Implementing Cubic Spline Transfer-Functions in the SNNS

Hofbauer Heinz
hofbauer.heinz@gmx.net

Berka Tobias
tobias_berka@yahoo.com

Stütz Thomas
arot.ts@mailexcite.com

Contents

1 Adding Activation Functions to SNNS in Four Steps

1.1 First step - Updating the Function Table

As a first step the new activation function has to be entered into the function table of the kernel, located in the file `<SNNS-DIR>/kernel/sources/func_tbl.c`. For one function three entries are necessary:

1. The actual activtaion function.
2. First derivate function of the activation function.
3. Second derivate function of the activation function.

These entries have to be created, even if there is no first or second derivate function.
The three entries for the sine activation function look like this:

1. activation function :
   `{"ACT_Sinus", ACT_FUNC, 0, 0, (FunctionPtr) ACT_SIN},`
2. first derivate function :
   `{"ACT_Sinus", ACT_DERIV_FUNC, 0, 0, (FunctionPtr) ACT_DERIV_SIN},`
3. first derivate function :
   `{"ACT_Sinus", ACT_2_DERIV_FUNC, 0, 0, (FunctionPtr) ACT_2_DERIV_SIN},`

For a better understanding I included the table description of the func_tbl.c file of the SNNSv4.1:

```
/*  The function table contains 5 entries:
     - the XGUI-Identifier of this function (a pointer to a string constant),
       note that the identifier have to match [A-Za-z]^[|, ]* (i.e. the first
       character of the symbol must be a letter and all other characters must
       not match [|, ])

     - an integer constant that determine the class of the function.
       8 different classes are possible:
       OUT_FUNC, ACT_FUNC, ACT_DERIV_FUNC, SITE_FUNC, UPDATE_FUNC, LEARN_FUNC,
       INIT_FUNC, PRUNING_FUNC and TEST_FUNC

     - the number of input parameters for this function (only needed for Update,
       Init, Learning and Test functions)

     - the number of output parameters returned by this function (only needed
       for Update, Learning and Test functions)

     - a pointer to the (user defined) function.
*/
```

If the activation function does not have a first or second derivate function then the appropriate entry must point to the dummy functions ACT_DERIV_DUMMY or ACT_2_DERIV_DUMMY respectively.

1.2 Second Step - Implementing the Activation Function

The activation functions, and the first and second derivate functions of the activation functions reside in `<SNNS-DIR>/kernel/sources/trans_f.c`.
The best idea is probably to copy/paste the declaration from another activation function:

```
FlintType ACT_SIN(struct Unit *unit_ptr)
{
ACT_FUNC_DEFS
register FlintType sum;

}
```

Here it gets a bit vague, as I did not look up how they declared the types' used here, but I do have some pretty neat ideas:

FlintType Float.

struct Unit Double linked list with some ID stuff that sparates between site and unit[1].

ACT_FUNC_DEFS Some macros that allow access to the list `struct Unit *unit_ptr`. To see how this macro works either search for them in the SNNS source files or read the code of the other activation functions.

Implementing the sine is quit simple: get the bias, get the inputs for every connection, add them up and return sin(sum).

```
FlintType ACT_SIN(struct Unit *unit_ptr)
{
  ACT_FUNC_DEFS
  register FlintType  sum;


  sum =  GET_UNIT_BIAS(unit_ptr);
  if (GET_FIRST_UNIT_LINK( unit_ptr ))
    do
      sum += GET_WEIGHTED_OUTPUT;
    while (GET_NEXT_LINK);
  else
    if (GET_FIRST_SITE( unit_ptr ))
      do
sum += GET_SITE_VALUE;
      while (GET_NEXT_SITE);

  return(sin(SIN_FAKTOR*sum));
}
```

Repeat this process for the first and second derivation function.

[1] Unit is the SNNS expression for neuron

1.3 Third Step - Defining the Function Prototypes

Declare the function prototypes in the file `<SNNS-DIR>/kernel/sources/trans_f.h`. In case of the sine activation function and its derivations it looks like this:

```
extern FlintType ACT_SIN(struct Unit *unit_ptr);
extern FlintType ACT_DERIV_SIN(struct Unit *unit_ptr);
extern FlintType ACT_2_DERIV_SIN(struct Unit *unit_ptr);
```

1.4 Fourth Step - Recompiling the Kernel

How this is done depends on version and installation, but one of these should work:

- make kernel
- make
- make compile
- build kernel
- build

2 Cubic Spline Interpolation

We aim to make the transfer function itself evolvable (using NetGen). We will not use a function table, but a tabulated function (i.e. a function defined via a set of points). Since the SNNS needs the first and second derivative, they have to be continuous. The goal of cubic spline interpolation is to get an interpolation formula which is smooth in the first derivative, and continuous in the second derivative, both within an interval and at its boundaries.

2.1 Mathematical Theory

(supplied by R. Schwaiger)

Given are n distinct points with $P_i = (x_i, y_i)$, $i = 1, \ldots, n$. We assume that the points are ordered and unique with respect to the first coordinate. Thus these n points define $n-1$ intervalls on the x-axes, denoted by $x \in [x_i, x_{i+1}]$. In each of these intervalls a function $f_i(x), i = 1, \ldots, n-1$ is defined. Since we assume that f_i is a cubic spline we may write

$$\begin{aligned} f_i(x) &= f_i(x_i) + a_i \times (x - x_i) + b_i \times (x - x_i)^2 + c_i \times (x - x_i)^3, \\ & \quad x_i \le x \le x_{i+1}, i = 1, \ldots, n-1 \end{aligned}$$

For x_{i+1}

$$f_i(x_{i+1}) = f_i(x_i) + a_i \times (x_{i+1} - x_i) + b_i \times (x_{i+1} - x_i)^2 + c_i \times (x_{i+1} - x_i)^3$$

if $\Delta x_i = x_{i+1} - x_i, i = 1, \ldots, n-1$ then

$$f_i(x_{i+1}) = f_i(x_i) + a_i \times \Delta x_i + b_i \times \Delta x_i^2 + c_i \times \Delta x_i^3$$

With another substitution we achieve

$$D_i = \frac{f_i(x_{i+1}) - f_i(x_i)}{\Delta x_i} = a_i + b_i \times \Delta x_i + c_i \times \Delta x_i^2$$

Or first requirements states that the first derivatives have to be equal in $x_i, i = 1, \ldots, n-1$, which yields

$$f_i'(x_{i+1}) = f_{i+1}'(x_{i+1})$$

which means

$$a_i + 2b_i \times \Delta x_i + 3c_i \times \Delta x_i^2 = a_{i+1}$$

Substituting equation in the equation above gives

$$c_i = \frac{a_i + a_{i+1} - 2D_i}{\Delta x_i^2}$$

Our second requirements states that the second derivatives have to be equal in $x_i, i = 1, \ldots, n-1$, which yields

$$f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$$

which means

$$b_i + 3c_i \times \Delta x_i = b_{i+1}$$

Substituting equation in the equation above gives

$$\frac{D_i - a_i}{\Delta x_i} + 2c_i \times \Delta x_i = \frac{D_{i+1} - a_{i+1}}{\Delta x_{i+1}} - c_{i+1} \times \Delta x_{i+1}$$

Combining equation and the equation above we get

$$\begin{aligned} a_i \times \Delta x_{i+1} + 2a_{i+1}(\Delta x_i + \Delta x_{i+1}) + a_{i+2} \times \Delta x_i &= 3(D_i \times \Delta x_{i+1} + D_{i+1} \times \Delta x_i) \\ & \quad i = 1, \ldots, n-2 \end{aligned}$$

3 Changes for NetGEN Cubic Spline

All files referenced in this section are located in the `<SNNS-DIR>/kernel/sources/` directory.

3.1 func_tbl.c

We added the following to the function table:

```
{"Act_NetGEN_CubicSpline", ACT_FUNC, 0, 0, (FunctionPtr) NetGEN_CS},
{"Act_NetGEN_CubicSpline", ACT_DERWeV_FUNC, 0, 0, (FunctionPtr) NetGEN_DERWeV_CS},
{"Act_NetGEN_CubicSpline", ACT_2_DERWeV_FUNC, 0, 0, (FunctionPtr) NetGEN_2_DERWeV_CS},
```

Additionally We added a nondummy pointer for the second derivation function of the sine function.

3.2 trans_f.c

We added the function bodies of the NetGEN CubicSpline functions and the second derivate function of the sine.

We also added a general function to get a FILE pointer to the files with the general function parameters (gfp). This functions was done with respect to future additions of other functions that require gfp.

Additionally we added a function that reads the coresponding file and calculates the spline function in the correct intervall or the requested derivation of the spline.

3.3 trans_f.h

We added the declarations for NetGEN and sine second derivation functions.

3.4 func_mac.h

We added the following macro to get the current unit's name:

```
#define  GET_UNIT_NAME( u_ptr )    ((u_ptr)->unit_name)
```

return value of this function is a `char *` .

4 Location and Layout of the GFP-Files

GFP stands for general function parameters. For every type of function used there is an individual directory, thus allowing future additions. The following two subsections deal with the GFP in regards to the cubic spline function.

4.1 GFP-Path

The path to the GFP files is given either by the environmental variable SNNS_GFP or, when it is not defined, a subdirectory `.SNNS_GFP` in the user's homedirectory (as defined by the envrionmental variable HOME) is used.
In case of the cubic spline the files are located in a subdirectory /CS.

e.g.: When `SNNS_GFP=/usr/share/snns-gfp` the files are located in the directory `/usr/share/snns-gfp/CS`. If the SNNS_GFP is undefined the HOME variable is used. Assuming that `HOME=/home/stud1/hhofbaue` the the files are located in `/home/stud1/hhofbaue/.SNNS_GFP/CS`.

The file name is identical to the unit name.
More detailed information can be obtained from the source code (section 5.1).

4.2 GFP File Layout

The first entry in the file is the number of entries. Followed by the given number of entries of the following format:

```
struct cs_gfp{
  float x;
  float y;
  float a;
  float b;
  float c;
}
```

Where the formula of the cubic spline function is

$$f(x') = y + a * z + b * z^2 + c * z^3,\ z = x' - x$$

Thus the file looks like this:

```
  int n
  struct cs_gfp(1)
  struct cs_gfp(2)
  .
  .
  .
  struct cs_gfp(n-1)
  struct cs_gfp(n)
```

The cs_gfp entries are sorted from the highest x to the lowest x (going from right to left in the functions graph), such that

$$cs\_gfp(1).x > cs\_gfp(2).x > \ldots > cs\_gfp(n).x$$

5 Function Source Code

For the readers convenience we included the code of the added functions.

5.1 get_GFP_file

This function returns a file pointer to the given function and filename.

```
FILE *get_GFP_file(char *func, char *filename)
{
  FILE *ret;
  char *buf;
  if( (buf = getenv("SNNS_GFP")) == NULL)
  {
      if( (buf = getenv("HOME")) == NULL) return NULL;
      strcat(buf, "/.SNNS_GFP");
  }
  strcat(buf, "/");
  strcat(buf, func);
```

```
  strcat(buf, "/");
  strcat(buf, filename);
  return fopen(buf, "r");
}
```

5.2 cs_f

This function calculates the value of the cubic spline (in the given derivative) for a specific unit.

```
float cs_f( float x, char *name, int derivation)
{
  FILE *fp;
  struct {
        float x;
        float y;
        float a;
        float b;
        float c;
    }gfp;

    int rv;
    int entries = 0;

    /****** when the file is not found -> id ******/
    if ( (fp = get_GFP_file("CS", name)) == NULL ) return x;

    rv = fread( &entries, sizeof(entries), 1, fp);
    if ( (rv != sizeof(entries)) || (entries < 2))
    {
        fclose(fp);
        return x;                 /****** no entries or to few enries -> id ******/
    }

    rv = fread( &gfp,sizeof(gfp), 1, fp);
    if( rv != sizeof(gfp) )
    {
        fclose(fp);
        return x;
    }

    if( x >= gfp.x )             /****** over right boundary ******/
    {
        fclose(fp);
        return gfp.y;
    }
    entries--;

    while(entries > 0 && x < gfp.x )
    {
        rv = fread( &gfp, sizeof(gfp), 1, fp);
        if( rv != sizeof(gfp) )  /****** read error -> id ******/
        {
            fclose(fp);
```

```
            return x;
        }
        entries--;
    };

    if( entries == 0 && x < gfp.x) /****** over left(lower) boundary ******/
    {
        fclose(fp);
        return gfp.y;
    }

    fclose(fp);
    rv = x - gfp.x;

    switch(derivation)
    {
        case 0 :
            rv = gfp.y + gfp.a*rv + gfp.b*rv*rv + gfp.c*rv*rv*rv;
            break;
        case 1 :
            rv = gfp.a + 2*gfp.b*rv + 3*gfp.c*rv*rv;
            break;
        case 2 :
            rv = 2*gfp.b + 6*gfp.c*rv;
            break;
        default :
            rv = x;        /****** no correct derivation number -> id ******/
    };

    return rv;
}
```