default in the program are those used in the first paper on the model (Perruchet & Vinter, 1998), and which have been used in most subsequent papers.

Irrespective of the selected model, if you have changed one or several parameters, it is possible to *reset all parameters to their default values* by clicking on the appropriate button. Note that this button is active only if one or several parameters have been changed during a prior simulation, hence ensuring the user that that the current configuration is standard whenever the button is inactive.

4 - The '*Step by Step*' vs. '*Normal*' mode

'*Step-by-step*' provides a detailed analysis on a single run, while '*normal*' only provides the final results for each run from an analysis that may comprise several runs

5 - Other options

Chain the simulation with an earlier one

To start a new simulation in the state reached after a previous one, it suffices to indicate to the program the file in which the previous results have been saved (thanks to the next option), through a standard Windows dialog box. Even if the results file comprises the results from several runs, only the results from the first run will be considered as the starting state, whatever the number of run required for the new simulation. To avoid any ambiguity, it is advisable to restrict the use of this option to the case where both the earlier and the current simulation are performed on a single run. Note that this option is somewhat redundant with the possibility of chaining several languages in generating the corpus, which should be preferred if several runs are needed (a new corpus can be generated for each run). Chaining the simulations may be useful, however, for instance if the database comprises long excerpts of child-directed language.

Save the results

Under the '*Normal*' mode, a complete record of the session can be saved as a file thanks to this option. The saved file includes: (1) the summary file, presented below, which recapitulates the whole set-up, (2) the complete results for each run, including the data allowing to plot learning curves and (3) the final table displaying the mean scores. Saving the results under the *step-by-step* mode is not possible. Usually, the sequences of the *step-by-step* states of the system is not deemed to be recorded. If a record is wanted nevertheless, for instance for illustrative purposes, note that all the forms that appear on the screen (i.e., the summary form and the results for each step) can be individually copied/ pasted to a word processor or to a worksheet.

Number of runs

(Available only in the '*Normal*' mode. Run is set to 1 in the *Step-by-step* mode).

Manipulating the number of runs may have two different objectives. In most cases, multiple runs are performed to address the very same goal as using multiple subjects in experimental procedures, namely reducing the random variation inherent to single observations. Running multiple simulations and reporting averaged results is indeed common practice in modeling studies.

However, the objective may be different. Suppose one is using an existent corpus, say the *Little Red Riding Hood* story. It may be interesting to examine how the units formed after a single reading of the story evolve across subsequent exposures to the same story. A possibility consists in using a corpus in which several successive exemplars of the story have been appended, but the procedure is not very elegant and only the final state of the system will be available. Another possibility is to use the *Chain the simulation with an earlier one* option, but the user needs to save the results, indicate the new file name, and so on, for each new session, which is both time-consuming and error-prone. U-Learn allows to deal with this issue in a simple way. Suppose one wishes to examine how the lexicon grows throughout the exposure to ten successive readings of the *Little Red Riding Hood* story. It suffices to enter "10" as the number of runs, to click on the *Check/Change options* button, then to check the "*Simulate N successive sessions for a single learner (cumulative)*" option. The program will return the state of the system for each of the ten runs (note this option only works for Parser in the current version).

Irrespective of this choice, clicking on the *Check/Change options* button also allows to select between *Using a different corpus for each run* and *using the same corpus for all runs*. The second option is inactive for the MDLChunker. Indeed, the MDLChunker follows a deterministic algorithm that returns the same results on successive analyzes of a given corpus. In other words, the only source of variance across successive runs is due to the use of different corpuses on each run. Therefore, a different language will be automatically created for each run whenever *Number of Runs* > 1.

By contrast, Parser is based on the selection of randomly chosen unit candidates and hence, multiple runs with the same corpus return different results. The user may require either a new corpus be created for each run, or the same corpus be used for all runs. Generating a new corpus for each run unavoidably slows down the program. The time required to generate a corpus depends on the length of the corpus, but also on other conditions. As a rule, generating a corpus with words of different frequencies without immediate repetition may be very time-consuming. However, the results may depend to some extent on some particularities of the specific corpus on which the simulations are performed. For instance, it is possible that a given word appears more frequently as expected by chance at the beginning of the corpus. These potential biases are prevented by using a new corpus for each simulation, which is the default option.

Learning curves

The options regarding the learning curves (yes/no, mode of appearance). are available irrespective of the selected model. Note however that drawing learning curves is possible only when test items have been provided, given that what is plotted in these curves is the presence (or the weight) of the test items in the model's lexicon.

Random seed

There are three options:

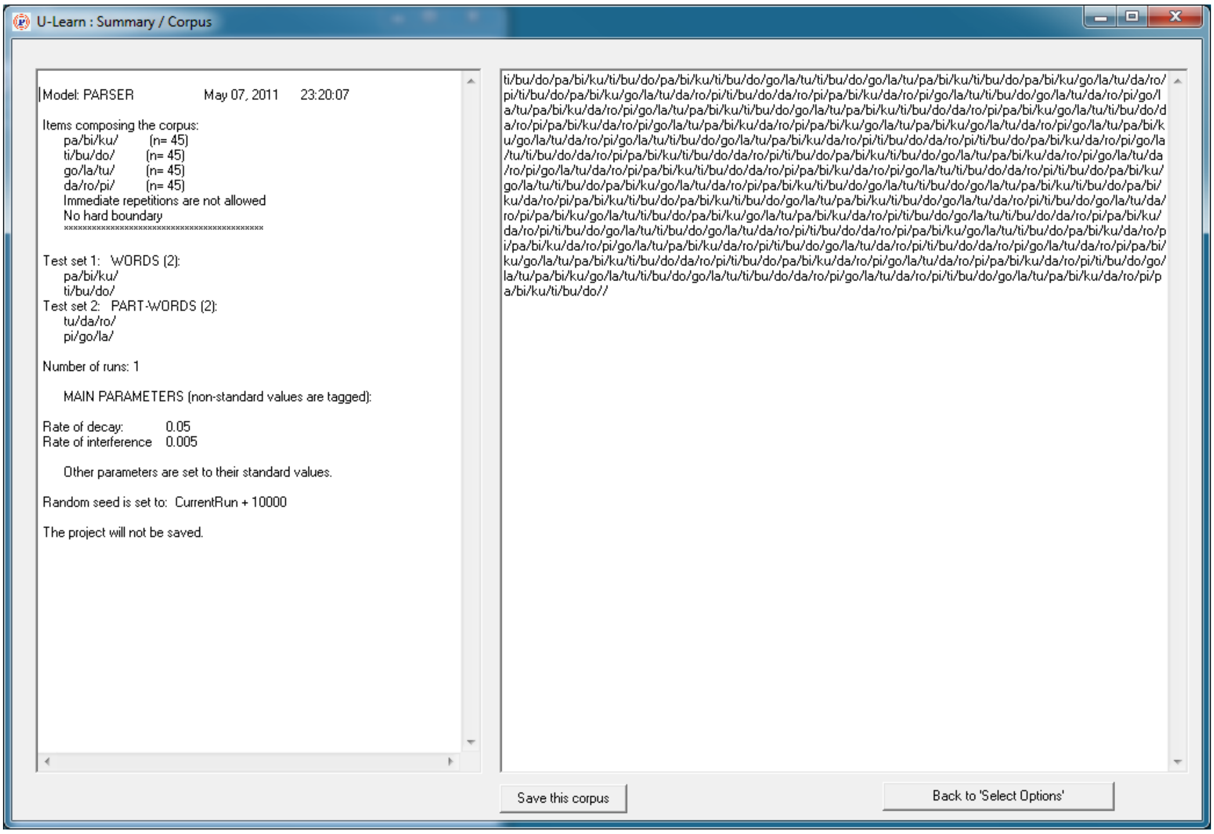
- Selecting '*Rand*'' ensures a different randomization in each case. However, this option does not allow to reproduce the same set of events. Reproducing the same set of events may be desirable for various purposes. For instance, if one wishes to draw learning curves by entering increasingly long corpuses, reproducing the same events across successive simulations appears appropriate. The next options make that possible.

- The option '*CurrentRun*' uses the number of the current run as the random seed (i.e., 1, 2, ... n, in succession). If you have selected this option during training, and you want to scrutinize the results reported for, say, Run 7, you can enter '7' as the random seed, for instance under the '*Step-by-step*' mode, to examine what happens in this particular situation. Note, however, that reproducing the same set of values implies that different corpuses have been used for different runs in the original simulation. If the box '*Use a different corpus for each run*' has been ticked off, the corpus processed during Run 7 is generated with 1 as random seed.

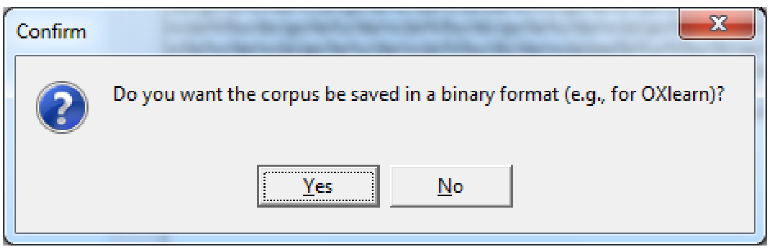
- The option '*CurrentRun + c*' is identical, except that a constant is added to the number of the current run. This allows to obtain an unlimited set of reproducible simulations. If you have selected this option during training with c=100, and you want to scrutinize the results reported for Run 7, you have to enter '107' as the random seed under the '*Step-by-step*' mode (but take care to avoid overlapping. Entering '3' as a constant, for instance, will generate the same values for the first run as for Run 4 under the '*CurrentRun*' option.)

6 - Summary / Corpus

A '*Summary/ Corpus*' window can be displayed whenever the mandatory options have been defined. If the option *'save the results* ' has been selected, the content of this form will be copied at the top of the results file. This windows allows to check that everything has been set as you intended to do. If something is wrong, do not edit this window. Changing the target values (e.g., the value of a parameter) on this form would have no other effect than getting you in a mess, because the changes would be ignored by the program, but reported in the project file if saved!



It is also possible to save selectively the corpus the program has generated. Clicking on "*Save the corpus*" opens a new window allowing the user to save the corpus either as such or in a binary code. In the latter case, U-Learn first extracts the primitives of the corpus (up to 100). These primitives are listed in the first line of the saved file as a function of their order of appearance in the corpus. Then each line of the file codes for a primitive, as a set of 0 and 1 separated by a space character. This option allows to use the data as input for other softwares, and notably OXlearn (Ruh and Westermann , 2009; http://psych.brookes.ac.uk/oxlearn). OXlearn is a free neural network simulation software that allows a



quick and easy start to connectionist modeling. Once the first line of the saved file has been removed, the file generated by U-Learn can be directly used as input for OXlearn, hence allowing to compare the results from chunk-based models (MDLChunker and Parser) to those from connectionist models (mostly SRN, given the data are sequential) on the very same data in a very easy way.

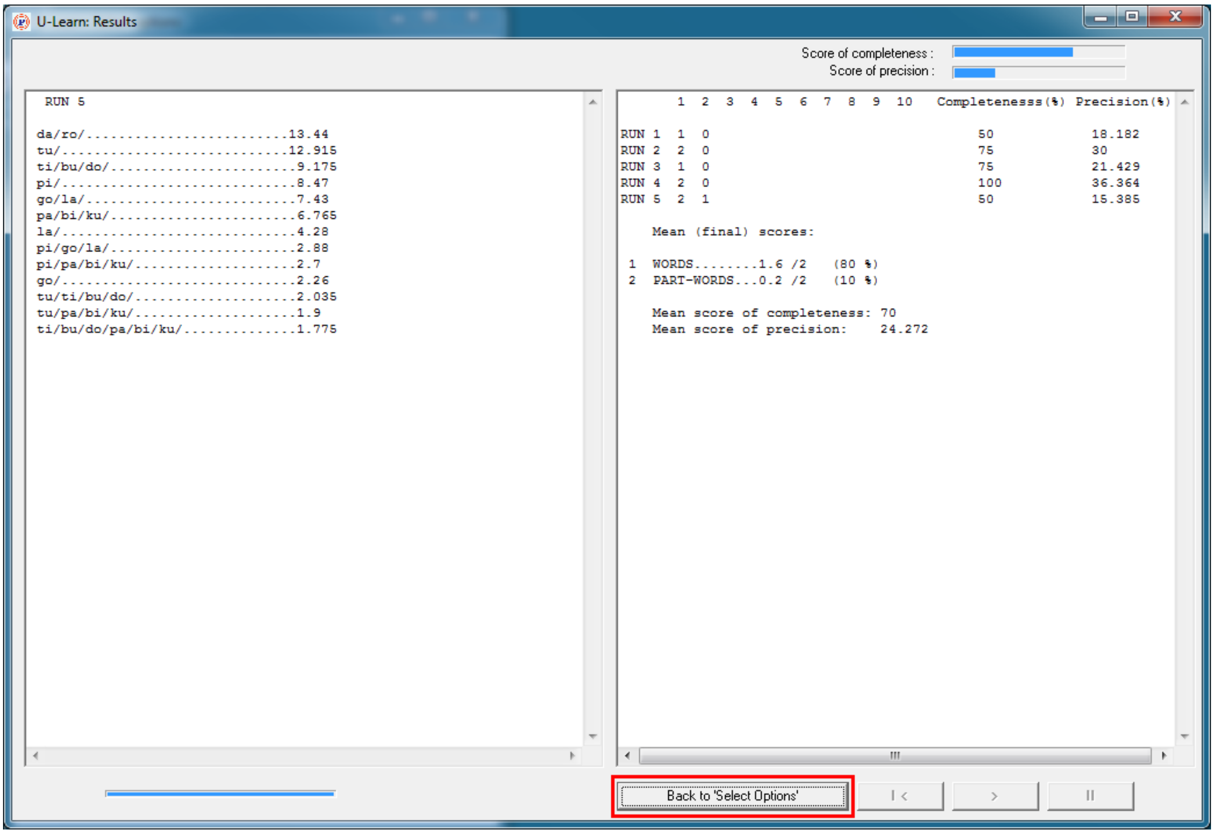
> When several runs are required with a different corpus for each run, only the first corpus is displayed. However, it is always possible to see and save the other corpuses, provided the random seed has been set to a controlled value. Suppose you want to see the corpus that was generated for Run 7, and that you had set the random seed to CurrentRun + 10000 (as in the example). It suffices to enter '10007' in the random seed combo box.

The program may fail to generate a corpus. It is not possible, for instance, to generate a language without immediate repetition comprising four words, *a*, *b*, *c*, and *d*, the frequency of which is *a*=10, *b*=10, *c*=10, and *d*=100 (or any value > 30). Indeed, the frequency of *a*, *b*, and *c*, is too low to avoid the repetition of *d*. Of course, the program does not assess the intractability of the problem through analytical means: It all simply gives up after 100.000 unsuccessful iterations. In this case, a message pops up: "*U-Learn fails to built a corpus. Please change the parameters*"

7 - Analyze the results

The result window comprises two main frames. Under the *'Step-by-step'* mode, the right-hand frame displays the result of the current step, and the left-hand frame displays the learning curves for each set of test items. As an option, the left-hand can also display the results for Step N-1, hence allowing analysis of the operations performed by the model on each step. The part of the currently processed corpus is displayed on the top of the page. The program automatically stops on the first window, in order to let it up to you how to go through the next steps. Standard media buttons allows to go ahead either step-by-step, or in a running way.

Under the '*Normal'* mode, if several runs have been required, the program waits for a few seconds after Run 1 in order to make it possible to click on the media button to go ahead run-by-run (click on ">"). Without any click during this delay, the processing of the following runs begins automatically. The left-hand frame displays the final state of the current run, and the right-hand panel displays a record of the final scores for each run. The content of these frames is reported on the result file (if the option *'Save the results*' has been selected), with the content of the left-hand frame being recorded in succession for each run (if several runs have been required), and the content of the last right-hand frame being appended to the file.



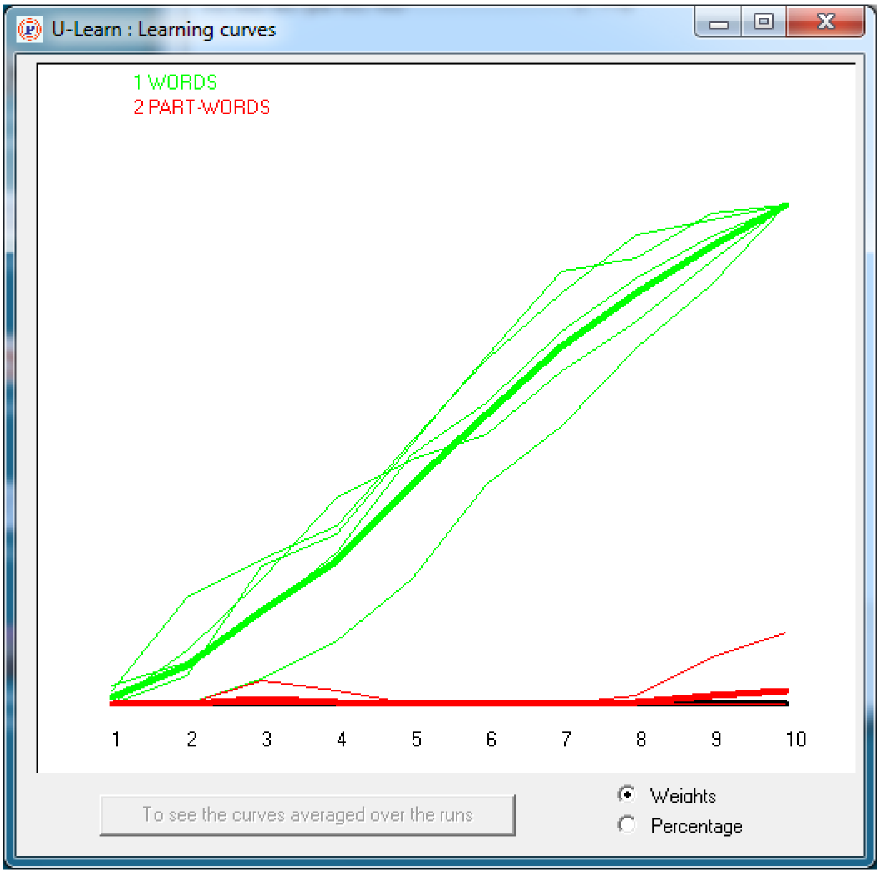
The two bars in the high-right corner indicate the scores of completeness (the proportion of words that are extracted) and precision (the proportion of actual words among the extracted units), respectively. Note that the scores of completeness and precision are correct only if all the words (and only the words) of the language have been previously provided. This condition is obviously fulfilled under the '*Generate one or several corpora*' option, given that the corpus is created on this basis, but under the '*Open an extant corpus*' mode, the program has no means to check that the words have been correctly entered.

If test lists have been provided, the program also returns the number of discovered items belonging to each list (e.g., test words and test part-words), and the number of items that have been found but that do not belong to the list(s).

Under the '*normal*' mode, the learning curves are displayed in an independent window, except if the '*no curve*' option has been ticked in the appropriate window. You may need to move this window to avoid any overlap with the numerical results. On the x-axis, the corpus is divided into 10 blocks of equal length, whatever its size. Below are the curves that you should obtain with the options '*Parser*' (with standard parameters), *Normal*, *Number of run* = 5, *Random seed* = *CurrentRun + 10000*. Individual curves are plotted in thin lines, and mean curves (which can be asked in option at the end of the analyses) are in wide lines.

> Note that all curves seemingly converge towards the same point. This is a consequence of the scaling method. Indeed, the scale is adjusted in order to fill all the available space for each run (the curves are drawn on-line, and it is not possible to anticipate the range of values for forthcoming runs). However, the mean curves are computed from the raw data, and not from the re-scaled scores. As a consequence, any point on the mean curves may deviate in substantial ways from the mean of the individual curves such as it could be estimated on the figure.

The numeric data used for the curves are reported at the end of the saved file, if the "*save the results*" option has been ticked. This allows to run ANOVAs with training blocks as a factor, and/or to plot a more sophisticated figure with Excel, for instance.


U-Learn : Learning curves
1 WORDS
2 PART-WORDS
1
2
3
4
5
6
7
8
9
10
To see the curves averaged over the runs
Weights
Percentage

Appendix 1: Source of the *'Ready to use configurations'*

Aslin, R. N., Saffran, J. R., & Newport, E. L. (1998). Computation of conditional probability statistics by 8-month-old infants. *Psychological Science*, 9, 321-324.

This is the first study using a 'frequency-balanced design'. In a 'frequency-balanced' design, some items are more frequent than other items in the familiarization speech. This allows to have test words and test part-words of equal frequency, but differing with regard to the transitional probability between their constituents. Note that the numbers of items that are displayed are those used in Aslin et al. In fact, the correct values to obtain a genuine frequency-balanced design would be 47 and 88, instead of 45 and 90 (French & Perruchet, BRM 2009).

Frank, M.C., Goldwater, S., Griffiths, T.L., & Tenenbaum, J.B. (2010). Modeling human performance in statistical word segmentation. *Cognition*, 117, 107-125.

Simulating the results of this paper implies to change several parameters. For Experiment 1, the number of words per sentence must be set successively to 1, 2, 3, 4, 6, 8, 12, and 24. For Experiment 2, the number of repetitions per word needs to be changed (8, 16, 50, 100, 150, and 200). Manipulating these variables in U-Learn takes just a few seconds. For Experiment 3, the required manipulation is a bit longer, because the number of different words must be set successively to 3, 4, 5, 6, and 9 (for 9 words, three new words have to be added to those that are supplied here). The paper shows that as long as ceiling effects are avoided, Parser provides very good predictions of human performances in all three experiments, but only when responses to words are considered. By contrast with human participants, Parser gets a null (or nearly null) score for part-words irrespective of the conditions.

Giroux, I., & Rey, A. (2009). Lexical and sub-lexical units in speech perception. *Cognitive Science*, 33, 260-272.

This study compares the recognition performance of adults for lexical and sublexical units of same length after hearing 2 or 10 min of an artificial spoken language. The results showed that performance on words is better than performance on part-words only after 10 min. These results are consistent with both MDLChunker's and Parser's predictions, Note that simulating all the results implies to change the frequency of words. The value by default (145) is for 10 min of exposure. For 2 min, the value needs to be set to 29. (see the Appendix A in the paper for details).

Perruchet, P., & Desaulty, S. (2008). A role for backward transitional probabilities in word segmentation? *Memory & Cognition*, 36, 1299-1305.

This study shows that adult participants are sensitive to the standard, 'forward', transitional probabilities, but also, more surprisingly, to backward transitional probabilities. Parser predicted this result, while a SRN is unable to account for it. The provided material is the one used in Experiment 2, in which the raw frequency is controlled, as in Aslin et al, 1998.

Perruchet, P. & Peereman, R. (2004). The exploitation of distributional information in syllable processing. *Journal of Neurolinguistics*, 17, 97-119

The paper reports an experiment collecting judgments of word-likeness as a function of the relationship between the phonemes composing the rimes (VC) of monosyllabic words. The contingency between Vs and Cs, as assessed by rphi (the normative measure of contingency) was the best predictor of children and adult judgments, and the backward transitional probability (pV/C) made a sizeable contribution. Parser proved to be a better predictor of performance than

an SRN (but better results are obtained if the role of interference in forgetting is increased –e.g. decay=0.025 and interference=0.025).

Perruchet, P., & Vinter, A. (1998). PARSER : A model for word segmentation. *Journal of Memory and Language,* 39, 246-263.

The provided material is the one used in Study 4. Parser turns out to be able to discover a word (' bu') that is a component of larger words (e.g., 'dutabu').

Saffran, J. R., Aslin, R. N., & Newport, E. L. (1996). Statistical learning by 8-month-old infants. *Science,* 274, 1926-1928.

One of the two seminal papers that prompted research on statistical learning.

Appendix 2: Expanding U-Learn with a new model.

If the option '*User's own model*' has been ticked, the user is required to indicate the name and the path of the executable file containing the model through a standard Windows dialog box. The general principles are the following:

-1 The transfer of data between the main program and the external program is carried out by the mean of temporary text files, which must be located in the folder comprising *U-Learn*.
-2 There are 3 intermediary text files. The first file (*param.tmp*) comprises information that the external program needs to work, such as the selected parameters for the model. The second file (*crps.tmp*) comprises the corpus, which is formatted as indicated above (i.e., with "/" and "//" used as a separator for delineating the primitives and the perceptually salient sentence boundaries respectively). The third file (*result.tmp*) is the file that is created in turn by the external program, and which is read by the main program for subsequent presentation and analyses.
-3 On each call of the external program, this program is expected to read the parameter file (*C:/UserFolder/param.tmp*) and the corpus file (*C:/UserFolder/crps.tmp*, with *UserFolder* standing for the folder comprising the *U-Learn.exe* file), to process the corpus once, and to return the results in *C:/UserFolder/result.tmp*. If *n* runs are required, the external program will be called *n* times.

The parameter file

When '*User's own model*' has been selected, the option window described above now allows to enter 8 parameters, numbered param-1 to param-8 (arbitrarily divided between 2 main and 6 additional parameters, but this classification is inconsequential). Note that the parameters are not necessarily digital values: any string of alphanumeric characters is accepted.

The parameter file that is generated by the main program comprises in succession (one value/ line):

SbSNormal: 1= Step-by-step, 2= Normal
the random seed selected by the user (always converted as a ready-to-use numerical value)
param-1
param-2
param-3
param-4
param-5
param-6
param-7
param-8

The result file

Any external program is supposed to generate a text file comprising first some information about the analysis, followed by a set of units. The file must comprise 3 alphanumeric variables (of course, one or several lines can be left empty). Then two variables are reserved for each unit. The first is for the unit itself, and the second is for a value, such as a weight or a size, related to the unit. If the user hopes to convey more detailed information for each unit, such as a weight and the location in the corpus where this unit was discovered by the model, an alphanumeric string has to be created in the external program (e.g., "W= x.x L= x"), and written as such on the line following the target unit.