

# **Fast Artificial Neural Network Library**

**Steffen Nissen**

**Evan Nemerson**

**Fast Artificial Neural Network Library**

by Steffen Nissen and Evan Nemerson

Copyright © 2004

# Table of Contents

<b>1. Introduction</b> .....	<b>1</b>
1.1. Getting FANN .....	1
1.2. Installation .....	1
1.2.1. RPMs .....	1
1.2.2. DEBs.....	1
1.2.3. Windows .....	1
1.2.4. Compiling from source .....	2
1.3. Getting Started.....	2
1.3.1. Training.....	2
1.3.2. Execution .....	4
1.4. Getting Help .....	4
<b>2. Advanced Usage</b> .....	<b>5</b>
2.1. Adjusting Parameters .....	5
2.2. Network Design.....	5
2.3. Understanding the Error Value .....	6
2.4. Training and Testing.....	6
2.5. Avoid Over-Fitting .....	7
2.6. Adjusting Parameters During Training .....	7
<b>3. Fixed Point Usage</b> .....	<b>9</b>
3.1. Training a Fixed Point ANN .....	9
3.2. Running a Fixed Point ANN .....	10
3.3. Precision of a Fixed Point ANN.....	11
<b>4. Neural Network Theory</b> .....	<b>13</b>
4.1. Neural Networks.....	13
4.2. Artificial Neural Networks .....	13
4.3. Training an ANN.....	14
<b>5. API Reference</b> .....	<b>15</b>
5.1. Creation, Destruction, and Execution.....	15
fann_create.....	15
fann_create_array .....	15
fann_create_shortcut.....	16
fann_create_shortcut_array.....	16
fann_destroy .....	17
fann_run.....	17
fann_randomize_weights.....	18
fann_init_weights .....	18
fann_print_connections.....	19
5.2. Input/Output .....	19
fann_save .....	20
fann_save_to_fixed .....	20
fann_create_from_file.....	21
5.3. Training .....	21
fann_train.....	21
fann_test.....	21
fann_get_MSE .....	22
fann_reset_MSE .....	22
5.4. Training Data.....	23
fann_read_train_from_file .....	23

fann_save_train .....	23
fann_save_train_to_fixed.....	24
fann_destroy_train .....	24
fann_train_epoch .....	25
fann_test_data .....	25
fann_train_on_data .....	25
fann_train_on_data_callback.....	26
fann_train_on_file.....	27
fann_train_on_file_callback .....	27
fann_shuffle_train_data .....	28
fann_merge_train_data .....	28
fann_duplicate_train_data.....	28
5.5. Options .....	29
fann_print_parameters .....	29
fann_get_training_algorithm .....	29
fann_set_training_algorithm.....	30
fann_get_learning_rate .....	30
fann_set_learning_rate.....	30
fann_get_activation_function_hidden.....	31
fann_set_activation_function_hidden .....	31
fann_get_activation_function_output .....	32
fann_set_activation_function_output.....	32
fann_get_activation_steepness_hidden.....	33
fann_set_activation_steepness_hidden .....	33
fann_get_activation_steepness_output .....	34
fann_set_activation_steepness_output.....	34
fann_set_train_error_function .....	35
fann_get_train_error_function.....	35
fann_get_quickprop_decay.....	35
fann_set_quickprop_decay .....	36
fann_get_quickprop_mu .....	36
fann_set_quickprop_mu .....	37
fann_get_rprop_increase_factor .....	37
fann_set_rprop_increase_factor.....	38
fann_get_rprop_decrease_factor.....	38
fann_set_rprop_decrease_factor .....	38
fann_get_rprop_delta_min.....	39
fann_set_rprop_delta_min .....	39
fann_get_rprop_delta_max .....	40
fann_set_rprop_delta_max .....	40
fann_get_num_input.....	41
fann_get_num_output.....	41
fann_get_total_neurons.....	41
fann_get_total_connections .....	42
fann_get_decimal_point .....	42
fann_get_multiplier.....	42
5.6. Error Handling.....	43
fann_get_errno .....	43
fann_get_errstr .....	43
fann_reset_errno .....	44
fann_reset_errstr .....	44
fann_set_error_log.....	45

fann_print_error.....	45
5.7. Data Structures .....	45
struct fann .....	46
struct fann_train_data .....	48
struct fann_error.....	49
struct fann_neuron .....	50
struct fann_layer .....	51
5.8. Constants .....	51
Training algorithms.....	51
Activation Functions.....	52
Training Error Functions .....	54
Error Codes.....	54
5.9. Internal Functions.....	56
5.9.1. Creation And Destruction .....	56
5.9.2. Input/Output.....	56
5.9.3. Training Data .....	57
5.9.4. Error Handling .....	59
5.9.5. Options.....	59
5.10. Deprecated Functions .....	60
5.10.1. Mean Square Error.....	60
5.10.2. Get and set activation function steepness. ....	61
<b>6. PHP Extension.....</b>	<b>64</b>
6.1. Installation .....	64
6.1.1. Using PEAR.....	64
6.1.2. Compiling into PHP.....	64
6.2. API Reference .....	65
fann_create.....	65
fann_train.....	66
fann_save .....	67
fann_run.....	67
fann_randomize_weights.....	68
fann_init_weights .....	68
fann_get_MSE .....	69
fann_get_num_input.....	69
fann_get_num_output.....	70
fann_get_total_neurons.....	70
fann_get_total_connections .....	70
fann_get_learning_rate .....	71
fann_get_activation_function_hidden.....	71
fann_get_activation_function_output .....	72
fann_get_activation_steepness_hidden.....	72
fann_get_activation_steepness_output .....	73
fann_set_learning_rate.....	73
fann_set_activation_function_hidden .....	73
fann_set_activation_function_output.....	74
fann_set_activation_steepness_hidden .....	74
fann_set_activation_steepness_output.....	75
<b>7. Python Bindings .....</b>	<b>76</b>
7.1. Python Install.....	76

<b>8. Delphi Bindings .....</b>	<b>77</b>
8.1. Delphi Install .....	77
8.2. TFannNetwork.....	77
8.3. Known Problems .....	77
<b>Bibliography .....</b>	<b>78</b>

# List of Examples

1-1. Simple training example.....	3
1-2. Simple execution example.....	4
2-1. The internals of the <code>fann_train_on_file</code> function, without writing the status line.....	6
2-2. Test all of the data in a file and calculates the mean square error.....	7
3-1. An example of a program written to support training in both fixed point and floating point numbers.....	9
3-2. An example of a program written to support both fixed point and floating point numbers .....	10
5-1. <code>fann_create_array</code> example.....	15
6-1. <code>fann_create</code> from scratch .....	65
6-2. <code>fann_create</code> loading from a file.....	65
6-1. <code>fann_create</code> from training data.....	66
6-1. <code>fann_runExample</code> .....	67

# Chapter 1. Introduction

fann - Fast Artificial Neural Network Library is written in ANSI C. The library implements multilayer feedforward ANNs, up to 150 times faster than other libraries. FANN supports execution in fixed point, for fast execution on systems like the iPAQ.

## 1.1. Getting FANN

Copies of FANN can be obtained from our SourceForge project page, located at <http://www.sourceforge.net/projects/fann/>

You can currently get FANN as source code (`fann-*.tar.bz2`), Debian packages (`fann-*.deb`), or RPM's (`fann-*.rpm`).

FANN is available under the terms of the GNU Lesser General Public License (<http://www.fsf.org/copyleft/lesser.html>).

## 1.2. Installation

### 1.2.1. RPMs

RPMs are a simple way to manage packages, and is used on many common Linux distributions such as Red Hat (<http://www.redhat.com>), Mandrake (<http://www.mandrake.com/>), and SuSE (<http://www.suse.com/>).

Two separate packages exist; `fann`, the runtime library, and `fann-devel`, the development library and header files.

After downloading FANN, simply run (as root) the following command: **`rpm -ivh $PATH_TO_RPM`**

### 1.2.2. DEBs

DEBs are packages for the Debian (<http://www.debian.org>) Linux distribution. Two separate packages exists `libfann1` and `libfann1-dev`, where `libfann1` is the runtime library and `libfann1-dev` is the development library.

Fann is included in the testing distribution of Debian, so testing users can simply run (as root) the following command: **`apt-get install libfann1 libfann1-dev`**.

After downloading the FANN DEB package, simply run (as root) the following command: **`dpkg -i $PATH_TO_DEB`**

### 1.2.3. Windows

FANN >= 1.1.0 includes a Microsoft Visual C++ 6.0 project file, which can be used to compile FANN for Windows. To build the library and examples with MSVC++ 6.0:

First, navigate to the MSVC++ directory in the FANN distribution and open the `all.dsw` workspace. In the Visual Studio menu bar, choose "Build" -> "Batch build...", select the project configurations that you would like to build (by default, all are selected), and press "rebuild all"

When the build process is complete, the library and examples can be found in the `MSVC++\Debug` and `MSVC++\Release` directories and the release versions of the examples are automatically copied into the `examples` where they are supposed to be run.

### 1.2.4. Compiling from source

Compiling FANN from source code entails the standard GNU autotools technique. First, configure the package as you want it by typing (in the FANN directory), `./configure` If you need help choosing the options you would like to use, try `./configure --help`

Next, you have to actually compile the library. To do this, simply type `make`

Finally, to install the library, type `make install`. Odds are you will have to be root to install, so you may need to `su` to root before installing. Please remember to log out of the root account immediately after `make install` finishes.

Some people have experienced problems with compiling the library with some compilers, especially windows compilers which can not use GNU autotools. Please look through the help forum ([http://sourceforge.net/forum/forum.php?forum\\_id=323465](http://sourceforge.net/forum/forum.php?forum_id=323465)) and the mailing list (<http://sourceforge.net/mailarchive/forum.php?forum=fann-general>) archives for info on how these problems was solved. If you do not find any information here, feel free to ask questions.

## 1.3. Getting Started

An ANN is normally run in two different modes, a training mode and an execution mode. Although it is possible to do this in the same program, using different programs is recommended.

There are several reasons to why it is usually a good idea to write the training and execution in two different programs, but the most obvious is the fact that a typical ANN system is only trained once, while it is executed many times.

### 1.3.1. Training

The following is a simple program which trains an ANN with a data set and then saves the ANN to a file.

#### Example 1-1. Simple training example

```
#include "fann.h"

int main()
{
    const float connection_rate = 1;
    const float learning_rate = 0.7;
    const unsigned int num_input = 2;
    const unsigned int num_output = 1;
    const unsigned int num_layers = 3;
    const unsigned int num_neurons_hidden = 4;
    const float desired_error = 0.0001;
    const unsigned int max_iterations = 500000;
    const unsigned int iterations_between_reports = 1000;

    struct fann *ann = fann_create(connection_rate, learning_rate, num_layers,
                                   num_input, num_neurons_hidden, num_output);

    fann_train_on_file(ann, "xor.data", max_iterations,
                      iterations_between_reports, desired_error);

    fann_save(ann, "xor_float.net");

    fann_destroy(ann);

    return 0;
}
```

The file xor.data, used to train the xor function:

```
4 2 1
0 0
0
0 1
1
1 0
1
1 1
0
```

The first line consists of three numbers: The first is the number of training pairs in the file, the second is the number of inputs and the third is the number of outputs. The rest of the file is the actual training data, consisting of one line with inputs, one with outputs etc.

This example introduces several fundamental functions, namely `fann_create`, `fann_train_on_file`, `fann_save`, and `fann_destroy`.

## 1.3.2. Execution

The following example shows a simple program which executes a single input on the ANN. The program introduces two new functions (`fann_create_from_file` and `fann_run`) which were not used in the training procedure, as well as the `fann_type` type.

### Example 1-2. Simple execution example

```
#include <stdio.h>
#include "floatfann.h"

int main()
{
    fann_type *calc_out;
    fann_type input[2];

    struct fann *ann = fann_create_from_file("xor_float.net");

    input[0] = 0;
    input[1] = 1;
    calc_out = fann_run(ann, input);

    printf("xor test (%f,%f) -> %f\n",
           input[0], input[1], *calc_out);

    fann_destroy(ann);
    return 0;
}
```

## 1.4. Getting Help

If after reading the documentation you are still having problems, or have a question that is not covered in the documentation, please consult the `fann-general` mailing list. Archives and subscription information are available here (<http://lists.sourceforge.net/lists/listinfo/fann-general>).

## Chapter 2. Advanced Usage

This section describes some of the low-level functions and how they can be used to obtain more control of the fann library. For a full list of functions, please see the API Reference, which has an explanation of all the fann library functions. Also feel free to take a look at the source code.

This section describes different procedures, which can help to get more power out of the fann library: *Adjusting Parameters*, *Network Design*, *Understanding the Error Value*, and *Training and Testing*.

### 2.1. Adjusting Parameters

Several different parameters exist in an ANN, these parameters are given defaults in the fann library, but they can be adjusted at runtime. There is no sense in adjusting most of these parameters after the training, since it would invalidate the training, but it does make sense to adjust some of the parameters during training, as will be described in *Training and Testing*. Generally speaking, these are parameters that should be adjusted before training.

The learning rate is one of the most important parameters, but unfortunately it is also a parameter which is hard to find a reasonable default for. I (SN) have several times ended up using 0.7, but it is a good idea to test several different learning rates when training a network. It is also worth noting that the activation function has a profound effect on the optimal learning rate [Thimm and Fiesler, 1997]. The learning rate can be set when creating the network, but it can also be set by the `fann_set_learning_rate` function.

The initial weights are random values between -0.1 and 0.1, if other weights are preferred, the weights can be altered by the `fann_randomize_weights` or `fann_init_weights` function.

In [Thimm and Fiesler, *High-Order and Multilayer Perceptron Initialization*, 1997], Thimm and Fiesler state that, "An (*sic*) fixed weight variance of 0.2, which corresponds to a weight range of [-0.77, 0.77], gave the best mean performance for all the applications tested in this study. This performance is similar or better as compared to those of the other weight initialization methods."

The standard activation function is the sigmoid activation function, but it is also possible to use the threshold activation function. A list of the currently available activation functions is available in the *Activation Functions* section. The activation functions are chosen using the `fann_set_activation_function_hidden` and `fann_set_activation_function_output` functions.

These two functions set the activation function for the hidden layers and for the output layer. Likewise the steepness parameter used in the sigmoid function can be adjusted with the `fann_set_activation_steepness_hidden` and `fann_set_activation_steepness_output` functions.

FANN distinguishes between the hidden layers and the output layer, to allow more flexibility. This is especially a good idea for users wanting discrete output from the network, since they can set the activation function for the output to threshold. Please note, that it is not possible to train a network when using the threshold activation function, due to the fact, that it is not differentiable.

## 2.2. Network Design

When creating a network it is necessary to define how many layers, neurons and connections it should have. If the network become too large, the ANN will have difficulties learning and when it does learn it will tend to over-fit resulting in poor generalization. If the network becomes too small, it will not be able to represent the rules needed to learn the problem and it will never gain a sufficiently low error rate.

The number of hidden layers is also important. Generally speaking, if the problem is simple it is often enough to have one or two hidden layers, but as the problems get more complex, so does the need for more layers.

One way of getting a large network which is not too complex, is to adjust the `connection_rate` parameter given to `fann_create`. If this parameter is 0.5, the constructed network will have the same amount of neurons, but only half as many connections. It is difficult to say which problems this approach is useful for, but if you have a problem which can be solved by a fully connected network, then it would be a good idea to see if it still works after removing half the connections.

## 2.3. Understanding the Error Value

The mean square error value is calculated while the ANN is being trained. Some functions are implemented, to use and manipulate this error value. The `fann_get_MSE` function returns the error value and the `fann_reset_MSE` resets the error value. The following explains how the mean square error value is calculated, to give an idea of the value's ability to reveal the quality of the training.

If  $d$  is the desired output of an output neuron and  $y$  is the actual output of the neuron, the square error is  $(d - y)$  squared. If two output neurons exists, then the mean square error for these two neurons is the average of the two square errors.

When training with the `fann_train_on_file` function, an error value is printed. This error value is the mean square error for all the training data. Meaning that it is the average of all the square errors in each of the training pairs.

## 2.4. Training and Testing

Normally it will be sufficient to use the `fann_train_on_file` training function, but sometimes you want to have more control and you will have to write a custom training loop. This could be because you would like another stop criteria, or because you would like to adjust some of the parameters during training. Another stop criteria than the value of the combined mean square error could be that each of the training pairs should have a mean square error lower than a given value.

**Example 2-1. The internals of the `fann_train_on_file` function, without writing the status line.**

```
struct fann_train_data *data = fann_read_train_from_file(filename);
for(i = 1 ; i <= max_epochs ; i++) {
    fann_reset_MSE(ann);
```

```

for (j = 0 ; j != data->num_data ; j++) {
    fann_train(ann, data->input[j], data->output[j]);
}
if ( fann_get_MSE(ann) < desired_error ) {
    break;
}
}
fann_destroy_train(data);

```

This piece of code introduces the `fann_train` function, which trains the ANN for one iteration with one pair of inputs and outputs and also updates the mean square error. The `fann_train_data` structure is also introduced, this structure is a container for the training data in the file described in figure 10. The structure can be used to train the ANN, but it can also be used to test the ANN with data which it has not been trained with.

**Example 2-2. Test all of the data in a file and calculates the mean square error.**

```

struct fann_train_data *data = fann_read_train_from_file(filename);
fann_reset_MSE(ann);
for(i = 0 ; i != data->num_data ; i++ ) {
    fann_test(ann, data->input[i], data->output[i]);
}
printf("Mean Square Error: %f\n", fann_get_MSE(ann));
fann_destroy_train(data);

```

This piece of code introduces another useful function: `fann_test` function, which takes an input array and a desired output array as the parameters and returns the calculated output. It also updates the mean square error.

## 2.5. Avoid Over-Fitting

With the knowledge of how to train and test an ANN, a new approach to training can be introduced. If too much training is applied to a set of data, the ANN will eventually over-fit, meaning that it will be fitted precisely to this set of training data and thereby losing generalization. It is often a good idea to test, how good an ANN performs on data that it has not seen before. Testing with data not seen before, can be done while training, to see how much training is required in order to perform well without over-fitting. The testing can either be done by hand, or an automatic test can be applied, which stops the training when the mean square error of the test data is not improving anymore.

## 2.6. Adjusting Parameters During Training

If a very low mean square error is required it can sometimes be a good idea to gradually decrease the learning rate during training, in order to make the adjusting of weights more subtle. If more precision is required, it might also be a good idea to use double precision floats instead of standard floats.

The threshold activation function is faster than the sigmoid function, but since it is not possible to train with this function, you may wish to consider an alternate approach:

While training the ANN you could slightly increase the steepness parameter of the sigmoid function. This would make the sigmoid function more steep and make it look more like the threshold function. After this training session you could set the activation function to the threshold function and the ANN would work with this activation function. This approach will not work on all kinds of problems, and has been successfully tested on the XOR function.

## Chapter 3. Fixed Point Usage

It is possible to run the ANN with fixed point numbers (internally represented as integers). This option is only intended for use on computers with no floating point processor, for example, the iPAQ, but a minor performance enhancement can also be seen on most modern computers [IDS, 2000].

### 3.1. Training a Fixed Point ANN

The ANN cannot be trained in fixed point, which is why the training part is basically the same as for floating point numbers. The only difference is that you should save the ANN as fixed point. This is done by the `fann_save_to_fixed` function. This function saves a fixed point version of the ANN, but it also does some analysis, in order to find out where the decimal point should be. The result of this analysis is returned from the function.

The decimal point returned from the function is an indicator of, how many bits is used for the fractional part of the fixed point numbers. If this number is negative, there will most likely be integer overflow when running the library with fixed point numbers and this should be avoided. Furthermore, if the decimal point is too low (e.g. lower than 5), it is probably not a good idea to use the fixed point version.

Please note, that the inputs to networks that should be used in fixed point should be between -1 and 1.

#### **Example 3-1. An example of a program written to support training in both fixed point and floating point numbers**

```
#include "fann.h"
#include <stdio.h>

int main()
{
    fann_type *calc_out;
    const float connection_rate = 1;
    const float learning_rate = 0.7;
    const unsigned int num_input = 2;
    const unsigned int num_output = 1;
    const unsigned int num_layers = 3;
    const unsigned int num_neurons_hidden = 4;
    const float desired_error = 0.001;
    const unsigned int max_iterations = 20000;
    const unsigned int iterations_between_reports = 100;
    struct fann *ann;
    struct fann_train_data *data;

    unsigned int i = 0;
    unsigned int decimal_point;

    printf("Creating network.\n");

    ann = fann_create(connection_rate, learning_rate, num_layers,
num_input,
```

```

num_neurons_hidden,
num_output);

printf("Training network.\n");

data = fann_read_train_from_file("xor.data");

fann_train_on_data(ann, data, max_iterations, iterations_between_reports, desired_error);

printf("Testing network.\n");

for(i = 0; i < data->num_data; i++){
calc_out = fann_run(ann, data->input[i]);
printf("XOR test (%f,%f) -> %f, should be %f, difference=%f\n",
data->input[i][0], data->input[i][1], *calc_out, data->output[i][0], fann_abs(*calc_out - data->output[i][0]));
}

printf("Saving network.\n");

fann_save(ann, "xor_float.net");

decimal_point = fann_save_to_fixed(ann, "xor_fixed.net");
fann_save_train_to_fixed(data, "xor_fixed.data", decimal_point);

printf("Cleaning up.\n");
fann_destroy_train(data);
fann_destroy(ann);

return 0;
}

```

## 3.2. Running a Fixed Point ANN

Running a fixed point ANN is done much like running an ordinary ANN. The difference is that the inputs and outputs should be in fixed point representation. Furthermore the inputs should be restricted to be between *-multiplier* and *multiplier* to avoid integer overflow, where the *multiplier* is the value returned from `fann_get_multiplier`. This multiplier is the value that a floating point number should be multiplied with, in order to be a fixed point number, likewise the output of the ANN should be divided by this multiplier in order to be between zero and one.

To help using fixed point numbers, another function is provided. `fann_get_decimal_point` which returns the decimal point. The decimal point is the position dividing the integer and fractional part of the fixed point number and is useful for doing operations on the fixed point inputs and outputs.

**Example 3-2. An example of a program written to support both fixed point and floating point numbers**

```

#include <time.h>
#include <sys/time.h>
#include <stdio.h>

```

```

#include "fann.h"

int main()
{
    fann_type *calc_out;
    unsigned int i;
    int ret = 0;

    struct fann *ann;
    struct fann_train_data *data;

    printf("Creating network.\n");

#ifdef FIXEDFANN
    ann = fann_create_from_file("xor_fixed.net");
#else
    ann = fann_create_from_file("xor_float.net");
#endif

    if(!ann){
        printf("Error creating ann --- ABORTING.\n");
        return 0;
    }

    printf("Testing network.\n");

#ifdef FIXEDFANN
    data = fann_read_train_from_file("xor_fixed.data");
#else
    data = fann_read_train_from_file("xor.data");
#endif

    for(i = 0; i < data->num_data; i++){
        fann_reset_MSE(ann);
        calc_out = fann_test(ann, data->input[i], data->output[i]);
#ifdef FIXEDFANN
        printf("XOR test (%d, %d) -> %d, should be %d, difference=%f\n",
            data->input[i][0], data->input[i][1], *calc_out, data->output[i][0], (float)fann_abs(*calc_out -
            if((float)fann_abs(*calc_out - data->output[i][0])/fann_get_multiplier(ann) > 0.1){
                printf("Test failed\n");
                ret = -1;
            }
#else
        printf("XOR test (%f, %f) -> %f, should be %f, difference=%f\n",
            data->input[i][0], data->input[i][1], *calc_out, data->output[i][0], (float)fann_abs(*calc_out -
#endif
    }

    printf("Cleaning up.\n");
    fann_destroy_train(data);
    fann_destroy(ann);

    return ret;
}

```

### 3.3. Precision of a Fixed Point ANN

The fixed point ANN is not as precise as a floating point ANN, furthermore it approximates the sigmoid function by a stepwise linear function. Therefore, it is always a good idea to test the fixed point ANN after loading it from a file. This can be done by calculating the mean square error as described earlier. There is, however, one problem with this approach: The training data stored in the file is in floating point format. Therefore, it is possible to save this data in a fixed point format from within the floating point program. This is done by the function `fann_save_train_to_fixed`. Please note that this function takes the decimal point as an argument, meaning that the decimal point should be calculated first by using the `fann_save_to_fixed` function.

# Chapter 4. Neural Network Theory

This section will briefly explain the theory of neural networks (hereafter known as NN) and artificial neural networks (hereafter known as ANN). For a more in depth explanation of these concepts please consult the literature; [Hassoun, 1995] has good coverage of most concepts of ANN and [Hertz et al., 1991] describes the mathematics of ANN very thoroughly, while [Anderson, 1995] has a more psychological and physiological approach to NN and ANN. For the pragmatic I (SN) could recommend [Tettamanzi and Tomassini, 2001], which has a short and easily understandable introduction to NN and ANN.

## 4.1. Neural Networks

The human brain is a highly complicated machine capable of solving very complex problems. Although we have a good understanding of some of the basic operations that drive the brain, we are still far from understanding everything there is to know about the brain.

In order to understand ANN, you will need to have a basic knowledge of how the internals of the brain work. The brain is part of the central nervous system and consists of a very large NN. The NN is actually quite complicated, so the following discussion shall be relegated to the details needed to understand ANN, in order to simplify the explanation.

The NN is a network consisting of connected neurons. The center of the neuron is called the nucleus. The nucleus is connected to other nucleuses by means of the dendrites and the axon. This connection is called a synaptic connection.

The neuron can fire electric pulses through its synaptic connections, which is received at the dendrites of other neurons.

When a neuron receives enough electric pulses through its dendrites, it activates and fires a pulse through its axon, which is then received by other neurons. In this way information can propagate through the NN. The synaptic connections change throughout the lifetime of a neuron and the amount of incoming pulses needed to activate a neuron (the threshold) also change. This behavior allows the NN to learn.

The human brain consists of around  $10^{11}$  neurons which are highly interconnected with around  $10^{15}$  connections [Tettamanzi and Tomassini, 2001]. These neurons activate in parallel as an effect to internal and external sources. The brain is connected to the rest of the nervous system, which allows it to receive information by means of the five senses and also allows it to control the muscles.

## 4.2. Artificial Neural Networks

It is not possible (at the moment) to make an artificial brain, but it is possible to make simplified artificial neurons and artificial neural networks. These ANNs can be made in many different ways and can try to mimic the brain in many different ways.

ANNs are not intelligent, but they are good for recognizing patterns and making simple rules for complex problems. They also have excellent training capabilities which is why they are often used in artificial intelligence research.

ANNs are good at generalizing from a set of training data. E.g. this means an ANN given data about a set of animals connected to a fact telling if they are mammals or not, is able to predict whether an animal outside the original set is a mammal from its data. This is a very desirable feature of ANNs, because you do not need to know the characteristics defining a mammal, the ANN will find out by itself.

### 4.3. Training an ANN

When training an ANN with a set of input and output data, we wish to adjust the weights in the ANN, to make the ANN give the same outputs as seen in the training data. On the other hand, we do not want to make the ANN too specific, making it give precise results for the training data, but incorrect results for all other data. When this happens, we say that the ANN has been over-fitted.

The training process can be seen as an optimization problem, where we wish to minimize the mean square error of the entire set of training data. This problem can be solved in many different ways, ranging from standard optimization heuristics like simulated annealing, through more special optimization techniques like genetic algorithms to specialized gradient descent algorithms like backpropagation.

The most used algorithm is the backpropagation algorithm, but this algorithm has some limitations concerning, the extent of adjustment to the weights in each iteration. This problem has been solved in more advanced algorithms like RPROP [Riedmiller and Braun, 1993] and quickprop [Fahlman, 1988].

# Chapter 5. API Reference

This is a list of all functions and structures in FANN.

## 5.1. Creation, Destruction, and Execution

### **fann\_create**

#### **Name**

`fann_create` — Create a new artificial neural network, and return a pointer to it.

#### **Description**

```
struct fann * fann_create(float connection_rate, float learning_rate, unsigned int num_layers,
```

`fann_create` will create a new artificial neural network, and return a pointer to it. The *connection\_rate* controls how many connections there will be in the network. If the connection rate is set to 1, the network will be fully connected, but if it is set to 0.5 only half of the connections will be set.

The *num\_layers* is the number of layers including the input and output layer. This parameter is followed by one parameter for each layer telling how many neurons there should be in the layer.

This function appears in FANN >= 1.0.0.

### **fann\_create\_array**

#### **Name**

`fann_create_array` — Create a new artificial neural network, and return a pointer to it.

#### **Description**

```
struct fann * fann_create_array(float connection_rate, float learning_rate, unsigned int num_la
```

`fann_create_array` will create a new artificial neural network, and return a pointer to it. It is the same as `fann_create`, only it accepts an array as its final parameter instead of variable arguments.

**Example 5-1. fann\_create\_array example**

```

unsigned int neurons_per_layer[3] = {2, 3, 1};

// The following two calls have identical results
struct fann * ann = fann_create_array(1.0f, 0.7f, 3, neurons_per_layer);
struct fann * ann2 = fann_create(1.0f, 0.7f, 3, 2, 3, 1);

fann_destroy(ann);
fann_destroy(ann2);

```

This function appears in FANN >= 1.0.5.

**fann\_create\_shortcut****Name**

`fann_create_shortcut` — Create a new artificial neural network with shortcut connections, and return a pointer to it.

**Description**

```
struct fann * fann_create_shortcut(float learning_rate, unsigned int num_layers, unsigned int .
```

`fann_create_shortcut` will create a new artificial neural network, and return a pointer to it. The network will be fully connected, and will furthermore have all shortcut connections connected.

Shortcut connections are connections that skip layers. A fully connected network with shortcut connections, is a network where all neurons are connected to all neurons in later layers. Including direct connections from the input layer to the output layer.

The `num_layers` is the number of layers including the input and output layer. This parameter is followed by one parameter for each layer telling how many neurons there should be in the layer.

This function appears in FANN >= 1.2.0.

## **fann\_create\_shortcut\_array**

### **Name**

`fann_create_shortcut_array` — Create a new artificial neural network with shortcut connections, and return a pointer to it.

### **Description**

```
struct fann * fann_create_shortcut_array(float learning_rate, unsigned int num_layers, unsigned
```

`fann_create_shortcut_array` will create a new artificial neural network, and return a pointer to it. It is the same as `fann_create_shortcut`, only it accepts an array as its final parameter instead of variable arguments.

This function appears in FANN  $\geq$  1.2.0.

## **fann\_destroy**

### **Name**

`fann_destroy` — Destroy an ANN.

### **Description**

```
void fann_destroy(struct fann * ann);
```

`fann_destroy` will destroy an artificial neural network, properly freeing all associate memory.

This function appears in FANN  $\geq$  1.0.0.

## **fann\_run**

### **Name**

`fann_run` — Run (execute) an ANN.

## Description

```
fann_type * fann_run(struct fann * ann, fann_type * input);
```

`fann_run` will run `input` through `ann`, returning an array of outputs, the number of which being equal to the number of neurons in the output layer.

This function appears in FANN >= 1.0.0.

## fann\_randomize\_weights

### Name

`fann_randomize_weights` — Give each connection a random weight.

### Description

```
void fann_randomize_weights(struct fann * ann, fann_type min_weight, fann_type max_weight);
```

Randomizes the weight of each connection in `ann`, effectively resetting the network.

See also: *Adjusting Parameters*, `fann_init_weights`

This function appears in FANN >= 1.0.0.

## fann\_init\_weights

### Name

`fann_init_weights` — Initialize the weight of each connection.

### Description

```
void fann_init_weights(struct fann * ann, struct fann_train_data * train_data);
```

This function behaves similarly to `fann_randomize_weights`. It will use the algorithm developed by Derrick Nguyen and Bernard Widrow [*Nguyen and Widrow, 1990*] to set the weights in such a way as to speed up training. This technique is not always successful, and in some cases can be *less* efficient than a purely random initialization.

The algorithm requires access to the range of the input data (ie, largest and smallest input), and therefore accepts a second argument, *data*, which is the training data that will be used to train the network.

See also: *Adjusting Parameters*, `fann_randomize_weights`

This function appears in FANN >= 1.1.0.

## fann\_print\_connections

### Name

`fann_print_connections` — Prints the connections of an ann.

### Description

```
void fann_print_connections(struct fann * ann);
```

`fann_print_connections` will print the connections of the ann in a compact matrix, for easy viewing of the internals of the ann.

The output from `fann_print_connections` on a small (2 2 1) network trained on the xor problem:

```
Layer / Neuron 012345
L 1 / N 3 ddb...
L 1 / N 4 bbb...
L 2 / N 6 ...cda
```

This network have five real neurons and two bias neurons. This gives a total of seven neurons named from 0 to 6. The connections between these neurons can be seen in the matrix. "." is a place where there is no connection, while a character tells how strong the connection is on a scale from a-z. The two real neurons in the hidden layer (neuron 3 and 4 in layer 1) has connection from the three neurons in the previous layer as is visible in the first two lines. The output neuron (6) has connections form the three neurons in the hidden layer 3 - 5 as is visible in the last line.

To simplify the matrix output neurons is not visible as neurons that connections can come from, and input and bias neurons are not visible as neurons that connections can go to.

This function appears in FANN >= 1.2.0.

## 5.2. Input/Output

### fann\_save

#### Name

fann\_save — Save an ANN to a file.

#### Description

```
void fann_save(struct fann * ann, const char * configuration_file);
```

fann\_save will attempt to save *ann* to the file located at *configuration\_file*

This function appears in FANN >= 1.0.0.

### fann\_save\_to\_fixed

#### Name

fann\_save\_to\_fixed — Save an ANN to a fixed-point file.

#### Description

```
void fann_save_to_fixed(struct fann * ann, const char * configuration_file);
```

fann\_save\_to\_fixed will attempt to save *ann* to the file located at *configuration\_file* as a fixed-point network.

This is useful for training a network in floating points, and then later executing it in fixed point.

The function returns the bit position of the fix point, which can be used to find out how accurate the fixed point network will be. A high value indicates high precision, and a low value indicates low precision.

A negative value indicates very low precision, and a very strong possibility for overflow. (the actual fix point will be set to 0, since a negative fix point does not make sense).

Generally, a fix point lower than 6 is bad, and should be avoided. The best way to avoid this, is to have less connections to each neuron, or just less neurons in each layer.

The fixed point use of this network is only intended for use on machines that have no floating point processor, like an iPAQ. On normal computers the floating point version is actually faster.

This function appears in FANN  $\geq$  1.0.0.

## **fann\_create\_from\_file**

### **Name**

`fann_create_from_file` — Load an ANN from a file.

### **Description**

```
struct fann * fann_create_from_file(const char * configuration_file);
```

`fann_create_from_file` will attempt to load an artificial neural network from a file.

This function appears in FANN  $\geq$  1.0.0.

## **5.3. Training**

### **fann\_train**

#### **Name**

`fann_train` — Train an ANN.

#### **Description**

```
void fann_train(struct fann * ann, fann_type * input, fann_type * output);
```

`fann_train` will train one iteration with a set of inputs, and a set of desired outputs. The training will be done by the standard backpropagation algorithm.

This function appears in FANN  $\geq$  1.0.0.

## fann\_test

### Name

fann\_test — Tests an ANN.

### Description

```
fann_type * fann_test(struct fann * ann, fann_type * input, fann_type * desired_output);
```

Test with a set of inputs, and a set of desired outputs. This operation updates the mean square error, but does not change the network in any way.

This function appears in FANN >= 1.0.0.

## fann\_get\_MSE

### Name

fann\_get\_MSE — Return the mean square error of an ANN.

### Description

```
float fann_get_MSE(struct fann * ann);
```

Reads the mean square error from the network. This value is calculated during training or testing, and can therefore sometimes be a bit off if the weights have been changed since the last calculation of the value.

This function appears in FANN >= 1.1.0. (before this fann\_get\_error is used)

## fann\_reset\_MSE

### Name

fann\_reset\_MSE — Reset the mean square error of an ANN.

## Description

```
void fann_reset_MSE(struct fann * ann);
```

Resets the mean square error from the network.

This function appears in FANN  $\geq$  1.1.0. (before this `fann_reset_error` is used)

## 5.4. Training Data

### `fann_read_train_from_file`

#### Name

`fann_read_train_from_file` — Read training data from a file.

#### Description

```
struct fann_train_data * fann_read_train_from_file(char * filename);
```

`fann_read_train_from_file` will load training data from a file. The file should be formatted in the following way:

```
num_train_data num_input num_output
inputdata seperated by space
outputdata seperated by space

.
.
.

inputdata seperated by space
outputdata seperated by space
```

An example of a properly formatted file is provided in the Introduction.

This function appears in FANN  $\geq$  1.0.0.

## fann\_save\_train

### Name

fann\_save\_train — Save training data.

### Description

```
void fann_save_train(struct data * train_data, FILE * filename);
```

Save *train\_data* to *filename*.

This function appears in FANN >= 1.0.0.

## fann\_save\_train\_to\_fixed

### Name

fann\_save\_train\_to\_fixed — Save training data as fixed point.

### Description

```
void fann_save_to_fixed(struct data * train_data, FILE * filename, unsigned int decimal_point);
```

Save *train\_data* as fixed point to *filename*.

This function appears in FANN >= 1.0.0.

## fann\_destroy\_train

### Name

fann\_destroy\_train — Destroy training data.

### Description

```
void fann_destroy_train_data(struct fann_train_data * train_data);
```

Destroy the training data stored in *train\_data*, freeing the associated memory.

This function appears in FANN  $\geq$  1.0.0.

## fann\_train\_epoch

### Name

fann\_train\_epoch — Trains one epoch.

### Description

```
float fann_train_epoch(struct fann * ann, struct fann_train_data * data);
```

Train one epoch with the training data stored in *data*. One epoch is where all of the training data is considered exactly once.

This function returns the MSE error as it is calculated either before or during the actual training. This is not the actual MSE after the training epoch, but since calculating this will require to go through the entire training set once more, it is more than adequate to use this value during training.

The training algorithm used by this function is chosen by the `fann_set_training_algorithm` function. The default training algorithm is `FANN_TRAIN_RPROP`.

This function appears in FANN  $\geq$  1.2.0.

## fann\_test\_data

### Name

fann\_test\_data — Calculates the mean square error for a set of data.

### Description

```
float fann_test_data(struct fann * ann, struct fann_train_data * data);
```

Calculates the mean square error for a set of data.

This function appears in FANN  $\geq$  1.2.0.

## fann\_train\_on\_data

### Name

fann\_train\_on\_data — Train an ANN.

### Description

```
void fann_train_on_data(struct fann * ann, struct fann_train_data * data, unsigned int max_epochs)
```

Trains *ann* using *data* until *desired\_error* is reached, or until *max\_epochs* is surpassed.

The training algorithm used by this function is chosen by the `fann_set_training_algorithm` function. The default training algorithm is `FANN_TRAIN_RPROP`.

This function appears in FANN  $\geq$  1.0.0.

## fann\_train\_on\_data\_callback

### Name

fann\_train\_on\_data\_callback — Train an ANN.

### Description

```
void fann_train_on_data_callback(struct fann * ann, struct fann_train_data * data, unsigned int
```

Trains *ann* using *data* until *desired\_error* is reached, or until *max\_epochs* is surpassed.

This function behaves identically to `fann_train_on_data`, except that `fann_train_on_data_callback` allows you to specify a function to be called every *epochs\_between\_reports* instead of using the default reporting mechanism. If the callback function returns -1 the training will terminate.

The callback function is very useful in GUI applications or in other applications which do not wish to report the progress on standard output. Furthermore the callback function can be used to stop the training at non standard stop criteria (see *Training and Testing*.)

This function appears in FANN  $\geq$  1.0.5.

The training algorithm used by this function is chosen by the `fann_set_training_algorithm` function. The default training algorithm is `FANN_TRAIN_RPROP`.

## fann\_train\_on\_file

### Name

fann\_train\_on\_file — Train an ANN.

### Description

```
void fann_train_on_file(struct fann * ann, char * filename, unsigned int max_epochs, unsigned i
```

Trains *ann* using the data in *filename* until *desired\_error* is reached, or until *max\_epochs* is surpassed.

The training algorithm used by this function is chosen by the `fann_set_training_algorithm` function. The default training algorithm is `FANN_TRAIN_RPROP`.

This function appears in FANN >= 1.0.0.

## fann\_train\_on\_file\_callback

### Name

fann\_train\_on\_file\_callback — Train an ANN.

### Description

```
void fann_train_on_file_callback(struct fann * ann, char * filename, unsigned int max_epochs, u
```

Trains *ann* using the data in *filename* until *desired\_error* is reached, or until *max\_epochs* is surpassed.

This function behaves identically to `fann_train_on_file`, except that `fann_train_on_file_callback` allows you to specify a function to be called every *epochs\_between\_reports* instead of using the default reporting mechanism. The callback function works as described in `fann_train_on_data_callback`

The training algorithm used by this function is chosen by the `fann_set_training_algorithm` function. The default training algorithm is `FANN_TRAIN_RPROP`.

This function appears in FANN >= 1.0.5.

## fann\_shuffle\_train\_data

### Name

`fann_shuffle_train_data` — Shuffle the training data.

### Description

```
void fann_shuffle_train_data(struct fann_train_data * data);
```

`fann_shuffle_train_data` will randomize the order of the training data contained in `data`.

This function appears in FANN  $\geq$  1.1.0.

## fann\_merge\_train\_data

### Name

`fann_merge_train_data` — Merge two sets of training data.

### Description

```
struct fann_train_data * fann_merge_train_data(struct fann_train_data * data1, struct fann_train_data * data2);
```

`fann_merge_train_data` will return a single set of training data which contains all data from `data1` and `data2`.

This function appears in FANN  $\geq$  1.1.0.

## fann\_duplicate\_train\_data

### Name

`fann_duplicate_train_data` — Copies a set of training data.

## Description

```
struct fann_train_data * fann_duplicate_train_data(struct fann_train_data * data);
```

`fann_duplicate_train_data` will return a copy of `data`.

This function appears in FANN >= 1.1.0.

## 5.5. Options

### `fann_print_parameters`

#### Name

`fann_print_parameters` — Prints all of the parameters and options of the ANN.

#### Description

```
void fann_print_parameters(struct fann * ann);
```

Prints all the parameters of the network, for easy viewing of all the values.

This function appears in FANN >= 1.2.0.

### `fann_get_training_algorithm`

#### Name

`fann_get_training_algorithm` — Retrieve training algorithm from a network.

#### Description

```
unsigned int fann_get_training_algorithm(struct fann * ann);
```

Return the training algorithm (as described in Training algorithms) for a given network.

The default training algorithm is `FANN_TRAIN_RPROP`.

This function appears in FANN  $\geq$  1.2.0.

## **fann\_set\_training\_algorithm**

### **Name**

`fann_set_training_algorithm` — Set a network's training algorithm.

### **Description**

```
void fann_set_training_algorithm(struct fann * ann, unsigned int training_algorithm);
```

Set the training algorithm (as described in Training algorithms) of a network.

The default training algorithm is `FANN_TRAIN_RPROP`.

This function appears in FANN  $\geq$  1.2.0.

## **fann\_get\_learning\_rate**

### **Name**

`fann_get_learning_rate` — Retrieve learning rate from a network.

### **Description**

```
float fann_get_learning_rate(struct fann * ann);
```

Return the learning rate for a given network.

This function appears in FANN  $\geq$  1.0.0.

## fann\_set\_learning\_rate

### Name

fann\_set\_learning\_rate — Set a network's learning rate.

### Description

```
void fann_set_learning_rate(struct fann * ann, float learning_rate);
```

Set the learning rate of a network.

This function appears in FANN >= 1.0.0.

## fann\_get\_activation\_function\_hidden

### Name

fann\_get\_activation\_function\_hidden — Get the activation function used in the hidden layers.

### Description

```
unsigned int fann_get_activation_function_hidden(struct fann * ann);
```

Return the activation function used in the hidden layers.

See *Activation Functions* for details on the activation functions.

This function appears in FANN >= 1.0.0.

## fann\_set\_activation\_function\_hidden

### Name

fann\_set\_activation\_function\_hidden — Set the activation function for the hidden layers.

## Description

```
fann_set_activation_function_hidden(struct fann * ann, unsigned int activation_function);
```

Set the activation function used in the hidden layers to *activation\_function*.

See *Activation Functions* for details on the activation functions.

This function appears in FANN >= 1.0.0.

## fann\_get\_activation\_function\_output

### Name

fann\_get\_activation\_function\_output — Get the activation function of the output layer.

### Description

```
unsigned int fann_get_activation_function_output(struct fann * ann);
```

Return the activation function of the output layer.

See *Activation Functions* for details on the activation functions.

This function appears in FANN >= 1.0.0.

## fann\_set\_activation\_function\_output

### Name

fann\_set\_activation\_function\_output — Set the activation function for the output layer.

### Description

```
void fann_set_activation_function_output(struct fann * ann, unsigned int activation_function);
```

Set the activation function of the output layer to *activation\_function*.

See *Activation Functions* for details on the activation functions.

This function appears in FANN  $\geq$  1.0.0.

## fann\_get\_activation\_steepness\_hidden

### Name

`fann_get_activation_steepness_hidden` — Retrieve the steepness of the activation function of the hidden layers.

### Description

```
fann_type fann_get_activation_steepness_hidden(struct fann * ann);
```

Return the steepness of the activation function of the hidden layers.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function appears in FANN  $\geq$  1.2.0. and replaces the `fann_get_activation_hidden_steepness` function from FANN  $\geq$  1.0.0.

## fann\_set\_activation\_steepness\_hidden

### Name

`fann_set_activation_steepness_hidden` — Set the steepness of the activation function of the hidden layers.

### Description

```
void fann_set_activation_steepness_hidden(struct fann * ann, fann_type steepness);
```

Set the steepness of the activation function of the hidden layers of *ann* to *steepness*.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function appears in FANN  $\geq$  1.2.0. and replaces the `fann_set_activation_hidden_steepness` function from FANN  $\geq$  1.0.0.

## fann\_get\_activation\_steepness\_output

### Name

`fann_get_activation_steepness_output` — Retrieve the steepness of the activation function of the output layer.

### Description

```
fann_type fann_get_activation_steepness_output(struct fann * ann);
```

Return the steepness of the activation function of the hidden layers.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function appears in FANN  $\geq$  1.2.0. and replaces the `fann_get_activation_output_steepness` function from FANN  $\geq$  1.0.0.

## fann\_set\_activation\_steepness\_output

### Name

`fann_set_activation_steepness_output` — Set the steepness of the activation function of the output layer.

### Description

```
void fann_set_activation_steepness_output(struct fann * ann, fann_type steepness);
```

Set the steepness of the activation function of the hidden layers of *ann* to *steepness*.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function appears in FANN  $\geq$  1.2.0. and replaces the `fann_set_activation_output_steepness` function from FANN  $\geq$  1.0.0.

## **fann\_set\_train\_error\_function**

### **Name**

`fann_set_train_error_function` — Sets the training error function to be used.

### **Description**

```
void fann_set_train_error_function(struct fann * ann, unsigned int train_error_function);
```

Set the training error function (as described in Training Error Functions) of a network.

The default training error function is `FANN_ERRORFUNC_TANH`.

This function appears in FANN  $\geq$  1.2.0.

## **fann\_get\_train\_error\_function**

### **Name**

`fann_get_train_error_function` — Gets the training error function to be used.

### **Description**

```
unsigned int fann_get_train_error_function(struct fann * ann);
```

Get the training error function (as described in Training Error Functions) of a network.

The default training error function is `FANN_ERRORFUNC_TANH`.

This function appears in FANN  $\geq$  1.2.0.

## fann\_get\_quickprop\_decay

### Name

fann\_get\_quickprop\_decay — Get the decay parameter used by the quickprop training.

### Description

```
float fann_get_quickprop_decay(struct fann * ann);
```

The decay is a small negative valued number which is the factor that the weights should become smaller in each iteration. This is used to make sure that the weights do not become too high during training.

The default value for this parameter is -0.0001.

This function appears in FANN >= 1.2.0.

## fann\_set\_quickprop\_decay

### Name

fann\_set\_quickprop\_decay — Set the decay parameter used by the quickprop training.

### Description

```
void fann_set_quickprop_decay(struct fann * ann, float quickprop_decay);
```

The decay is a small negative valued number which is the factor that the weights should become smaller in each iteration. This is used to make sure that the weights do not become too high during training.

The default value for this parameter is -0.0001.

This function appears in FANN >= 1.2.0.

## fann\_get\_quickprop\_mu

### Name

fann\_get\_quickprop\_mu — Get the mu factor used by quickprop training.

## Description

```
float fann_get_quickprop_mu(struct fann * ann);
```

The mu factor is used to increase and decrease the step-size during quickprop training. The mu factor should always be above 1, since it would otherwise decrease the step-size when it was suppose to increase it.

The default value for this parameter is 1.75.

This function appears in FANN >= 1.2.0.

## fann\_set\_quickprop\_mu

### Name

fann\_set\_quickprop\_mu — Set the mu factor used by quickprop training.

### Description

```
void fann_set_quickprop_mu(struct fann * ann, float quickprop_mu);
```

The mu factor is used to increase and decrease the step-size during quickprop training. The mu factor should always be above 1, since it would otherwise decrease the step-size when it was suppose to increase it.

The default value for this parameter is 1.75.

This function appears in FANN >= 1.2.0.

## fann\_get\_rprop\_increase\_factor

### Name

fann\_get\_rprop\_increase\_factor — Get the increase factor used by RPROP training.

### Description

```
float fann_get_rprop_increase_factor(struct fann * ann);
```

The increase factor is a value larger than 1, which is used to increase the step-size during RPROP training.

The default value for this parameter is 1.2.

This function appears in FANN  $\geq$  1.2.0.

## fann\_set\_rprop\_increase\_factor

### Name

`fann_set_rprop_increase_factor` — Get the increase factor used by RPROP training.

### Description

```
void fann_set_rprop_increase_factor(struct fann * ann, float rprop_increase_factor);
```

The increase factor is a value larger than 1, which is used to increase the step-size during RPROP training.

The default value for this parameter is 1.2.

This function appears in FANN  $\geq$  1.2.0.

## fann\_get\_rprop\_decrease\_factor

### Name

`fann_get_rprop_decrease_factor` — Get the decrease factor used by RPROP training.

### Description

```
float fann_get_rprop_decrease_factor(struct fann * ann);
```

The increase factor is a value smaller than 1, which is used to decrease the step-size during RPROP training.

The default value for this parameter is 0.5.

This function appears in FANN  $\geq$  1.2.0.

## fann\_set\_rprop\_decrease\_factor

### Name

fann\_set\_rprop\_decrease\_factor — Set the decrease factor used by RPROP training.

### Description

```
void fann_set_rprop_decrease_factor(struct fann * ann, float rprop_decrease_factor);
```

The increase factor is a value smaller than 1, which is used to decrease the step-size during RPROP training.

The default value for this parameter is 0.5.

This function appears in FANN  $\geq$  1.2.0.

## fann\_get\_rprop\_delta\_min

### Name

fann\_get\_rprop\_delta\_min — Get the minimum step-size used by RPROP training.

### Description

```
float fann_get_rprop_delta_min(struct fann * ann);
```

The minimum step-size is a small positive number determining how small the minimum step may be.

The default value for this parameter is 0.0.

This function appears in FANN  $\geq$  1.2.0.

## fann\_set\_rprop\_delta\_min

### Name

fann\_set\_rprop\_delta\_min — Set the minimum step-size used by RPROP training.

## Description

```
void fann_set_rprop_delta_min(struct fann * ann, float rprop_delta_min);
```

The minimum step-size is a small positive number determining how small the minimum step may be.

The default value for this parameter is 0.0.

This function appears in FANN >= 1.2.0.

## fann\_get\_rprop\_delta\_max

### Name

fann\_get\_rprop\_delta\_max — Get the maximum step-size used by RPROP training.

### Description

```
float fann_get_rprop_delta_max(struct fann * ann);
```

The maximum step-size is a small positive number determining how small the minimum step may be.

The default value for this parameter is 50.0.

This function appears in FANN >= 1.2.0.

## fann\_set\_rprop\_delta\_max

### Name

fann\_set\_rprop\_delta\_max — Set the maximum step-size used by RPROP training.

### Description

```
void fann_set_rprop_delta_max(struct fann * ann, float rprop_delta_max);
```

The maximum step-size is a small positive number determining how small the minimum step may be.

The default value for this parameter is 50.0.

This function appears in FANN  $\geq$  1.2.0.

## **fann\_get\_num\_input**

### **Name**

`fann_get_num_input` — Get the number of neurons in the input layer.

### **Description**

```
unsigned int fann_get_num_input(struct fann * ann);
```

Return the number of neurons in the input layer of *ann*.

This function appears in FANN  $\geq$  1.0.0.

## **fann\_get\_num\_output**

### **Name**

`fann_get_num_output` — Get number of neurons in the output layer.

### **Description**

```
unsigned int fann_get_num_output(struct fann * ann);
```

Return the number of neurons in the output layer of *ann*.

This function appears in FANN  $\geq$  1.0.0.

## **fann\_get\_total\_neurons**

### **Name**

`fann_get_total_neurons` — Get the total number of neurons in a network.

## Description

```
unsigned int fann_get_total_neurons(struct fann * ann);
```

Return the total number of neurons in *ann*. This number includes the bias neurons.

This function appears in FANN >= 1.0.0.

## fann\_get\_total\_connections

### Name

`fann_get_total_connections` — Get the total number of connections in a network.

### Description

```
unsigned int fann_get_total_connections(struct fann * ann);
```

Return the total number of connections in *ann*.

This function appears in FANN >= 1.0.0.

## fann\_get\_decimal\_point

### Name

`fann_get_decimal_point` — Get the position of the decimal point.

### Description

```
unsigned int fann_get_decimal_point(struct fann * ann);
```

Return the position of the decimal point in *ann*.

This function is only available when the ANN is in fixed point mode.

This function appears in FANN >= 1.0.0.

## fann\_get\_multiplier

### Name

`fann_get_multiplier` — Get the multiplier.

### Description

```
fann_get_multiplier(struct fann * ann);
```

Return the multiplier that fix point data in *annis* multiplied with.

This function is only available when the ANN is in fixed point mode.

This function appears in FANN >= 1.0.0.

## 5.6. Error Handling

### fann\_get\_errno

#### Name

`fann_get_errno` — Return the numerical representation of the last error.

#### Description

```
unsigned int fann_get_errno(struct fann_error * errdat);
```

Returns the numerical representation of the last error. The error codes are defined in `fann_errno.h`.

This function appears in FANN >= 1.1.0.

### fann\_get\_errstr

#### Name

`fann_get_errstr` — Return the last error.

## Description

```
char * fann_get_errstr(struct fann_error * errdat);
```

Returns the last error.

Note: This will reset the network's error- any subsequent calls to `fann_get_errno` or `fann_get_errstr` will yield 0 and NULL, respectively.

This function appears in FANN >= 1.1.0.

## fann\_reset\_errno

### Name

`fann_reset_errno` — Reset the last error number.

### Description

```
void fann_reset_errno(struct fann_error * errdat);
```

Reset the last error number.

This function appears in FANN >= 1.1.0.

## fann\_reset\_errstr

### Name

`fann_reset_errstr` — Reset the last error string.

### Description

```
void fann_reset_errstr(struct fann_error * errdat);
```

Reset the last error string.

This function appears in FANN >= 1.1.0.

## fann\_set\_error\_log

### Name

fann\_set\_error\_log — Set the error log to a file descriptor.

### Description

```
void fann_set_error_log(struct fann_error * errdat, FILE * log);
```

Set the error log to *log*.

The error log defaults to stderr.

This function appears in FANN >= 1.1.0.

## fann\_print\_error

### Name

fann\_print\_error — Print the last error to the error log.

### Description

```
void fann_print_error_log(struct fann * ann);
```

Prints the network's last error to the error log.

The error log defaults to stderr.

This function appears in FANN >= 1.1.0.

## 5.7. Data Structures

### struct fann

#### Name

struct fann — Describes a neural network.

#### Description

This structure is subject to change at any time. If you need to use the values contained herein, please see the Options functions. If these functions do not fulfill your needs, please open a feature request on our SourceForge project page (<http://www.sourceforge.net/projects/fann>).

#### Properties

unsigned int `errno_f`

The type of error that last occurred.

FILE \* `error_log`

Where to log error messages.

char \* `errstr`

A string representation of the last error.

float `learning_rate`

The learning rate of the network.

float `connection_rate`

The connection rate of the network. Between 0 and 1, 1 meaning fully connected.

unsigned int `shortcut_connections`

Is 1 if shortcut connections are used in the ann otherwise 0 Shortcut connections are connections that skip layers. A fully connected ann with shortcut connections is an ann where neurons have connections to all neurons in all later layers.

ANNs with shortcut connections are created by `fann_create_shortcut`.

struct fann\_layer \* `first_layer`

Pointer to the first layer (input layer) in an array of all the layers, including the input and output layer.

struct fann\_layer \* `last_layer`

Pointer to the layer past the last layer in an array of all the layers, including the input and output layer.

`unsigned int total_neurons`

Total number of neurons. Very useful, because the actual neurons are allocated in one long array.

`unsigned int num_input`

Number of input neurons (not calculating bias)

`unsigned int num_output`

Number of output neurons (not calculating bias)

`fann_type * train_errors`

Used to contain the error deltas used during training. Is allocated during first training session, which means that if we do not train, it is never allocated.

`unsigned int activation_function_output`

Used to choose which activation function to use in the output layer.

`unsigned int activation_function_hidden`

Used to choose which activation function to use in the hidden layers.

`unsigned int activation_steepness_hidden`

Parameters for the activation function in the hidden layers.

`unsigned int activation_steepness_output`

Parameters for the activation function in the output layer.

`unsigned int training_algorithm`

Training algorithm used when calling `fann_train_on...` and `fann_train_epoch`.

`unsigned int decimal_point`

*Fixed point only.* The decimal point, used for shifting the fix point in fixed point integer operations.

`unsigned int multiplier`

*Fixed point only.* The multiplier, used for multiplying the fix point in fixed point integer operations. Only used in special cases, since the `decimal_point` is much faster.

`fann_type * activation_results_hidden`

An array of six members used by some activation functions to hold results for the hidden layer(s).

`fann_type * activation_values_hidden`

An array of six members used by some activation functions to hold values for the hidden layer(s).

`fann_type * activation_results_output`

An array of six members used by some activation functions to hold results for the output layer.

`fann_type * activation_values_output`

An array of six members used by some activation functions to hold values for the output layer.

`unsigned int total_connections`

Total number of connections. Very useful, because the actual connections are allocated in one long array.

`fann_type * output`

Used to store outputs in.

`unsigned int num_MSE`

The number of data used to calculate the mean square error.

`float MSE_value`

The total error value. The real mean square error is  $MSE\_value/num\_MSE$ .

`unsigned int train_error_function`

When using this, training is usually faster. Makes the error used for calculating the slopes higher when the difference is higher.

`float quickprop_decay`

Decay is used to make the weights not go so high.

`float quickprop_mu`

Mu is a factor used to increase and decrease the step-size.

`float rprop_increase_factor`

Tells how much the step-size should increase during learning.

`float rprop_decrease_factor`

Tells how much the step-size should decrease during learning.

`float rprop_delta_min`

The minimum step-size.

`float rprop_delta_max`

The maximum step-size.

`fann_type * train_slopes`

Used to contain the slope errors used during batch training. Is allocated during first training session, which means that if we do not train, it is never allocated.

`fann_type * prev_steps`

The previous step taken by the quickprop/rprop procedures. Not allocated if not used.

`fann_type * prev_train_slopes`

The slope values used by the quickprop/rprop procedures. Not allocated if not used.

## struct fann\_train\_data

### Name

struct fann\_train\_data — Describes a set of training data.

### Description

This structure is subject to change at any time. If you need to use the values contained herein, please see the Training Data functions. If these functions do not fulfill your needs, please open a feature request on our SourceForge project page (<http://www.sourceforge.net/projects/fann>).

### Properties

unsigned int `errno_f`

The type of error that last occurred.

FILE \* `error_log`

Where to log error messages.

char \* `errstr`

A string representation of the last error.

unsigned int `num_data`

The number of sets of data in the array.

unsigned int `num_input`

The number of inputs per set of data.

unsigned int `num_output`

The number of outputs per set of data.

fann\_type \*\* `input`

An array of `num_data` elements, each of which contain an array of `num_input` elements, which represent every item of input data.

fann\_type \*\* `output`

An array of `num_data` elements, each of which contain an array of `num_output` elements, which represent every item of output data.

## struct fann\_error

### Name

`struct fann_error` — Describes an error.

### Description

This structure is subject to change at any time. If you need to use the values contained herein, please see the Error Handling functions. If these functions do not fulfill your needs, please open a feature request on our SourceForge project page (<http://www.sourceforge.net/projects/fann>).

You may notice that this structure is identical to the first three properties of the `fann` and `fann_train_data` structures. This is so you can cast each of those structures to `struct fann_error *` when calling the Error Handling functions.

### Properties

`unsigned int errno_f`

The type of error that last occurred.

`FILE * error_log`

Where to log error messages.

`char * errstr`

A string representation of the last error.

## struct fann\_neuron

### Name

`struct fann_neuron` — Describes an individual neuron.

### Description

This structure is subject to change at any time. If you require direct access to the contents of this structure, you may want to consider contacting the FANN development team (<mailto:fann-general@lists.sourceforge.net>).

### Properties

`fann_type * weights`

This property is not yet documented.

```
struct fann_neuron ** connected_neurons
```

This property is not yet documented.

```
unsigned int num_connections
```

This property is not yet documented.

```
fann_type value
```

This property is not yet documented.

## struct fann\_layer

### Name

`struct fann_layer` — Describes a layer in a network.

### Description

This structure is subject to change at any time. If you require direct access to the contents of this structure, you may want to consider contacting the FANN development team (<mailto:fann-general@lists.sourceforge.net>).

### Properties

```
struct fann_neuron * first_neuron
```

A pointer to the first neuron in the layer. When allocated, all the neurons in all the layers are actually in one long array, this is because we want to easily clear all the neurons at once.

```
struct fann_neuron * last_neuron
```

A pointer to the neuron past the last neuron in the layer the number of neurons is `last_neuron - first_neuron`

## 5.8. Constants

### Training algorithms

#### Name

`Training algorithms` — Constants representing training algorithms.

## Description

These constants represent the training algorithms available within the fann library. The list will grow over time, but probably not shrink.

The training algorithm used by this function is chosen by the `fann_set_training_algorithm` function. The default training algorithm is `FANN_TRAIN_RPROP`.

## Constants

### FANN\_TRAIN\_INCREMENTAL

Standard backpropagation algorithm, where the weights are updated after each training pattern. This means that the weights are updated many times during a single epoch. For this reason some problems, will train very fast with this algorithm, while other more advanced problems will not train very well.

### FANN\_TRAIN\_BATCH

Standard backpropagation algorithm, where the weights are updated after calculating the mean square error for the whole training set. This means that the weights are only updated once during a epoch. For this reason some problems, will train slower with this algorithm. But since the mean square error is calculated more correctly than in incremental training, some problems will reach a better solutions with this algorithm.

### FANN\_TRAIN\_RPROP

A more advanced batch training algorithm which achieves good results for many problems. The RPROP training algorithm is adaptive, and does therefore not use the `learning_rate`. Some other parameters can however be set to change the way the RPROP algorithm works, but it is only recommended for users with insight in how the RPROP training algorithm works.

The RPROP training algorithm is described in [*Riedmiller and Braun, 1993*], but the actual learning algorithm used here is the iRPROP- training algorithm [*Igel and Hüsken, 2000*] which is an variety of the standard RPROP training algorithm.

### FANN\_TRAIN\_QUICKPROP

A more advanced batch training algorithm which achieves good results for many problems. The quickprop training algorithm uses the `learning_rate` parameter along with other more advanced parameters, but it is only recommended to change these advanced parameters, for users with insight in how the quickprop training algorithm works.

The quickprop training algorithm is described in [*Fahlman, 1988*].

# Activation Functions

## Name

Activation Functions — Constants representing activation functions.

## Description

These constants represent the activation functions available within the fann library. The list will grow over time, but probably not shrink.

## Constants

### FANN\_THRESHOLD

*Execution only* - Threshold activation function.

This activation function gives output that is either 0 or 1.

### FANN\_THRESHOLD\_SYMMETRIC

*Execution only* - Threshold activation function.

This activation function gives output that is either -1 or 1.

### FANN\_LINEAR

*Can not be used in fixed point* - Linear activation function.

This activation function gives output that is unbounded.

### FANN\_SIGMOID

Sigmoid activation function. One of the most used activation functions.

This activation function gives output that is between 0 and 1.

### FANN\_SIGMOID\_STEPWISE

Stepwise linear approximation to sigmoid. Faster than sigmoid but a bit less precise.

This activation function gives output that is between 0 and 1.

#### FANN\_SIGMOID\_SYMMETRIC

Symmetric sigmoid activation function, AKA tanh. One of the most used activation functions.

This activation function gives output that is between -1 and 1.

#### FANN\_SIGMOID\_SYMMETRIC\_STEPWISE

Stepwise linear approximation to symmetric sigmoid. Faster than symmetric sigmoid but a bit less precise.

This activation function gives output that is between -1 and 1.

## Training Error Functions

### Name

Training Error Functions — Constants representing errors functions.

### Description

These constants represent the error functions used when calculating the error during training.

The training error function used is chosen by the `fann_set_train_error_function` function. The default training error function is `FANN_ERRORFUNC_TANH`.

### Constants

#### FANN\_ERRORFUNC\_LINEAR

The basic linear error function which simply calculates the error as the difference between the real output and the desired output.

#### FANN\_ERRORFUNC\_TANH

The tanh error function is an error function that makes large deviations stand out, by altering the error value used when training the network. The idea behind this is that it is worse to have 1 output that misses the target by 100%, than having 10 outputs that misses the target by 10%.

This is the default error function and it is usually better. It can however give poor results with high learning rates.

## Error Codes

### Name

Error Codes — Constants representing errors.

### Description

These constants represent the various errors possible in fann, as defined by `fann_errno.h`.

### Constants

FANN\_E\_NO\_ERROR

No error.

FANN\_E\_CANT\_OPEN\_CONFIG\_R

Unable to open configuration file for reading

FANN\_E\_CANT\_OPEN\_CONFIG\_W

Unable to open configuration file for writing

FANN\_E\_WRONG\_CONFIG\_VERSION

Wrong version of configuration file

FANN\_E\_CANT\_READ\_CONFIG

Error reading info from configuration file

FANN\_E\_CANT\_READ\_NEURON

Error reading neuron info from configuration file

FANN\_E\_CANT\_READ\_CONNECTIONS

Error reading connections from configuration file

FANN\_E\_WRONG\_NUM\_CONNECTIONS

Number of connections not equal to the number expected

FANN\_E\_CANT\_OPEN\_TD\_W

Unable to open train data file for writing

FANN\_E\_CANT\_OPEN\_TD\_R

Unable to open train data file for reading

FANN\_E\_CANT\_READ\_TD

Error reading training data from file

FANN\_E\_CANT\_ALLOCATE\_MEM

Unable to allocate memory

FANN\_E\_CANT\_TRAIN\_ACTIVATION

Unable to train with the selected activation function

FANN\_E\_CANT\_USE\_ACTIVATION

Unable to use the selected activation function

FANN\_E\_TRAIN\_DATA\_MISMATCH

Irreconcilable differences between two `fann_train_data` structures

## 5.9. Internal Functions

### 5.9.1. Creation And Destruction

#### **fann\_allocate\_structure**

##### **Name**

`fann_allocate_structure` — Allocate the core elements of a struct `fann`.

##### **Description**

```
struct fann * fann_allocate_structure(float learning_rate, unsigned int num_layers);
```

`fann_allocate_structure` is used internally to create a struct `fann`.

This function appears in FANN  $\geq$  1.0.0.

### 5.9.2. Input/Output

#### **fann\_save\_internal**

##### **Name**

`fann_save_internal` — Save an ANN to a file.

## Description

```
int fann_save_internal(struct fann * ann, const char * configuration_file, unsigned int save_as
```

`fann_save_internal_fd` is used internally to save an ANN to a file.

This function appears in FANN >= 1.0.0.

## fann\_save\_internal\_fd

### Name

`fann_save_internal_fd` — Save an ANN to a file descriptor.

### Description

```
int fann_save_internal_fd(struct fann * ann, FILE * conf, const char * configuration_file, unsi
```

`fann_save_internal_fd` is used internally to save an ANN to a location pointed to by `conf`. `configuration_file` is the name of the file, used only for debugging purposes.

This function appears in FANN >= 1.1.0.

## fann\_create\_from\_fd

### Name

`fann_create_from_fd` — Load an ANN from a file descriptor.

### Description

```
struct fann * fann_create_from_fd(FILE * conf, const char * configuration_file);
```

`fann_create_from_fd` will load an ANN from a file descriptor.

This function appears in FANN >= 1.1.0.

### 5.9.3. Training Data

## fann\_save\_train\_internal

### Name

fann\_save\_train\_internal — Save training data to a file.

### Description

```
void fann_save_train_internal(struct fann_train_data * data, char * filename, unsigned int save_as_fixed, int decimal_point);
```

Saves the data in *data* to *filename*. *save\_as\_fixed* is either TRUE or FALSE. *decimal\_point* tells FANN where the decimal point may be if using fixed point math.

This function appears in FANN >= 1.0.0.

## fann\_save\_train\_internal\_fd

### Name

fann\_save\_train\_internal\_fd — Save training data to a file descriptor.

### Description

```
void fann_save_train_internal_fd(struct fann_train_data * data, FILE * file, char * filename, unsigned int save_as_fixed, int decimal_point);
```

Saves the data in *data* to *file*. *save\_as\_fixed* is either TRUE or FALSE. *decimal\_point* tells FANN where the decimal point may be if using fixed point math.

*filename* is used for debugging output only.

This function appears in FANN >= 1.1.0.

## fann\_read\_train\_from\_fd

### Name

fann\_read\_train\_from\_fd — Read training data from a file descriptor.

## Description

```
struct fann_train_data * fann_read_train_from_file(FILE * file, char * filename);
```

`fann_read_train_from_file` will load training data from the file descriptor *file*.

*filename* is used for debugging output only.

This function appears in FANN >= 1.1.0.

## 5.9.4. Error Handling

### **fann\_error**

#### **Name**

`fann_error` — Throw an internal error.

#### **Description**

```
void fann_error(struct fann_error * errdat, unsigned int errno, ...);
```

This will set the network's error to correspond to *errno*. The variable arguments depend (both in type and quantity) on *errno*. Possible *errno* values are defined in `fann_errno.h`.

This function appears in FANN >= 1.1.0.

## 5.9.5. Options

### **fann\_update\_stepwise\_hidden**

#### **Name**

`fann_update_stepwise_hidden` — Adjust the stepwise function in the hidden layers.

## Description

```
void fann_update_stepwise_hidden(struct fann * ann);
```

Update the stepwise function in the hidden layers of *ann*.

This function appears in FANN >= 1.0.0.

## fann\_update\_stepwise\_output

### Name

`fann_update_stepwise_output` — Adjust the stepwise functions in the output layer.

### Description

```
void fann_update_stepwise_output(struct fann * ann);
```

Update the stepwise function in the output layer of *ann*.

This function appears in FANN >= 1.0.0.

## 5.10. Deprecated Functions

### 5.10.1. Mean Square Error

## fann\_get\_error

### Name

`fann_get_error` — Return the mean square error of an ANN.

### Description

```
float fann_get_error(struct fann * ann);
```

This function is deprecated and will be removed in a future version. Use `fann_get_MSE` instead.

This function appears in FANN  $\geq$  1.0.0, but is deprecated in FANN  $\geq$  1.1.0.

## fann\_reset\_error

### Name

`fann_reset_error` — Reset the mean square error of an ANN.

### Description

```
void fann_reset_error(struct fann * ann);
```

This function is deprecated and will be removed in a future version. Use `fann_reset_MSE` instead.

This function appears in FANN  $\geq$  1.0.0, but is deprecated in FANN  $\geq$  1.1.0.

## 5.10.2. Get and set activation function steepness.

## fann\_get\_activation\_hidden\_steepness

### Name

`fann_get_activation_hidden_steepness` — Retrieve the steepness of the activation function of the hidden layers.

### Description

```
fann_type fann_get_activation_hidden_steepness(struct fann * ann);
```

Return the steepness of the activation function of the hidden layers.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function is deprecated and will be removed in a future version. Use `fann_get_activation_steepness_hidden` instead.

This function appears in FANN  $\geq$  1.0.0, and is deprecated in FANN  $\geq$  1.2.0.

## fann\_set\_activation\_hidden\_steepness

### Name

`fann_set_activation_hidden_steepness` — Set the steepness of the activation function of the hidden layers.

### Description

```
void fann_set_activation_hidden_steepness(struct fann * ann, fann_type steepness);
```

Set the steepness of the activation function of the hidden layers of *ann* to *steepness*.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function is deprecated and will be removed in a future version. Use `fann_set_activation_steepness_hidden` instead.

This function appears in FANN >= 1.0.0. and is deprecated in FANN >= 1.2.0.

## fann\_get\_activation\_output\_steepness

### Name

`fann_get_activation_output_steepness` — Retrieve the steepness of the activation function of the output layer.

### Description

```
fann_type fann_get_activation_output_steepness(struct fann * ann);
```

Return the steepness of the activation function of the output layer.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function is deprecated and will be removed in a future version. Use `fann_get_activation_steepness_output` instead.

This function appears in FANN  $\geq$  1.0.0. and is deprecated in FANN  $\geq$  1.2.0.

## **fann\_set\_activation\_output\_steepness**

### **Name**

`fann_set_activation_output_steepness` — Set the steepness of the activation function of the hidden layers.

### **Description**

```
void fann_set_activation_output_steepness(struct fann * ann, fann_type steepness);
```

Set the steepness of the activation function of the hidden layers of *ann* to *steepness*.

The steepness defaults to 0.5 and a larger steepness will make the slope of the activation function more steep, while a smaller steepness will make the slope less steep. A large steepness is well suited for classification problems while a small steepness is well suited for function approximation.

This function is deprecated and will be removed in a future version. Use `fann_set_activation_steepness_output` instead.

This function appears in FANN  $\geq$  1.0.0. and is deprecated in FANN  $\geq$  1.2.0.

# Chapter 6. PHP Extension

These functions allow you to interact with the FANN library from PHP.

This extension requires the FANN (<http://fann.sf.net/>) library, version 1.1.0 or later.

This extension supports the same activation functions as the library, a list of which can be found in the *Activation Functions* section.

## 6.1. Installation

### 6.1.1. Using PEAR

The easiest way to install FANN-PHP is to use PEAR- if you have a fairly recent version of PHP installed, simply run **pear install fann**. Note that if there are no stable releases of FANN-PHP, you may have to specify the URI for the package, which can be obtained from <http://pecl.php.net/fann>.

If you cannot install FANN-PHP using PEAR, you can try following the (obsolete) instructions at <http://www.cs.utexas.edu/users/UTCS/online-docs/php/pear/faq.install-pecl.html>.

If you use one of these methods, you'll need to either `dl('fann.so')` or add it to your `php.ini`

If you use either of the above methods, you will probably need to be root.

### 6.1.2. Compiling into PHP

Please only use this method if using the methods outlined in *Using PEAR* have failed.

If you wish to compile FANN-PHP into PHP itself, you can. First, uncompress the package into the `ext` subdirectory of your copy of the PHP source code, and rename the directory to `ext/fann` (from `fann-x.x.x`).

Next, you must rebuild the configure script- to do so, run **./buildconf** from the PHP source directory.

From here on, the procedure is similar to when you built PHP originally- run **./configure** with your desired options, plus `--with-fann`.

Finally, run **make** and **make install**. Note that you will probably need to be root for **make install** to work.

This method may require flex and bison to work- more information can be obtained at <http://www.php.net/anoncvvs.php>

## 6.2. API Reference

### fann\_create

#### Name

`fann_create` — Creates an artificial neural network.

#### Description

```
mixed fann_create(mixed data, float connection_rate, float learning_rate);
```

`fann_create` will create an artificial neural network using the data given.

If the first parameter is an array, `fann_create` will use the data and structure of the array, as well as `connection_rate` and `learning_rate`.

If `fann_create` is called with a sole string argument, it will attempt to load an ANN created with `fann_save` from the file at `filename`.

`fann_create` will return the artificial neural network on success, or `FALSE` if it fails.

#### Example 6-1. fann\_create from scratch

```
<?php
$ann = fann_create(
    /* Layers. In this case, three layers-
     * two input neurons, 4 neurons on a
     * hidden layer, and one output neuron. */
    array(2, 4, 1),
    1.0,
    0.7);
?>
```

#### Example 6-2. fann\_create loading from a file

```
<?php
$ann = fann_create("http://www.example.com/ann.net");
?>
```

See also `fann_save`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_train

### Name

`fann_train` — Train an artificial neural network.

### Description

```
bool fann_train(resource ann, mixed data, int max_iterations, double desired_error, int iterations)
```

`fann_train` will train `ann` on the data supplied, returning TRUE on success or FALSE on failure.

`Resources` is an artificial neural network returned by `fann_create`.

`data` must be either an array of training data, or the URI of a properly formatted training file.

`fann_train` will continue training until `desired_error` is reached, or `max_iterations` is exceeded.

If `iterations_between_reports` is set, `fann_create` will output a short progress report every `iterations_between_reports`. Default is 0 (meaning no reports).

#### Example 6-1. fann\_create from training data

```
<?php
$ann = fann_create(array(2, 4, 1), 1.0, 0.7);
if ( fann_train($ann,
    array(
        array(
            array(0,0), /* Input(s) */
            array(0) /* Output(s) */
        ),
        array(
            array(0,1), /* Input(s) */
            array(1) /* Output(s) */
        ),
        array(
            array(1,0), /* Input(s) */
            array(1) /* Output(s) */
        ),
        array(array(1,1), /* Input(s) */
            array(0) /* Output(s) */
        )
    ),
    100000,
```

```

        0.00001,
        1000) == FALSE) {
    exit('Could not train $ann.');
```

```

}
?>
```

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_save

### Name

fann\_save — Save an artificial neural network to a file.

### Description

```
bool fann_save(resource ann, string filename);
```

fann\_save will save *ann* to *filename*, returning TRUE on success or FALSE on failure.

See also fann\_create.

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_run

### Name

fann\_run — Run an artificial neural network.

### Description

```
mixed fann_run(resource ann, array input);
```

fann\_run will run *input* through *ann*, returning an an output array on success or FALSE on failure.

**Example 6-1. fann\_runExample**

```

<?php
if ( ($ann = fann_create("http://www.example.com/ann.net")) == FALSE )
    exit("Could not create ANN.");
if ( fann_train($ann, "http://www.example.com/train.data", 100000, 0.00001) == FALSE )
    exit("Could not train ANN.");

if ( ($output = fann_run($ann, array(0, 1))) == FALSE )
    exit("Could not run ANN.");
else
    print_r($output);
?>

```

This function appears in FANN-PHP  $\geq$  0.1.0.

**fann\_randomize\_weights****Name**

`fann_randomize_weights` — Randomize the weights of the neurons in the network.

**Description**

```
void fann_randomize_weights(resource ann, float minimum, float maximum);
```

`fann_randomize_weights` will randomize the weights of all neurons in *ann*, effectively resetting the network.

See also: *Adjusting Parameters*, `fann_init_weights`

This function appears in FANN-PHP  $\geq$  0.1.0.

**fann\_init\_weights****Name**

`fann_init_weights` — Initialize the weight of each connection.

## Description

```
void fann_init_weights(resource ann, mixed training_data);
```

This function behaves similarly to `fann_randomize_weights`. It will use the algorithm developed by Derrick Nguyen and Bernard Widrow [Nguyen and Widrow, 1990] to set the weights in such a way as to speed up training.

The algorithm requires access to the range of the input data (ie, largest and smallest input), and therefore accepts a second argument, *data*, which is the training data that will be used to train the network.

See also: *Adjusting Parameters*, `fann_randomize_weights`

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_get\_MSE

### Name

`fann_get_MSE` — Get the mean squared error.

### Description

```
float fann_get_MSE(resource ann);
```

`fann_get_MSE` will return the mean squared error (MSE) of *ann*, or 0 if it is unavailable.

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_get\_num\_input

### Name

`fann_get_num_input` — Get the number of input neurons.

### Description

```
int fann_get_num_input(resource ann);
```

`fann_get_num_input` will return the number of input neurons in *ann*.

See also `fann_get_num_output`, `fann_get_total_neurons`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_get\_num\_output**

### **Name**

`fann_get_num_output` — Get the number of output neurons.

### **Description**

```
int fann_get_num_output(resource ann);
```

`fann_get_num_output` will return the number of output neurons in *ann*.

See also `fann_get_num_input`, `fann_get_total_neurons`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_get\_total\_neurons**

### **Name**

`fann_get_total_neurons` — Get the total number of neurons.

### **Description**

```
int fann_get_total_neurons(resource ann);
```

`fann_get_total_neurons` will return the total number of neurons in *ann*.

See also `fann_get_num_input`, `fann_get_num_output`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_get\_total\_connections**

### **Name**

`fann_get_total_connections` — Get the total number of connections.

### **Description**

```
int fann_get_total_connections(resource ann);
```

`fann_get_total_connections` will return the total number of connections in *ann*.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_get\_learning\_rate**

### **Name**

`fann_get_learning_rate` — Get the learning rate.

### **Description**

```
float fann_get_learning_rate(resource ann);
```

`fann_get_learning_rate` will return the learning rate of *ann*.

See also `fann_set_learning_rate`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_get\_activation\_function\_hidden**

### **Name**

`fann_get_activation_function_hidden` — Get the activation function of the hidden neurons.

## Description

```
int fann_get_activation_function_hidden(resource ann);
```

`fann_get_activation_function_hidden` will return the activation function for the hidden neurons in *ann*.

See also `fann_set_activation_function_hidden`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_get\_activation\_function\_output

### Name

`fann_get_activation_function_output` — Get the activation function of the output neurons.

### Description

```
int fann_get_activation_function_output(resource ann);
```

`fann_get_activation_function_output` will return the activation function for the output neurons in *ann*.

See also `fann_set_activation_function_output`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## fann\_get\_activation\_steepness\_hidden

### Name

`fann_get_activation_steepness_hidden` — Get the steepness of the activation function for the hidden neurons.

### Description

```
float fann_get_activation_steepness_hidden(resource ann);
```

`fann_get_activation_steepness_hidden` will return the steepness of the activation function for the hidden neurons in *ann*.

See also `fann_set_activation_steepness_hidden`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_get\_activation\_steepness\_output**

### **Name**

`fann_get_activation_steepness_output` — Get the steepness of the activation function for the output neurons.

### **Description**

```
float fann_get_activation_steepness_output(resource ann);
```

`fann_get_activation_steepness_output` will return the steepness of the activation function for the output neurons in *ann*.

See also `fann_set_activation_steepness_output`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_set\_learning\_rate**

### **Name**

`fann_set_learning_rate` — Set the learning rate.

### **Description**

```
float fann_set_learning_rate(resource ann);
```

`fann_set_learning_rate` will return the learning rate of *ann*.

See also `fann_set_learning_rate`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_set\_activation\_function\_hidden**

### **Name**

`fann_set_activation_function_hidden` — Set the activation function for the hidden neurons.

### **Description**

```
void fann_set_activation_function_hidden(resource ann, int activation_function);
```

`fann_set_activation_function_hidden` sets the activation function for the hidden neurons to *activation\_function*, which must be one of the supported activation functions.

See also `fann_get_activation_function_hidden`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_set\_activation\_function\_output**

### **Name**

`fann_set_activation_function_output` — Set the activation function for the output neurons.

### **Description**

```
void fann_set_activation_function_output(resource ann, int activation_function);
```

`fann_set_activation_function_output` sets the activation function for the output neurons to *activation\_function*, which must be one of the supported activation functions.

See also `fann_get_activation_function_output`.

This function appears in FANN-PHP  $\geq$  0.1.0.

## **fann\_set\_activation\_steepness\_hidden**

### **Name**

`fann_set_activation_steepness_hidden` — Set the steepness of the activation function for the hidden neurons.

### **Description**

```
void fann_set_activation_steepness_hidden(resource ann, float steepness);
```

`fann_set_activation_steepness_hidden` sets the steepness of the activation function hidden neurons to *steepness*.

See also `fann_get_activation_steepness_hidden`.

This function appears in FANN-PHP  $\geq 0.1.0$ .

## **fann\_set\_activation\_steepness\_output**

### **Name**

`fann_set_activation_steepness_output` — Set the steepness of the activation function for the output neurons.

### **Description**

```
void fann_set_activation_steepness_output(resource ann, float steepness);
```

`fann_set_activation_steepness_output` sets the steepness of the activation function output neurons to *steepness*.

See also `fann_get_activation_steepness_output`.

This function appears in FANN-PHP  $\geq 0.1.0$ .

# Chapter 7. Python Bindings

These functions allow you to interact with the FANN library from Python.

This extension requires the FANN (<http://fann.sf.net/>) library, version 1.1.0 or later.

This python binding is provided by Vincenzo Di Massa ([hawk.it@tiscalinet.it](mailto:hawk.it@tiscalinet.it)) and updated by Gil Megidish ([gil@megidish.net](mailto:gil@megidish.net))

## 7.1. Python Install

Make sure to make and install the fann library first. Make sure that you have swig and python development files installed. Perhaps change the include directory of python. Then run 'make' to compile in the python directory.

Copy the generated `_fann.so` and `fann.py` files to python modules or into working directory.

After the install, just import `fann` and all the C functions will be available to your python code.

# Chapter 8. Delphi Bindings

These functions allow you to interact with the FANN library from Delphi.

This extension requires the FANN (<http://fann.sf.net/>) library, version 1.2.0 or later.

This extension can be downloaded from the FANN (<http://fann.sf.net/>) library download section.

This Delphi binding is provided by Maurício Pereira Maia ([mauricio@uaisol.com.br](mailto:mauricio@uaisol.com.br))

## 8.1. Delphi Install

Make sure to make and install the fann library first. Put the file `fannfloat.dll` in your `PATH`. (If you want to use the fixed version you should define `FIXEDFANN` on `fann.pas`). Include `fann.pas` in your project and in your unit uses clause, and have fun! See the `XorConsole` sample for more details.

## 8.2. TFannNetwork

`TFannNetwork` is a Delphi component that encapsulates the Fann Library. You do not have to install `TFannNetwork` to use Fann on Delphi, but it will make the library more Delphi friendly. Currently it has only a small subset of all the library functions, but I hope that will change in the near future.

To install `TFannNetwork` you should follow all the previous steps and copy the `FannNetwork.pas` and `Fann.dcr` to your Delphi Library `PATH`. Choose Component/Install Component. In the Unit file name field, click on Browse and point to the `fannnetwork.pas` file. By default Delphi will install in the Borland User Components package, it might be changed using Package file name field or Into new package page. Click on Ok. A confirmation dialog will be shown asking if you want to build the package. Click on Yes. You have just installed `TFannNetwork`, now close the package window (Don't forget to put Yes when it ask if you want to save the package). See the `FannNetwork.pas` file or the `Xor Sample`.

## 8.3. Known Problems

If you are getting in trouble to use your own compiled FANN DLL with Delphi that might be because of the C++ naming mangle that changes between C++ compilers. You will need to make a `TDUMP` on your dll and changes all the name directives on `fann.pas` to the correct function names on your dll.

# Bibliography

- [Anderson, 1995] J.A. Anderson, 1995, *An Introduction to Neural Networks*, The MIT Press.
- [Anguita, 1993] D. Anguita, *Matrix back propagation v1.1*.
- [Bentley, 1982] J.L. Bentley, 1982, *Writing Efficient Programs*, Prentice-Hall.
- [Blake and Merz, 1998] C. Blake, C. Merz, 1998, *UCI repository of machine learning databases*, <http://www.ics.uci.edu/mllearn/MLRepository.html> (<http://www.ics.uci.edu/mllearn/MLRepository.html>) .
- [Darrington, 2003] J. Darrington, 2003, *Libann*, <http://www.nongnu.org/libann/index.html> .
- [Fahlman, 1988] S.E. Fahlman, 1988, *Faster-learning variations on back-propagation*, <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/sef/www/publications/qp-tr.ps> , *An empirical study*.
- [LGPL] Free Software Foundation, 1999, *GNU Lesser General Public License*, Free Software Foundation, <http://www.fsf.org/copyleft/lesser.html> .
- [Hassoun, 1995] M.H. Hassoun, 1995, *Fundamentals of Artificial Neural Networks*, The MIT Press.
- [Heller, 2002] J. Heller, 2002, *Jet's Neural Library*, [http://www.voltar.org/jneural/jneural\\_doc/](http://www.voltar.org/jneural/jneural_doc/) .
- [Hertz et al., 1991] J. Hertz, A. Krogh, R.G. Palmer, 1991, *Introduction to The Theory of Neural Computing*, Addison-Wesley Publishing Company.
- [IDS, 2000] ID Software, 2000, *Quake III Arena*, <http://www.idsoftware.com/games/quake/quake3-arena/> (<http://www.idsoftware.com/games/quake/quake3-arena/>) .
- [Igel and Hüsken, 2000] Christian Igel, Michael Hüsken, 2000, *Improving the Rprop Learning Algorithm*, <http://citeseer.ist.psu.edu/igel00improving.html> (<http://citeseer.ist.psu.edu/igel00improving.html>) .
- [Kaelbling, 1996] L.P. Kaelbling, M.L. Littman, A.P. Moore, 1996, *Reinforcement Learning: A New Survey*, Journal of Artificial Intelligence Research, 4, 237-285.
- [LeCun et al., 1990] Y. LeCun, J. Denker, S. Solla, R.E. Howard, L.D. Jackel, 1990, *Advances in Neural Information Processing Systems II*.
- [Nguyen and Widrow, 1990] *Reinforcement Learning*, Derrick Nguyen, Bernard Widrow, 1990, Proc. IJCNN, 3, 21-26, <http://www.cs.montana.edu/~clemens/nguyen-widrow.pdf> .
- [Nissen et al., 2003] S. Nissen, J. Damkjær, J. Hansson, S. Larsen, S. Jensen, 2003, *Real-time image processing of an ipaq based robot with fuzzy logic (fuzzy)*, <http://www.hamster.dk/~purple/robot/fuzzy/weblog/> (<http://www.hamster.dk/~purple/robot/fuzzy/weblog/>) .
- [Nissen et al., 2002] S. Nissen, S. Larsen, S. Jensen, 2003, *Real-time image processing of an iPAQ based robot (iBOT)*, <http://www.hamster.dk/~purple/robot/iBOT/report.pdf> (<http://www.hamster.dk/~purple/robot/iBOT/report.pdf>) .
- [OSDN, 2003] 2003, *SourceForge.net*, <http://sourceforge.net/> .
- [Pendleton, 1993] R.C. Pendleton, 1993, *Doing it Fast*, <http://www.gameprogrammer.com/4-fixed.html> .
- [Prechelt, 1994] L. Prechelt, 1994, *Proben1: A set of neural network benchmark problems and benchmarking rules*.

- [Riedmiller and Braun, 1993] Riedmiller, Braun, 1993, *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*, 586-591, <http://citeseer.nj.nec.com/riedmiller93direct.html> (<http://citeseer.nj.nec.com/riedmiller93direct.html>) .
- [Sarle, 2002] W.S. Sarle, , *Neural Network FAQ*, [ftp://ftp.sas.com/pub/neural/FAQ2.html#A\\_binary](ftp://ftp.sas.com/pub/neural/FAQ2.html#A_binary) .
- [Pemstein, 2002] Dan Pemstein, 2002, *ANN++*, <http://savannah.nongnu.org/projects/annpp/> .
- [Tettamanzi and Tomassini, 2001] A. Tettamanzi, M. Tomassini, , *Soft Computing*, Springer-Verlag.
- [Thimm and Fiesler, High-Order and Multilayer Perceptron Initialization, 1997] Georg Thimm, Emile Fiesler, March 1997, *High-Order and Multilayer Perceptron Initialization*, IEEE Transactions on Neural Networks, 8, 2, 249-259, <http://citeseer.ist.psu.edu/thimm96high.html> .
- [Thimm and Fiesler, 1997] G Thimm, E Fiesler, 1997, *Optimal Setting of Weights, Learning Rate, and Gain*, <http://citeseer.ist.psu.edu/thimm97optimal.html> .
- [van Rossum, 2003] P. van Rossum, 2003, *Lightweight neural network*, <http://lwneuralnet.sourceforge.net/> .
- [van Waveren, 2001] J.P. van Waveren, 2001, *The quake III arena bot*, <http://www.kbs.twi.tudelft.nl/Publications/MSc/2001-VanWaveren-MSc.html> (<http://www.kbs.twi.tudelft.nl/Publications/MSc/2001-VanWaveren-MSc.html>) .
- [Zell, 2003] A. Zell, 2003, *Stuttgart neural network simulator*, <http://www-ra.informatik.uni-tuebingen.de/SNNS/> .