Genann

Genann is a minimal, well-tested library for training and using feedforward artificial neural networks (ANN) in C. Its primary focus is on being simple, fast, reliable, and hackable. It achieves this by providing only the necessary functions and little extra.

Features

- **ANSI C with no dependencies**.
- Contained in a single source code and header file.
- Simple.
- Fast and thread-safe.
- Easily extendible.
- Implements backpropagation training.
- *Compatible with alternative training methods* (classic optimization, genetic algorithms, etc)
- Includes examples and test suite.
- Released under the zlib license - free for nearly any use.

Building

Genann is self-contained in the !Genann.Source directory. I already added a lot of Makefiles to make sure you can easly re-build it for your specific C Compiler on RISC OS and your specific ARM Architecture variant.

So open !Genann.Source with your Desktop Filer and then edit the BuildDDE or BuildGCC TaskObey files to "enable/disable" the builds you need/don't need.

If you let it as default then it will try to build every Genann variant supported by your compiler of choice.

Example Code

Four example programs are included with the source code.

- `!GenannEx1` - Trains an ANN on the XOR function using backpropagation.

- `!GenannEx2` - Trains an ANN on the XOR function using random search.
- `!GenannEx3` - Loads and runs an ANN from a file.
- `!GenannEx4` - Trains an ANN on the IRIS data-set using backpropagation.

Quick Example

We create an ANN taking 2 inputs, having 1 layer of 3 hidden neurons, and providing 2 outputs. It has the following structure:

We then train it on a set of labeled data using backpropagation and ask it to predict on a test data point:

```
#include "GenannLib:genann.h"

/* Not shown, loading your training and test data. */
double **training_data_input, **training_data_output,
**test_data_input;

/* New network with 2 inputs,
 * 1 hidden layer of 3 neurons each,
 * and 2 outputs. */
genann *ann = genann_init(2, 1, 3, 2);

/* Learn on the training set. */
for (i = 0; i < 300; ++i) {
    for (j = 0; j < 100; ++j)
        genann_train(ann, training_data_input[j],
training_data_output[j], 0.1);
}

/* Run the network and see what it predicts. */
double const *prediction = genann_run(ann, test_data_input[0]);
printf("Output for the first test data point is: %f, %f\n",
prediction[0], prediction[1]);

genann_free(ann);
```

This example is to show API usage, it is not showing good machine learning techniques. In a real application you would likely want to learn on the test data in a random order. You would also want to monitor the learning to prevent over-fitting.

Usage

Creating and Freeing ANNs

```
genann *genann_init(int inputs, int hidden_layers, int hidden, int
outputs);
genann *genann_copy(genann const *ann);
void genann_free(genann *ann);
```

Creating a new ANN is done with the `genann_init()` function. Its arguments are the number of inputs, the number of hidden layers, the number of neurons in each hidden layer, and the number of outputs. It returns a `genann` struct pointer.

Calling `genann_copy()` will create a deep-copy of an existing `genann` struct.

Call `genann_free()` when you're finished with an ANN returned by `genann_init()`.

Training ANNs

```
void genann_train(genann const *ann, double const *inputs,
        double const *desired_outputs, double learning_rate);
```

`genann_train()` will preform one update using standard backpropogation. It should be called by passing in an array of inputs, an array of expected outputs, and a learning rate. See *GenannExamples.!GenannEx1.Source.c.example1* for an example of learning with backpropogation.

A primary design goal of Genann was to store all the network weights in one contigious block of memory. This makes it easy and efficient to train the network weights using direct-search numeric optimizion algorthims, such as Hill Climbing, the Genetic Algorithm, Simulated Annealing, etc. These methods can be used by searching on the ANN's weights directly. Every `genann` struct contains the members `int total_weights;` and `double *weight;`. `*weight` points to an array of `total_weights` size which contains all weights used by the ANN. See *example2.c* for an example of training using random hill climbing search.

Saving and Loading ANNs

```
genann *genann_read(FILE *in);
void genann_write(genann const *ann, FILE *out);
```

Genann provides the `genann_read()` and `genann_write()` functions for

loading or saving an ANN in a text-based format.

Evaluating

```
double const *genann_run(genann const *ann, double const *inputs);
```

Call `genann_run()` on a trained ANN to run a feed-forward pass on a given set of inputs. `genann_run()` will provide a pointer to the array of predicted outputs (of `ann->outputs` length).

Compiling your code

For DDE:

```
    cc blah blah -I GenannLib:
    link blah blah GenannLib:o.genanDDE[<X><Y>]
```

Where X Y are optionals and should be specified only when you want to use an architecture optimized version of the library, while for general compilations just using GenannDDE is more than enough.

X can be either 26 or 32

Y can be ARMv2 or ARMv4 or ARMv5 or ARMv6 or ARMv7

So for example:

```
    link blah blah GenannLib:o.genannDDE
```

```
    link blah blah GenannLib:o.genannDDE26ARMv2
```

```
    link blah blah GenannLib:o.genannDDE32ARMv7
```

For GCC:

```
    CFLAGS=-IGenannLib:
    LDFLAGS= blah blah -l:GenannLib:o.genannGCC
```

And then use the variables above as:

```
    gcc $(CFLAGS) blah blah $(LDFLAGS)
```

Hints

- All functions start with `genann_` .
- The code is simple. Dig in and change things.

Extra Resources

The comp.ai.neural-nets FAQ is an excellent resource for an introduction to artificial neural networks.