TR-I-0377

FBI: A program for inferring stochastic grammar rules from example text

H. Lucke

10 March 1993

Abstract

An experimental program infering stochastic context-free grammar rules from example text. The program uses Bayesian belief updating to incorporate causal and evidential reasoning. It takes as input continuous unlabeled text and produces a stochastic context-free grammar in Chomsky Normal form with a pre-set number of terminal and non-terminal symbols.

1 Outline

The program fbi described in this document is a research program to study the effectiveness of Bayesian inference applied to the problem of grammar inference from example text. It takes as input a corpus of running text and iteratevly adjusts its internal grammar representation to fit the grammar to the corpus. A technical description of the method can be found elsewhere [2].

2 Building fbi

In order to build fbi, the following files should be present in a directory:

fbi.c	common_node_ops.c	graphic.c	graphic_seg.c
hinshi.c	matrix_primitives.c	node_op1.c	symbol_alloc.c
fbihead.h	makefile		

The command `make` will then build the program.

3 Running fbi

There are basically two modes in which the program can be run: A batch mode and a on-line mode. In the batch mode the the program reads in the entire training data at the beginning and then performs training on this data. In the on-line mode no data is read initially. Instead the program attempts to read data from standard input and after each symbol read performs as much processing as possible. In this mode the amount of training data is not limited in any way. The program can for example be connected to a continues text source such as electronic news groups and can be left to train indefinitely on the incoming text.

If the program is to be run in batch mode the program can be invoked by issuing the command

`fbi <filename>`,

where `<filename>` is the name of a parameter file, specifying certain model parameters and file names. Its format will be discussed in section 6.

If the program is to be run in on-line mode the corresponding invocation command is

`<data reader> | fbi <filename>`,

where `<data reader>` is a program that reads the training data formats it if necessary and writes it to standard output. A simple application could use the unix `cat` program.

4 Input format

At present, two input formats for the training data are supported.

simple format In the simple formats consists of plain ASCII with words being separated by whitespace. In other words consecutive sequence of non-whitespace characters terminated on either side by a whitespace character is treated as word.

ATR dialogue database format The ATR dialogue database and its file format is described elsewhere [1]. Essentially, each each word in the database occupies one line with several transcriptions given for each word as in the following line:

使い	つかい	使う	32	01	01
word as it appears	kana transcription	base form	品詞 code	活用型 code	活用形 code

Of this the program uses the 品詞 code and the 活用形 code. Together these define 51 distinct parts of speech.

5 Dictionary

The fbi program requires a "dictionary containing the list of words that it is capable of handling. If the training data is specified in the simple format, this dictionary is simply a file containing all the words to be processed separated by whitespace. The number of entries in this file determines the number of terminal symbols to be used by the program.

If the training data is in the ATR dialogue database format, each entry in the dictionary is of the form 品詞／活用形. The first ten lines of the dictionary currently used are included here to illustrate this format:

記号
形容詞／未然
形容詞／連用
形容詞／終止
形容詞／連体
形容詞／仮定
形容詞／語幹
形容詞
普通名詞
サ変名詞

6 The parameter file

As was mentioned in section 3, the fbi program requires a parameter file as argument when invoked. The parameter file consists of keywords followed by (optional) parameter values. An example file is provided below:

```
status                  st
batch
file_format             0
dictionary              /q28/users/lucke/work/bgi/dict
data                    /q28/users/lucke/work/bgi/data.100
non-terminals           16
logging                 25
iterations              2000
max_tree_base           12
penalty                 0.0
```

`tree_base_mult` 2.9

A description of the allowable keywords follows:

`status <file>` This line specifies the filename in which all (rather most) internal parameters are to be saved during training and from which parameters are read if the `cont` or `test` flags are specified.

During training parameters are saved after each parameter update as long as n seconds have elapsed since the last time parameters were saved. Here n is the parameter supplied to the `save_interval` keyword. In addition after every 10 iterations, parameters are saved in a file called `<file>.it_no`, where `it_no` is the current number of iteration. This saving is also suppressed if the previous save into such a file was less then n seconds ago, where n is the value supplied `save_interval_iter` keyword described below. At the end of training the parameters are once more saved in the file `<file>` irrespective of the timing of the last save.

If the `cont` flag is specified parameters are first read from `<file>` and then later during training saved into this file as well as into the file `<file>.it_no` as was described above. Please note that the original content of `<file>` may be erased.

If the `test` flag is specified the parameters are read once initially. In this case the program does not save to any file.

`cont` The `cont` flag tells the program to continue training using the parameters in the specified `status` file, rather than then starting a new training process with random parameters. This flag is no longer supported and may or may not work.

`test` This flag tells the process to merely measure the entropy over the specified data. When specified, the parameters are read from the status file and one iteration over the specified data is performed. parameters are not updated or saved. This flag is useful to evaluate the a parameter set over a test set.

`dictionary <file>` This line specifies the file to be used as dictionary file.

`data <file>` This line specifies the file to be read as training or test material, if the program operates in batch mode. Only one of the keywords `data` or `online` should be present.

`online` This flag tells the program to operate 'on line', i.e. to read its data from standard input. This key word conflicts with the keyword `data` and only one of the two should be specified.

`batch` This is an obsolete flag, which should always be specified for consistency with earlier parameter files. Not specifying it will cause the program to print an error message and exit.

`update_period <n>` This parameter specifies (the approximate) number of symbols processed before parameters should be updated. If the program is run on-line, this parameter must be specified. If the program runs in batch mode the parameter need not be specified in which case it defaults to the number of symbols in the training data. By

specifying this parameter in batch mode it is possible to have more than one parameter update per training epoch.

`var_update <n>` If specified, this option causes the update period to be increased by n symbols after each update.

`non-terminals <n>` This parameter specifies the number of non-terminal symbols to be used.

`logging <n>` This parameter specifies how much logging information is to be provided during test and training. The integer n is regarded as a binary number and each bit causes a particular type of verbose information to be printed. Some flags were used during program development and are no longer supported. The important flags are:

1 After each parameter update a line is printed consisting of 7 fields:

- number of symbols processed
- average entropy when measured at the terminal nodes
- average entropy when measured at the root nodes
- average size of trees
- number of non-terminal symbols
- number of non-terminal symbols used for production that are not pre-terminal productions. This only properly when the matrices are sparse coded.
- The number of production rules. Only applicable when the matrices are sparse coded.

2 Entropy information is printed after each symbol processed. (used during debugging)

4 Currently not used.

8 The program establishes an X connection and displays important parameters as histograms in special windows. In particular a windows are opened that display the B matrix and the matrices $A_{i..}$ for each i. Further a window displaying the priors used for the root nodes of trees of various sizes and a window displaying the usage of each symbol are opened. The usage of a symbol is defined as the average number of times it occurs on the left-hand side of a production rule. The usage window displays this in a normalized fashion, for all productions in the top row, for non-terminal production in the second row and for terminal production in the bottom row.

16 This flag will create an X window called "Segmentation" in which the tree structures constructed for the observation sequence are displayed. An example screen dump is shown in figure 1. The window shows the first 50 symbols of each iteration. The terminal symbols are printed at the bottom upon which a graphical structure representing the tree is drawn. At each node the non-terminal symbol with highest belief is also marked. The histogram above the terminal symbols represents the entropy for the terminal symbols i.e. the quantity $-log(P(s_t|s_t^-))$ where s_t is the current symbol and s_t^- is the context of this symbol within the current tree.

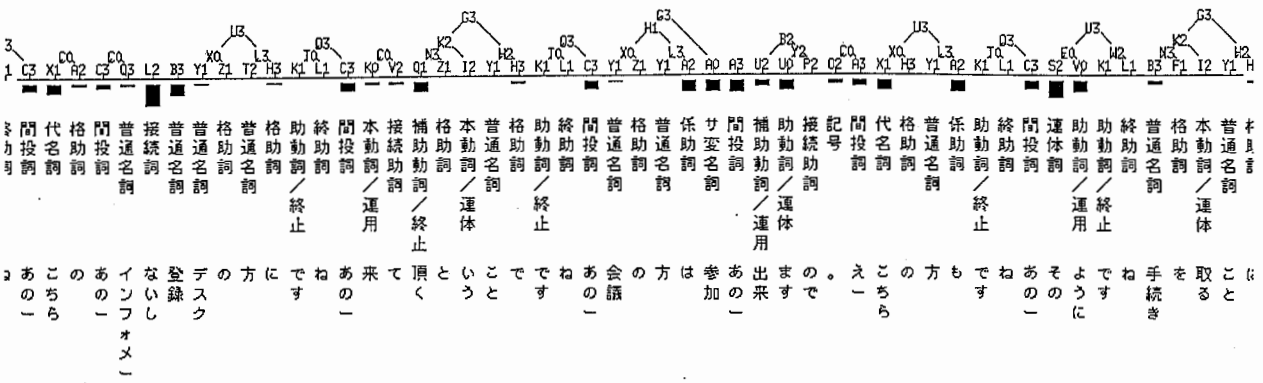

図 1: Segmentation window to be displayed when the 16 flag in the logging parameter is specified.

The appropriate numbers should be added and supplied to the `logging` keyword. A value of 0 provides no logging information at all. The default is 1.

`bigram` When specifying this flag, the program estimates or evaluates (depending on whether the `test` flag is specified a bi-gram and tri-gram grammar on the specified corpus. This is used as a comparison.

`iterations` <*n*> This parameter specifies the maximum number of iterations (parameter updates) to be performed before the program exits. The default value is 1000.

`file_format` <*n*> If n is 0 the simple file format is expected for the data file. If n is 1, the file is expected to be in ATR Dialogue database format.

`max_tree_base` <*n*> The maximum number of symbols spanned by each tree.

`tree_base_mult` <*f*> After each iteration, the average tree size from the last iteration is multiplied by this factor. The result is rounded down to the nearest integer This number becomes the maximum size of the trees to be considered during the next iteration unless this number is bigger than the number specified by the `max_tree_base` keyword above in which case the latter becomes the upper limit.

`penalty` <*f*> This number is added to the entropy of each tree when the best segmentation of the observation sequence is selected (segmentation problem). Thus a positive value should encourage larger trees whereas a negative value discourage larger trees. However when a non-zero value is specified the E-M algorithm used in the program is not guaranteed to converge. For this reason the value should always be set to 0.0 or otherwise not specified in which case it defaults to 0.0;

`save_interval` <*f*> The minimum time in seconds between two successive savings to the status file (see keyword status above). The default value is 300.0.

`save_interval_iter` <*f*> The minimum time in seconds between two successive savings to the file `status.it_no` (see keyword status above). The default value is 300.0.

7 The format of the status file

After each iteration and important internal parameters are save in a status file. This is useful should the process get killed during training. In this case training could in principal be re-commenced using the information in the status file. If the `cont` keyword is specified in the parameter file, the program should read the status file and recommence training from where the previous process finished. However this feature has not been supported recently and is not guaranteed to work.

At present the parameters describing the trained grammar can only be extracted from the status file. Since the process writes relatively often to this file a binary file format was chosen. Thus a programmer wishing to read the status file will need to write his own bit of C-code to read the file. The following routine used to save the status file is printed here to guide programmers to write their own reading routine.

Many of the parameters saved are only of interest to the training process and can be ignored. (Some of these have since become obsolete, but are kept here for compatibility with earlier versions of the program.) The comments on the right state the meaning of the parameter, if relevant, and the type in the C language. The most important parameters are the number of terminal and non-terminal symbols the A tensor, the B matrix and the priors.

```
#define WRITE(a) fwrite(&a,sizeof(a),1,fp)

void save_status(fname)
     char *fname;
{
  int i,j;
  FILE *fp = fopen(fname,"w");
  if (!fp) {
    fprintf(stderr, "Couldn't open %s.\n", fname);
    exit(1);
  }
  WRITE(version);              /* internal parameter (int) */
  WRITE(Nnt);                  /* number of non-terminal symbols (int) */
  WRITE(Nt);                   /* number of terminal symbols (int) */
  WRITE(llll);                 /* parameter supplied by keyword logging (int) */
  WRITE(maxiter);              /* parameter supplied by keyword maxiter (int) */
  WRITE(iter);                 /* current iteration number (int) */
  WRITE(symb);                 /* internal parameter (int) */
  WRITE(energy);               /* internal parameter (double) */
  WRITE(max_chain_length);     /* internal parameter (int) */
  WRITE(max_tree_base);        /* parameter supplied by max_tree_base keyword (int) */
  WRITE(cur_tree_base);        /* internal parameter (int) */
```

```
  WRITE(tau);                 /* internal parameter (double) */
  WRITE(save_interval);       /* internal parameter (double) */
  WRITE(trainall_flag);       /* internal parameter (int) */
  WRITE(batch_flag);          /* internal parameter (int) */
  WRITE(rand_flag);           /* internal parameter (int) */
  WRITE(on_line_flag);        /* 1 if training is online (int) */
  WRITE(update_period);       /* internal parameter (int) */
  WRITE(var_update_flag);     /* 1 if var_update keyword was specified (int) */
  WRITE(cur_update_period);   /* current update period (int) */
  WRITE(current_symbol);      /* internal parameter */
  WRITE(file_format);         /* argument supplied to file_format keyword (int) */
  WRITE(alloc_flag);          /* internal parameter (int) */
  WRITE(kill_thres);          /* internal parameter (double) */
  WRITE(split_thres);         /* internal parameter (double) */
  WRITE(update_inc);          /* argument supplied to var_update keyword (int) */
  WRITE(sparse_thres_mult);   /* internal parameter (double) */
  WRITE(fast_flag);           /* internal parameter (int) */
  WRITE(diffs_flag);          /* internal parameter (int) */
  fwrite(fname_diffs, sizeof(char), 100, fp);      /* obsolete */
  fwrite(fname_data, sizeof(char), 100, fp);       /* data file name */
  fwrite(fname_dict, sizeof(char), 100, fp);       /* dictionary file name */
  for(i=0;i<Nnt;i++) for(j=0;j<Nnt;j++)
    fwrite(A[i][j], sizeof(double), Nnt, fp);       /* The A tensor */
  for(i=0;i<Nnt;i++)
    fwrite(B[i], sizeof(double), Nt, fp);           /* The B matrix */
  for(j=0;j<cur_tree_base;j++)
    fwrite(prior[j+1], sizeof(double), Nnt, fp);    /* The priors */
  fclose(fp);
}
```

参考文献

[1] T. Ehara, N. Inoue, H. Kohyama, T. Hasegawa, F. Shohyama, and T. Morimoto. Contents of the ATR Dialogue Database. Technical Report TR-I-0186, ATR Interpreting Telephony Research Laboratories, 1990.

[2] Helmut Lucke. A method for inferring stochastic context-free grammars using the theory of bayesian causal trees. In *Proceedings of the Institute of Electronics, Information and Communication Engineers*, SP92-113, pages 79–86, December 1992.