Aimotive C++ test

> A simple neral network implementation by Máté Kerekes.

1. Notes on embedded implementation

This project contains a simple implementation that favors convenience over performance or generality. There are numerous directions for improvements and optimizations most of which depend on the needs and capabilities of the target platform.

1.1 Network bias

The example does not explicitly mention the bias term, (and I forgot to ask), so I've included it in the implementation for good measure. This might be unnecessary generality, if the implemented network is confirmed as not ever requiring a bias term, the related code would obviously better be removed.

1.2 Network representation

I've purposefully opted to go for a simple connection-weight-matrix based approach (see details in `neuralnet.h`). Since running a network requires no interaction with individual neurons there is little point in representing them as individual classes. Layers could deserve their own implementation if a wider assortment of activation functions, and more advanced layer types are required.

Nested `std::vector`s were used because of their ease of use. They can be input as neat bracket initializer lists without having to worry about size template arguments. Since they are not resized, their overhead compared to `std::array` should be constant at worst. Alternatively a custom `Matrix*`-esque structure using static arrays could be implemented.

With a couple of orders of magnitude larger networks, the current implementation fails spectacularly on two fronts: depending on the environment, the time it takes to load the net into memory the inputs could already have became outdated. It's also possible that the network will not even fit into the memory of the embedded target.

1.3 Network streaming

To remedy these issues, the network class could be implemented in a streaming manner, where only a chunk of the network is present in memory at any given time. Since the dependencies within the network are one directional (input -> ... -> output), the streaming implementation is completely straight forward. This could also allow pipe lining i.e. loading an upcoming chunk into memory while the input is being propagated trough a preceding one.

1.4 Network throughput

In extreme cases, given modest network sizes and a relative abundance of hardware resources (e.g. on an FPGA?), a different avenue of improvement could be pipe lining across layers. This could allow a constant high throughput, with latency mainly dependent on the number of hidden layers.

1.5 Parallel computations

Partially included in the previous two points. Apart from pipe lining, straight up parallelism is also well implementable thanks to the linear dependency of the network. Propagating a signal boils down to a series of dot products, which are independently calculable within layers. The nature of available parallelism mechanisms are going to be highly platform dependent.

1.6 Standard library

The std library was extensively used (excessively even at places) to be able to write code quickly. The little experience I have, tells me that this convenience is anything but present on embedded systems. Naturally if the used features/implementations happen to be nonexistent (or not performant enough) on the target architecture, these would have to be replaced with custom implementations.

---

2. Structure

Configuring the network

To modify the neural net, edit the `network` definition in `config.h`. For details about the representation and structure of the network constant, see the definition of `Network` in `neuralnet.h`.

> If your project does not have `config.h`, make a manual copy of `example.config.h` and name it accordingly, or, run `./script/build` to automatically generate it. `config.h` is not tracked in git to allow you changing the network without causing merge conflicts.

---

3. Scripts

Build

```
# Build in debug mode and run tests
> ./script/build debug
# Run the executable
> ./build/debug/neuralnet 0.2 -0.4 0.6
```

```
# Build in release mode and run tests
> ./script/build release
# Run the executable
> ./build/release/neuralnet 0.2 -0.4 0.6
```

```
# Clean build directories
> ./script/build clean
```

Distribution

```
# Create dist folder with project archive and release
> ./script/package
```