Lisp in Summer Projects Submission

Submission Date	2013-10-24 16:14:52
Full Name	Zach Kost-Smith
Country	USA
Project Name	CLANN
Type of software	library
General category	library
LISP dialect	Commmon Lisp
GitHub URL	https://github.com/smithzvk/clann
Did you start this project?	Yes, all the code is written by me
Project Description	I want to describe my project in this form.
Purpose	To provide a flexible yet efficient framework for developing and using artificial neural networks in Common Lisp. The primary intent is machine learning.
Function	This project allows you to build simple feed forward networks (with options of a few different activation functions). It also provides an implementation of the back-propagation learning algorithm for these simple feed forward networks. The end goal was to provide efficient BLAS and CUDA implementations, but this didn't come to fruition.
Motivation	Machine learning is an extremely interesting subject, but try as I might, I cannot find a good NN implementation for CL. The closest I have found is CL-FANN, which is quite limited (though not as limited as this library in its current state).
Audience	I wrote this for people interested in machine learning and artificial intelligence while also wanting to study these topics from inside a REPL.

Methodology

The work here is based on implementations of neural networks from Geoffry Hinton's class on Neural Networks and Andrew Ng's class on Machine Learning. In principle, this is only a library that aids in multiplying matrices and vectors, then mapping the vectors with an activation function. From this point of view, NN computation (and training) is merely a layer of abstraction over a linear algebra library such as BLAS, cuBLAS, or AAPML.

The programming method was based on first building a reference implementation (that runs entirely within Lisp) that can be used to validate the lower level code that will likely be based in C or some kind of GPU DSL. In it's current version, only a partial implementation of the reference implementation is complete.

Each neural network is represented as a list of layers. Each layer is a list containing a transition matrix (which maps the outputs of the previous layer (plus a bias input) to the inputs of the current layer), a bias vector, and a vector of activation functions.

All of the computation for this project is basically matrix-matrix multiplication and mapping functions over the resulting matrix. The matrix manipulation is implemented using my Index-mapped-arrays library, which provides a uniform interface over indexable data structures in Lisp.

Some effort was made to write CLANN in a literate programming style. The method I use is one of my own devising that I call Literate-Lisp. This is a simple system where you place any literate documentation in your programs comments (annotated by a leading "@" to tell the parser to switch to literate documentation mode). See https://github.com/smithzvk/literate-lisp This is not a necessary requirement to compile, load, or run the program.

Conclusion

In the end the reference implementation seems to work somewhat for some very simple tasks, such as implementing a NOT or OR gate as a neural network. The reference implementation is very slow, and is likely not correct as it has not been fully verified.

Finally, when I stopped work on this in mid July (when I determined that writing my thesis was drastically more important), I was resigned to not submitting anything. Your friendly email has persuaded me otherwise. I would very much like to be a part of Lisp community, even if my accomplishments are of poor quality right now. I hope you do this again next year.

Build Instructions

If you have quicklisp installed in ~/quicklisp, it will install Iterate. You also need my Index-mapped-arrays library.

Type "make" from the cloned CLANN directory to have it clone index-mapped-arrays into your local-projects folder.

Test Instructions

No automated test suite.

Execution Instructions

You may work through the examples in examples.lisp. Hopefully this will provide enough info of how to use this.

Make a network: (make-network '(n-inputs hidden-layer1 ...

	hidden-layer-n n-outputs)) Predict: (predict input(s) *net*) Train: (gradient-descent inputs *net* expected-outputs)
Describe any bugs or caveats	The biggest bug is that this was not left in a build-able state at the time of the contest deadline. To see what work was done at the time of the contest end see the master branch. To see a version that works (a few minor bug-fixes to get it to working status, and adding Makefile and examples.lisp) see the LISP-submission branch. I understand that this is not within the rules, but I figured it is better to give people code that actually runs than just a mess.
Official	I have read rules and have abided by them. I am 18 years of age or older. I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.