LibSDN / Conceptual Overview

Outline

The LibSDN library has a few important concepts:

- Blocks
- Networks
- Scripts (and Callbacks)
- Script Runtime Manager
- Registries
- Parameters: Configurable, Persistent, Transient

Parameters

As a coding convention, we define 3 types of parameters:

1. Configurable

- These are parameters which need to be chosen or tuned. For example, learning rates, target activations, weight decay etc.

1. Persistent

- These are often learned parameters that need to persist when saving the model. For example, weight and bias matrices

1. Transient

- These are parameters which exist for short moments and need not be saved. For example, gradients or temporary caches.

When serializing, we would serialize the configurable and persistent parameters and call initTransientParameters to set those up.

Blocks

Blocks are independent pieces of code meant to help make code modular and reusable. Any piece of code that is used multiple times in your code base should be made into a block. Similarly, if you would like to share code, try to chunk up the code into blocks that are easy to put together.

Examples of blocks include: exponentially weight mean tracker, weights visualization, normalization of input, etc.

Composable Blocks are a special case of blocks. These are blocks which further specify 3 operations: forwardProp, backProp, gradientUpdate.

Composable blocks often form the layers of a feed-forward neural network. Forward prop computes the output of the inputs. Back prop computes the gradients, while gradient update modifies the block's internal state (weights). The library provides affine, tanh and a sparse version of the affine-tanh block.

Networks

We abstractly define two types of networks:

- AutoEncoder Networks
- Supervised Networks

Both networks are often composed of many composable blocks. The main reason for separating these two networks are that we often end up using an AutoEncoder Network in many ways: stacking them up, or using them as part of a supervised network.

The library provides a few pre-built networks you can use, including Composable AutoEncoder/Supervised Networks, Stacked AutoEncoder Networks, Standard AutoEncoder Network.

Hence, one could for example, train a few composable autoencoder networks, then stack them up using the stacked autoencoder network and finally push this into a supervised network. To put all these steps together, we introduce scripts in the next section.

Scripts

Scripts is a proxy to the "main()" function. A script contains the instructions on how to setup a network, a dataset and train them. When working with the library, we expect most users to be creating their own custom scripts.

Scripts read configurations from XML files. This enables us to keep the code separate from the parameters. Often, one can move even the network specification to the XML configuration. The composable autoencoder for example, is normally created by a configuration specified using XML.

Script Runtime Manager and Callbacks

To actually run scripts, we provide a script runtime manager. The runtime manager manages the running script and callbacks. Only one script can run at anytime. The runtime manager also provide an option to use callbacks while running the scripts.

These callbacks often perform operations on the state of the script such as - reporting error rates, saving, visualization. For example, we might want to report error rates every 1000 iterations. A callback would be suited for use there.

Registries

In order to do dynamically loading of classes, we have various registries for various classes. Most importantly, we have a registry for scripts which you should register your script with to be able to load it using the main program.

Configuration Manager/Listener

To aid developers in making easy to customize code, ConfigManager and ConfigListener classes have been provided. The ConfigManager is a static class which is always loaded. The main function of an app needs to register the config file (normally passed in via the cmd line) with the ConfigManager.

Classes which wish to read XML information from the config file registered can simply spawn a ConfigListener/Parser which will request the data from the ConfigManager.

See any script to understand how this works.

Program Flow

When using the script system, the program flow goes like this:

- A static Configuration Manager is created and the main function registers the config file with it.
- The script runtime manager is created and loads the XML config file
- It looks for a specified script and loads the script using the registry.
- The script loads the XML config file as well (through the ConfigManager), requesting only for its portion of the configuration.
  - The script is responsible for loading any datasets, networks it wishes to use
- The script runtime manager runs the scripts by calling Script::executeOneIteration.
- Every Iteration, we check if any call backs need to be called. If so, we call the callback, giving it all the information it needs
- The program terminates when the desired # iterations has been reached in the script runtime manager

Others

Objective Terms

- Unary Objective Terms
  - L2 Penalty
  - Logistic Cost
  - Quadratic 0-1 Cost
- Cost Functions (Binary Objective Terms)
  - Sum Of Squares Cost Function
  - L1 Cost Function
  - Smoothed L1 Cost Function

Page last modified on January 12, 2010, at 09:20 AM by jngiam