Running the Learning Program

# Getting Started

Download the file workbench.zip from the website. This is a compressed (zip) file. Unzip it using a tool such as WinZip or 7-Zip. The compressed file contains:

1. This set of instructions
2. An executable .jar file, learner.jar, to run the learner program.
3. A folder ‘Language’ which contains data files used by the program
4. A folder ‘java’ containing the source code of the program.

Copy the .jar file and the ‘Language’ folder to a convenient location on your computer (before doing this, you need to ensure that Java is installed on your computer). Double-click the .jar file to start the Java program. This will initially show a file dialogue like:

Graphical user interface, text, application, email

Description automatically generated

You need to move to the ‘Language’ folder, and select the file langFiles.txt

Graphical user interface, text, application, email

Description automatically generated

langFiles.txt is a short text file which only gives the names and locations of other data files used by the program. Open it, and the program window will then look like:

Graphical user interface, text, application, Word

Description automatically generated

The learning program has a graphical user interface, which is similar (but not identical) to the online demonstration.

To run the learning program, you need to keep pressing the ‘Auto-Learn’ button. The first time you press it, the program will show the words which are used to generate random learning examples:

Words used to generate random learning examples:

Productivity [0,0]: apple, bed, table, biscuit, chair, banana, cherry, head, shoes, foot, girl, boy, girls, boys, pie, towel, he, she, I, him, me, anna, ben, tim, john, zack, (26 words)

Productivity [0,1]: green, red, cold, warm, hot, big, little, my, his, (9 words)

Productivity [1,0]: runs, ran, fell, falls, (4 words)

Productivity [1,1]: hits, eats, hugs, drops, sees(thing), saw(thing), (6 words)

‘Productivity’ is as explained in the first paper.

Each subsequent time you press the Auto-learn button, the program will run one or more cycles of 100 learning examples (initially, it runs 4 cycles per press) - showing both the learning examples, and the words learnt from them. For instance, after four cycles, you will see the last few learning examples:

Example Ex395: pie falls : <p-ie-> [f-or-l-z-]

Example Ex396: his red bed fell : [h-i-z-] <r-e-d-> [b-e-d-] [f-e-l-]

Example Ex397: warm girl runs : [w-or-m-] [g-ur-l-] [r-u-n-z-]

Example Ex398: girl sees(thing) zack : [g-ur-l-] [s-ee-z-] [z-a-k-]

Example Ex399: big anna falls : [b-i-g-] [a-n-u-] [f-or-l-z-]

[..] denote words that have been learnt; <..> denotes words that have not yet been learnt. Below these you will see the words learnt up to cycle 4:

Words learnt by cycle 4:

Productivity [0,0]: (21 words): h-i-m, z-a-k, b-i-s-k-i-t, ch-e-r-i, b-e-n, b-e-d, a-n-u, b-u-n-ar-n-u, ch-ai-r, sh-oo-s, h-ee, h-e-d, g-r-ee-n, t-ai-b-u-l, t-i-m, j-o-n, b-o-y, g-ur-l, t-a-w-u-l, sh-ee, sh-ee,

Productivity [0,1]: (8 words): h-u-g-s, m-ie, h-o-t, k-oe-l-d, w-or-m, b-i-g, h-i-z, l-i-t-l,

Productivity [1,0]: (5 words): f-e-l, f-or-l-z, r-a-n, r-u-n-z, d-r-o-p-s,

Productivity [1,1]: (3 words): s-or, s-ee-z, h-i-t-s,

You can inspect the feature structures learnt for these words, using the ‘Learn’ menu:

Graphical user interface, application, table

Description automatically generated

Note that some words, such as ‘hugs’, have not yet been learnt with their full [1,1] productivity. To see a typical word such as ‘hot’:

Diagram

Description automatically generated

Below the word feature structure, you will see two of the learning examples used to learn ‘hot’, and by pressing the ‘Step’ button you can see the results of generalising them together – and more learning examples for the word. You can also display the word ‘hot’ from the ‘Words’ menu. This is the word feature structure used to make the learning examples – not the feature structure learnt from them. The two feature structures are not necessarily identical, but they are usually the same or only have small differences.

You will see that making 400 learning examples, and learning words from them, only takes the model a few seconds.

You can keep pressing the auto-learn button as many times as you like; but generally after about 12 learning cycles, nearly all the words have been learnt, and not much else happens after that.

You can stop and restart the program – in which case it will generate a different set of random learning examples, and the results will be slightly different.

Every time you run the program, it produces some data files of its outputs, in the sub-folder ‘LearningResults’ of the language folder. These files are:

* ExampleLog.txt – a log of all learning examples and how they were processed
* learntMeanings.txt – the feature structures for each word learnt, and a list of the examples used to learn it
* progress.csv – a small spreadsheet of learning progress by cycle
* support.csv – a spreadsheet showing the level of support for each word after each learning cycle, and a final comparison of the learnt word feature structure with the feature structure used to make learning examples, with a match score percentage.

In every run of the program, previous versions of these files are overwritten.

# Changing the Parameters of the Program

In the ‘Language’ folder is a file ‘generator.xml’, which you can edit to change the main parameters of the program. If you open this file with a text editor, you will see in its second line:

<generator preLearn="1000" exPerCycle="100" cyclesPerPress = "4" memory="600" expLife="20" threshold="4" distractors = "2" startCriteria = "[8,4]" extraSlots="3" permutable="20" permutations="30">

You can edit these parameters to alter the running of the program. Their meanings are as follows:

* **preLearn** is the number of random learning examples which are made in the ‘pre-learning’ phase (when an infant hears sounds, but does not understand their meanings), used statistically to learn the segmentation of sounds into words.
* **exPerCycle** is the number of learning examples made per cycle
* **cyclesPerPress** is the number of cycles run for each press of the ‘Auto-Learn’ button
* **memory** is the number of learning examples by which the learner can look back, to find another learning example with the same word, to generalise the two examples to make a first approximation to a word
* **expLife** is an exponential decay factor used in calculating the level of support for a word
* **threshold :** if, when refining a word witha learning example, the information content of the word falls by more than the threshold, the example is not used
* **distractors:** is the average number wrongly inferred distractor meanings, plus one, made for each accurately inferred meaning.
* **startCriteria:** The first number is the number of bits in the meaning needed to start a new word, if some word meanings are known in the two learning examples used to help remove distractor meanings; the second number is the number of bits needed to start a word, if no other words in either of the first two examples are known, so they are likely to be distractor meanings.
* **extraSlots:** is the mean number of extra slots observed by the learner, but not expressed by the speaker, for each learning example.
* **permutable:** is the number of learning examples for a word which are randomly permuted for permutation learning of that word (in an approximation to Bayesian optimal learning)
* **permutations:** is the number of random permutations of learning examples for a word, which are tried out in permutation learning, to learn the best form of the word.

The rest of the file generator.xml is the definition of the probabilistic categorial grammar used to generate random learning examples. This grammar is not known to the learner. A typical entry for a non-leaf category in this grammar is:

<cat name="S" description="sentence">

<made weight="1" cats="NP TV NP"/>

<made weight="1" cats="NP IV"/>

</cat>

Here, the category is S for sentence. The two ‘made’ elements are two productions, which define respectively how an S can be made by a sequence NP TV NP, or by a sequence NP IV. The ‘weight’ values describe the relative frequencies with which these two constructions are used.

A leaf category in the grammar defines how words are used in categories:

<cat name="PN" description="proper noun">

<made weight="3" words="anna ben tim john"/>

<made weight="1" words="zack"/>

</cat>

To make a PN, one of the two productions (‘made’ element) is chosen at random depending on its weight, then a word is chosen at random from the production list.

That is the way candidate learning examples are made. For some examples, it is not possible to unify the word feature structures because of their semantic restrictions, so the utterance is rejected. For other examples, the speaker’s meaning is got by unifying the words. Duplicate learning examples are removed.

The grammar can be edited, using words already in the model, which you can see using the ‘Words’ menu. To add other words, or to test the model on a different language, the program has a graphical editor to make new words.

# The Graphical Language Editors

The program has an ‘Edit Language’ menu which you can use to display any word feature structure and edit it, or to add new word feature structures, and so on:

Graphical user interface, application

Description automatically generated

After any sequence of changes to a language definition, you use the ‘Save‘ command to save the changed state of the language model in a data file, and ’Finish’ to finish editing.

For some changes to a language definition, you may need to change the values of slots in the model or introduce new slot values, such as new sounds – or even add new node types for the feature structures. For these changes, you can use the ‘Edit Model’ menu:

Graphical user interface, application, Word

Description automatically generated

For instance, displaying the values for a typical slot c\_role (conversational role)

Diagram

Description automatically generated

You can add, remove or edit slots in this tree. For some complex slots, there is a hierarchy of values and sub-values.

After any sequence of edits to the language model (node types, slots, and slot values) you use the ‘Save ‘ command to save the changed state of the model in a data file, and ’Finish’ to finish editing.

These two editors can only be used before any automatic learning has been done. If in doubt, re-start the program to use the editors.

In this way you can either extend the model of English, or build a model of any language you like, to test the learning model.

# General Demonstration of the Language Model

Like the interactive demonstration, the program can also be used for other demonstrations of the language model, such as:

* Producing a sentence by unifying word feature structures
* Understanding a sentence by unifying word feature structures
* Understanding a new sentence entered by the user
* Learning a word by generalising a few learning examples

These can be accessed using the ‘Sentences’ menu, or by using the ‘Learn’ menu before any automated learning has been done.