9.2.8 tlbe

Application purpose

tlbe stands for **tl**earn **b**it **e**rror. It can extract the true error (not the averaged error generated by tlearn) for a distributed output and target. It will give the number of incorrect predictions over a tlearn run for a one-step-lookahead task.

Usage

```
*** TlearnBitError ver 0.24 -- outputs learning error statistics from tlearn
Usage: tlbe {tagsFile} {inputFile} {patternFile} {outputFile}
Where: {tagsFile}    -- tags representing input lines
       {inputFile}   -- output from tlearn verify sweep
                        (specify '-' for stdin)
       {patternFile} -- file to specify output targets
       {outputFile}  -- file to append output to
                        format: (#errors) (av. bit error) (av. correct) (av.
output sum)
                        (specify '-' for stdout)
```

Modules used

StdDefs, TokenLst, TokScan

Source code

Makefile

```
DISTFILES=tokenlst.c tokenlst.h tokscan.c tokscan.h tlbe.c tlbe Makefile stddefs.h
OBJFILES=tokenlst.o tokscan.o tlbe.o
TARGET=tlbe
VERSION=024

CC=gcc -g
CFLAGS=
LFALGS=

${TARGET}: ${OBJFILES}
   ${CC} -o ${TARGET} ${OBJFILES}
clean:
   rm -f *.o ${TARGET}
dist: ${DISTFILES}
   tar cvf ${TARGET}_v${VERSION}.tar ${DISTFILES}
   gzip ${TARGET}_v${VERSION}.tar

.c.o:
   ${CC} ${CFLAGS} ${LFLAGS} -c $*.c
```

tlbe.c

```
/* Tlearn bit error (testing) calculation
 *
 * Author: Dylan Muir (dr.muir@student.qut.edu.au)
 *         QUT MLRC LPG August 1999
 * Date: 18th August, 1999
 * Modified: 9th November, 1999
 * Version: 0.24
 */

/* --- Includes --- */

#include <stdio.h>
#include <string.h>
#include <unistd.h>     /* for exit() */
#include <stdlib.h>
#include "tokenlst.h"
#include "tokscan.h"
#include "stddefs.h"


/* --- Defines --- */

#define VERSION   "0.24"
```

```
#define HELPFILE  "tlbe.help"

#define  APPNAME     0
#define  TAGSFILE    1
#define  INPUTFILE   2
#define  PATTERNFILE 3
#define  OUTPUTFILE  4
#define  NUMARGS     5

#define  MAXTAGLENGTH   10


/* --- Helper functions' definitions --- */

void Usage(FILE *output, char *appName);
void  ProcessCommandLine(int  argc,  char  *argv[],  FILE  **tagsFile,  FILE  **inputFile,  FILE  **
patternFile, FILE **outputFile);
void Abort(int retnum);
void Help(char *appName);
void ReadTags(FILE *tagsFile, char **tagsArray[], int *numOutputs);
tokenList *ReadWholeFile(FILE *input);
void ReadPattern(float patternArray[], FILE *inputFile, bool *inFile);
double OutputSum(float patternArray[], int numOutputs);
int ReadTarget(FILE *patternFile, bool *inFile);
int MaxSlot(float patternArray[], int numOutputs);
void CloseFiles(FILE *tagsFile, FILE *inputFile, FILE *patternFile, FILE *outputFile);
bool Allocate2DArray(char ***array, int xSize, int ySize, size_t elementSize);


/* --- Main --- */

int main(int argc, char *argv[])
{
   FILE     *tagsFile, *inputFile, *patternFile, *outputFile;
   char     **tagsArray;
   float    *patternArray;
   int      numPatterns, numOutputs, target, predicted;
   long     bitError;
   double   avOutputSum;
   bool     inFile;

   ProcessCommandLine(argc, argv,
            &tagsFile, &inputFile, &patternFile, &outputFile);

   ReadTags(tagsFile, &tagsArray, &numOutputs);

   if ((patternArray = (float *) malloc(sizeof(float) * numOutputs)) == NULL) {
      fprintf(stderr, "*** Could not allocate pattern array.\n");
      Abort(7);
   }

   printf(" - TlearnBitError ver %s\n", VERSION);
   printf(" | Input file (tlearn output sweeps) [%s]\n", argv[INPUTFILE]);
   printf(" | Pattern file (targets) [%s]\n", argv[PATTERNFILE]);
   printf(" +-> Output file [%s", argv[OUTPUTFILE]);

   outputFile == stdout ? printf(" (stdout)]\n\n") : printf("]\n\n");

   printf("Read %d tags (%d output lines)\n", numOutputs, numOutputs);

   numPatterns = 0;
   avOutputSum = 0.0;
   bitError = 0;
   inFile = TRUE;
   while (inFile) {
      ReadPattern(patternArray, inputFile, &inFile);
      avOutputSum += OutputSum(patternArray, numOutputs);

      numPatterns++;

      target = ReadTarget(patternFile, &inFile);
      predicted = MaxSlot(patternArray, numOutputs);

      if (target != predicted)
         bitError++;

      if (numPatterns > 100 && (numPatterns % 1000 == 0))
         fprintf(stderr, "Processed %d patterns...\n", numPatterns);
   }
   fprintf(stderr, "Processed %d patterns total.\n", numPatterns - 1);
```

```
   fprintf(outputFile, "%d\t%5.2f%%\t%5.2f%%\t%.2f\n", bitError, bitError / (float) numPatterns *
100.0, (1.0 - (bitError / (float) numPatterns)) * 100.0, avOutputSum / (float) numPatterns);

   CloseFiles(tagsFile, inputFile, patternFile, outputFile);

   return 0;
}


/* --- Helper functions --- */

void Usage(FILE *output, char *appName)
{
   fprintf(output, "\n*** TlearnBitError ver %s -- outputs learning error statistics from tlearn\n",
VERSION);
   fprintf(output, "Usage: %s {tagsFile} {inputFile} {patternFile} {outputFile}\n", appName);
   fprintf(output, "Where: {tagsFile}    -- tags representing input lines\n");
   fprintf(output, "       {inputFile}   -- output from tlearn verify sweep\n");
   fprintf(output, "                        (specify '-' for stdin)\n");
   fprintf(output, "       {patternFile} -- file to specify output targets\n");
   fprintf(output, "       {outputFile}  -- file to append output to\n");
   fprintf(output, "                           format: (#errors) (av. bit error) (av. correct) (av.
output sum)\n");
   fprintf(output, "                        (specify '-' for stdout)\n");
   fprintf(output, "For detailed info, type %s --help\n\n", appName);
}

void  ProcessCommandLine(int  argc,  char  *argv[],  FILE  **tagsFile,  FILE  **inputFile,  FILE
**patternFile, FILE **outputFile)
{
   if ((!(argc < 2)) && ((strcasecmp(argv[1], "--help") == 0))) {
      Help(argv[APPNAME]);
      exit(0);
   }

   if (argc < NUMARGS) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Invalid number of arguments.\n");
      Abort(1);
   }

   if (argc > NUMARGS)
      fprintf(stderr, "--- Warning: extra arguments ignored.\n");

   if ((*tagsFile = fopen(argv[TAGSFILE], "rt")) == NULL) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Could not open tags file [%s].\n", argv[TAGSFILE]);
      Abort(3);
   }

   if (strcasecmp(argv[INPUTFILE], "-") == 0)
      *inputFile = stdin;
   else if ((*inputFile = fopen(argv[INPUTFILE], "rt")) == NULL) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Could not open input file [%s].\n", argv[INPUTFILE]);
      Abort(4);
   }

   if ((*patternFile = fopen(argv[PATTERNFILE], "rt")) == NULL) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Could not open pattern file [%s].\n", argv[PATTERNFILE]);
      Abort(5);
   }

   if (strcasecmp(argv[OUTPUTFILE], "-") == 0)
      *outputFile = stdout;
   else if ((*outputFile = fopen(argv[OUTPUTFILE], "at")) == NULL) {
      Usage(stderr, argv[APPNAME]);
      fprintf(stderr, "*** Could not open output file [%s].\n", argv[OUTPUTFILE]);
   }
}

void Abort(int retnum)
{
   fprintf(stderr, "Aborting...\n");
   exit(retnum);
}

void Help(char *appName)
{
   FILE *helpFile;
```

```
   if ((helpFile = fopen(HELPFILE, "wt")) == NULL) {
      fprintf(stderr, "*** Could not create help file [%s].\n", HELPFILE);
      Abort(2);
   }

   fprintf(helpFile, "----------------------------------------\n");
   fprintf(helpFile, "   Help file for TlearnBitErr ver %s\n", VERSION);
   fprintf(helpFile, "----------------------------------------\n\n");

   Usage(helpFile, appName);

   fprintf(helpFile, "\nThis program is used to output a table of stats derived\n");
   fprintf(helpFile, "from an Elman Tlearn testing (verification) sweep.\n");
   fprintf(helpFile, "It takes as input the following files:\n");
   fprintf(helpFile, "   o Tags file: Contains signatures for each ouput line,\n");
   fprintf(helpFile, "     separated by whitespace.\n");
   fprintf(helpFile, "   o Input file: (stdin if field = '-') Direct output from\n");
   fprintf(helpFile, "     tlearn verify option.\n\n");

   fprintf(helpFile, "It appends to the output file the averaged bit error and the\n");
   fprintf(helpFile, "averaged sum of the outputs.\n");

   fprintf(stderr, "Created %s\n", HELPFILE);
}


void ReadTags(FILE *tagsFile, char **tagsArray[], int *numOutputs)
{
   tokenList   *allTags, *index;
   int      numTags;

   allTags = ReadWholeFile(tagsFile);

   if (allTags == NULL) {
      fprintf(stderr, "*** Error reading token file.\n");
      Abort(5);
   }

   numTags = 0;
   index = allTags;
   while (index != NULL) {
      index = index -> NEXT;
      numTags++;
   }

   *numOutputs = numTags;

   if (!Allocate2DArray(tagsArray, numTags, 1, MAXTAGLENGTH)) {
      fprintf(stderr, "*** Could not allocate tag array.\n");
      Abort(6);
   }

   index = allTags;
   numTags = 0;
   while (index != NULL) {
      strcpy((*tagsArray)[numTags], index -> token);
      index = index -> NEXT;
   }

   DestroyTokenList(allTags);
}

tokenList *ReadWholeFile(FILE *input)
{
   tokenList   *list, *temp;
   bool     inFile;

   inFile = TRUE;
   list = ReadLineTokens(input, &inFile);    /* Get the first line */

   if (list == NULL)
      return list;

   while (inFile) {
      temp = ReadLineTokens(input, &inFile); /* Get the next line */
      list = ConcatenateTokenList(list, temp);
   }

   return list;
}

void ReadPattern(float patternArray[], FILE *inputFile, bool *inFile)
```

```
{
   tokenList   *outputs, *outputsIndex;
   int      arrayIndex;

   if ((outputs = ReadLineTokens(inputFile, inFile)) == NULL) {
      if (!*inFile) return;
      fprintf(stderr, "*** Error reading input file.\n");
      Abort(8);
   }

   arrayIndex = 0;
   outputsIndex = outputs;
   while (outputsIndex != NULL) {
      patternArray[arrayIndex] = atof(outputsIndex -> token);
      outputsIndex = outputsIndex -> NEXT;
      arrayIndex++;
   }

   DestroyTokenList(outputs);
}

double OutputSum(float patternArray[], int numOutputs)
{
   double   sum = 0;

   while (numOutputs > 0)
      sum += patternArray[--numOutputs];

   return sum;
}

int ReadTarget(FILE *patternFile, bool *inFile)
{
   tokenList   *pattern, *tIndex;
   int      target;

   if (!(pattern = ReadLineTokens(patternFile, inFile)))
      return 0;

   tIndex = pattern;
   while (tIndex -> NEXT != NULL)
      tIndex = tIndex -> NEXT;

   target = atoi(tIndex -> token);
   pattern = DestroyTokenList(pattern);

   return target;
}

int MaxSlot(float patternArray[], int numOutputs)
{
   int   maxSlot;
   float max;

   max = 0;
   maxSlot = 0;

   while (numOutputs > 0) {
      if (patternArray[numOutputs - 1] > max) {
         maxSlot = numOutputs;
         max = patternArray[numOutputs - 1];
      }
      numOutputs--;
   }
   return maxSlot;
}

void CloseFiles(FILE *tagsFile, FILE *inputFile, FILE *patternFile, FILE *outputFile)
{
   fclose(tagsFile);
   fclose(inputFile);
   fclose(patternFile);
   fclose(outputFile);
}

bool Allocate2DArray(char ***array, int xSize, int ySize, size_t elementSize)
{
   int   index;

   if ((*array = malloc(xSize * sizeof(void *))) == NULL)
      return FALSE;
```

```
   for (index = 0; index < xSize; index++) {
      if (((*array)[index] = malloc(ySize * elementSize)) == NULL)
         printf("ySize\n");

      xSize--;
   }
   return TRUE;
}

/* --- END of tlbe.c --- */
```