Miscellaneous: `init` and `misc`

Carl Edward Rasmussen

January 18th 1996

In this document various small routine are defined that take care of initiallizations and timing of the runs. There are two simple routines to create and destroy networks which take care of allocating and freeing memory. Also there is a routine to randomize the network weights; this routine takes a scale parameter and initializes weights randomlly from a zero mean normal distribution with a standard deviation given by the scale divided by the square root of the fan-in of the units the weights are projecting to.

"init.c" 1 ≡

```
#include <math.h>
#include <stdlib.h>
#include "util.h"

extern int  no_inp, no_tar, no_hid, no_wts;
extern real *w, **wi, **wd, **wo, *dw1, *dw2, *s, *hidden, *output;

void createNet()
{
  int i;

  w = (real *) malloc((size_t) no_wts*4*sizeof(real));
  wo = (real **) malloc((size_t) no_tar*sizeof(real*));
  wd = (real **) malloc((size_t) no_tar*sizeof(real*));
  wo[0] = &w[0]; wd[0] = &w[no_hid+1];
  for (i=1; i<no_tar; i++) { wo[i] = wo[i-1]+no_hid+1; wd[i] = wo[i]+no_inp; }
  wi = (real **) malloc((size_t) no_hid*sizeof(real*));
  wi[0] = &w[(no_hid+no_inp+1)*no_tar];
  for (i=1; i<no_hid; i++) wi[i] = wi[i-1]+no_inp+1;
  dw1 = &w[no_wts]; dw2 = &w[no_wts*2]; s = &w[no_wts*3];
  hidden = (real *) malloc((size_t) (no_hid+no_inp+1)*sizeof(real));
  hidden[no_hid] = 1.0;                              /* set the output bias */
  output = (real *) malloc((size_t) no_tar*sizeof(real));
}

void destroyNet()
{
  free(w); free(wi); free(hidden); free(output); free(wo); free(wd);
```

```
}

#define NORM cos(6.283185*drand48())*sqrt(-2.0*log(drand48()))

void randomizeNet(real magnitude)
{
  int i;

  for (i=0; i<(no_hid+no_inp+1)*no_tar; i++)
    w[i] = magnitude*NORM/sqrt(no_hid+no_inp+1.0);
  for (; i<no_wts; i++)
    w[i] = magnitude*NORM/sqrt(no_inp+1.0);
}
```

◇

Three routines handle the timing. The timers can be set and the elapsed time can be queried. All times are given in milli-seconds. When time has run out, the timeout function is called by an interrupt; this routine simply sets the global variable **TIMEOUT** to 1, which should be detected at convenience by the main program.

"misc.c" 2 ≡

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/time.h>
#include <limits.h>
#include <signal.h>

#define TIMER ITIMER_VIRTUAL

int TIMEOUT;        /* Global variable telling a running job that time is up */

void timeout()
{
  TIMEOUT++;     /* This will be detected by the main program at convenience */
  signal(SIGVTALRM, timeout);
}

void initTimer(struct itimerval *timer, int time)
{
  timer->it_interval.tv_sec = time/1000;
  timer->it_interval.tv_usec = 1000*(time-1000*(time/1000));
  timer->it_value.tv_sec = time/1000;
  timer->it_value.tv_usec = 1000*(time-1000*(time/1000));
  signal(SIGVTALRM, timeout);
  TIMEOUT = 0;
  setitimer(TIMER, timer, NULL);
}

long elapsedTime(struct itimerval *timer)
```

```
{
  long elapsed;

  getitimer(TIMER, timer);
  elapsed = 1000*(timer->it_interval.tv_sec-timer->it_value.tv_sec);
  elapsed += (timer->it_interval.tv_usec - timer->it_value.tv_usec)/1000;
  if (TIMEOUT) elapsed += TIMEOUT*(timer->it_interval.tv_sec*1000 +
                                   timer->it_interval.tv_usec/1000);
  return elapsed;
}
```
◇