BreakingCaptcha Reference Manual

0.1

Generated by Doxygen 1.5.3

Sat Nov 3 01:33:02 2007

Contents

Chapter 1

BreakingCaptcha Class Index

1.1 BreakingCaptcha Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

BreakingCaptcha File Index

2.1 BreakingCaptcha File List

Here is a list of all documented files with brief descriptions:

Chapter 3

BreakingCaptcha Class Documentation

3.1 GenericLayer Class Reference

```
#include <GenericLayer.h>
```

Public Member Functions

- GenericLayer (int numNeurons, GenericLayer ∗parent, GenericLayer ∗child)
- virtual ∼GenericLayer ()
- void initWeights ()
- void initNeurons ()
- void init ()

Public Attributes

- GenericLayer ∗ parentLayer
- GenericLayer ∗ childLayer
- Neuron ∗ neurons
- int numNeurons
- double ∗∗ weights

3.1.1 Detailed Description

Provides a generic implementation of a neural network layer. In general, the network is defined as loosely coupled neurons in a feed forward three layered design. The neurons are all each connected to adjacent layer neurons, and those connections have weights associated with them. In this implementation, those neurons with a child layer will be associated with the weights between them and their children. Also included are bias factors, which are stored in the neurons themselves since there is only one bias per neuron. The neurons will be initialized with values of 0 except for the bias value which will be 1. The error factor is also included on the neuron itself.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 GenericLayer::GenericLayer (int *numNeurons*, GenericLayer ∗ *parent*, GenericLayer ∗ *child*)

Defines the constructor for the generic neural layer.

Parameters:

numNeurons The number of neurons to create in this layer.

parent A pointer to the parent layer of this current layer. NULL if this layer is the inputer layer.

child A pointer to the child layer of this current layer. NULL if this layer is the output layer.

```
GenericLayer input, hidden, output;
input =  new GenericLayer(50, NULL, &hidden);
hidden = new GenericLayer(50, &input, &output);
output = new GenericLayer(50, &hidden, NULL);

input.init(); hidden.init(); output.init();
```

Exceptions:

integer Throws an integer exception when the bounds of the GenericLayer::MAX_NEURONS or GenericLayer::MIN_NEURONS number of neurons in this layer has been broken.

Postcondition:

All the default values have been assigned and all arrays have been allocated.

3.1.2.2 GenericLayer::∼GenericLayer () `[virtual]`

Deallocates the memory from the neuron and weight arrays.

3.1.3 Member Function Documentation

3.1.3.1 void GenericLayer::initWeights ()

Initialize the weights array, but only if we are in a layer with a child because they are responsible for keeping track of the weights. The number of total entries should number GenericLayer.numNeurons∗GenericLayer.childLayer.numNeurons. The weights are all initialized to zero before training.

Postcondition:

The array containing the weights for each neuron is allocated and has been filled with zeros. It is ready to be trained.

3.1.3.2 void GenericLayer::initNeurons ()

Precondition:

The neuron array is undefined.

Postcondition:

The neuron array has been allocated and intelligent defaults have been set. There is now GenericLayer::numNeurons neurons in this layer. It is ready to be trained.

3.1.3.3 void GenericLayer::init ()

Since the layers are created in a linked manner, such that the input layer is linked to the hidden layer, etc., we need another method, outside of the constructor so we can create the data structures when all the information is made available. Doing so in the constructor would fail, for example, since creation of the input layer's data structures implies knowledge of the hidden layer's data, which may or may not have been previously defined. In this way we can safegaurd that the data is available.

Precondition:

All layers have been instanciated with a number of neurons to create, a parent layer if any, and a child layer if any.

Postcondition:

The current layer's data structures will all have been created based on the other related layers.

3.1.4 Member Data Documentation

3.1.4.1 GenericLayer∗ GenericLayer::parentLayer

Pointer to the parent layer of this layer.

3.1.4.2 GenericLayer∗ GenericLayer::childLayer

Pointer to the child layer of this layer.

3.1.4.3 Neuron∗ GenericLayer::neurons

The array of neurons that populate this layer of the network.

3.1.4.4 int GenericLayer::numNeurons

The number of neurons to create in this layer.

3.1.4.5 double∗∗ GenericLayer::weights

The 2D array of weights with dimensions of this layer's numNeurons by the child layer's numNeruons. If there is no child layer then it remains NULL.

The documentation for this class was generated from the following files:

- src/GenericLayer.h
- src/GenericLayer.cpp

3.2 NeuralNet Class Reference

```
#include <NeuralNet.h>
```

Public Member Functions

- NeuralNet (int numInput, int numHidden, int numOutput)
- void train ()
- void compute ()
- void calculateNeuronValues (GenericLayer *layer)
- double logisticActivation (double x)
- void calculateLocalGradients (GenericLayer *layer)

Public Attributes

- GenericLayer * input
- GenericLayer * hidden
- GenericLayer * output
- int numInput
- int numHidden
- int numOutput
- double * trainingInput
- double * desiredOutput
- double * inputData

3.2.1 Detailed Description

This class brings together the work done by various parts of the Neural system. It is the binding layer between the neural layers and the data structures inherent in the system. As such, it is responsible for initializing, stucturing, training, and calculating the neural network environment.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 NeuralNet::NeuralNet (int *numInput*, int *numHidden*, int *numOutput*)

Parameters:

numInput The number of neurons to create in the input layer.

numHidden The number of neurons to create in the hidden layer.

numOutput The number of neurons to create in the output layer.

3.2.3 Member Function Documentation

3.2.3.1 void NeuralNet::train ()

Precondition:

The trainingData and desiredValues arrays have been assigned and are of the appropriate dimensions for this particular NN.

Postcondition:

All the weights and biases are set according to the trainingData and the NN is ready for real environment data.

3.2.3.2 void NeuralNet::compute ()

Precondition:

The NN has been initialized and trained, and the inputData array has been assigned and is of the appropriate dimensions for this particular NN.

Postcondition:

The output layer nodes now contain, based on the NN's weights and biases, the output values for the given data set.

3.2.3.3 void NeuralNet::calculateNeuronValues (GenericLayer $*$ *layer*)

To calculate the value of each neuron we calculate the sum of the weights connected to each neuron multiplied by the value of each corresponding neuronal value, finally adding the adjusted bias and passing this value through an activation function.

The formula we use is given as follows, where y_i is the i-th neuron on the parent layer and w_{ji} is the weight from the parent to the neuron whose value we are calculating (note that in this notation, for simplicity, we denote the bias and its weight as the first element): $v_j(n) = \sum_{i=0}^{n} y_i(n)w_{ji}(n)$

And similarly we note that the final value of this neuron is $y(n)_j = \phi_j(v_j(n))$ where $\phi_j()$ is the activation function on neuron j.

3.2.3.4 double NeuralNet::logisticActivation (double *x*)

Used for computing the value of a neuron after calculating the raw value. We use an easily differentiable function so that it is easier later on to calculate the change in weights. The particular function we use here is the sigmoidal activation function, which constrains the output to between 0 and +1, a good fit for NNs.

The precise function we use is $\phi_j(x) = \frac{1}{1 + exp(-x)}$.

3.2.3.5 void NeuralNet::calculateLocalGradients (GenericLayer $*$ *layer*)

We calculate the errors of the each of the neurons in the NN here for use in adjusting their weights and biases for training purposes. There are three main cases for a three layer feed forward NN.

The first is the input layer, and since the desired value is always equal to the actual value, since it is given, the gradient is 0.

The second is the case of a hidden layer, where the errors are calculated from the layer nearest to the output layer to the layer closest to the input layer. This is done using the back propogation algorithm, which, given the gradients of adjacent layers, calculates the gradients recursively working backwords and using the derivative of the activation function used for calculating neuron values. This is done since the error signals cannot be determined for hidden layers since there is no value to compare their output to. It can be written as $\gamma_j(n) = \phi_j^{'}(v_j(n)) \sum_{k=0}^{m} \gamma_k(n) w_{kj}(n)$ where neuron j is the gradient we are calculating, neuron k is in the child layer and m is the number of neurons on that layer.

The third case is the output layer, where the error is trivially desired-actual. The gradient is then defined much the same as case 2 where it is the error multiplied by the derivative of the activation function applied to the value of the neuron. It can be written as $\gamma_j(n) = e_j(n)\phi_j^{'}(v_j(n))$.

Parameters:

layer The layer for which we are to calculate the errors.

3.2.4 Member Data Documentation

3.2.4.1 GenericLayer∗ NeuralNet::input

The input layer to the neural network.

3.2.4.2 GenericLayer∗ NeuralNet::hidden

The hidden layer to the neural network.

3.2.4.3 GenericLayer∗ NeuralNet::output

The output layer to the neural network.

3.2.4.4 int NeuralNet::numInput

The number of input neurons to create.

3.2.4.5 int NeuralNet::numHidden

The number of hidden neurons to create.

3.2.4.6 int NeuralNet::numOutput

The number of output neurons to create.

3.2.4.7 double∗ NeuralNet::trainingInput

The data that the network will be trained against. Each index corresponds to an input neuron. This is used in conjuction with desiredValues to train the network.

3.2.4.8 double∗ NeuralNet::desiredOutput

The expected results from the training data. Each element is related to each output neuron's expected value

3.2.4.9 double∗ NeuralNet::inputData

The data to be calculated from the environment, in the same form as each set of data in trainingData.

The documentation for this class was generated from the following files:

- src/NeuralNet.h
- src/NeuralNet.cpp

3.3 Neuron Class Reference

```
#include <Neuron.h>
```

Public Member Functions

- Neuron ()
- virtual ∼Neuron ()

Public Attributes

- double value
- double bias
- double biasWeight
- double error

3.3.1 Detailed Description

Provides the structure of the neuron. The Neuron class has details about the neuron's value, which is the value stored by the neruon. The neuron's bias, which is used and adjusted when training and computing new values for the network. The neuron bias' weight which is used similarly to the bias. And also the error, which is used primarily in training for computing how far off the network's output is from the desired output. All of this is used generically and in conjuction with the GenericLayer class to implement any sort of nerual network layout.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Neuron::Neuron ()

Defines the constructor for the Neuron class.

3.3.2.2 Neuron::∼Neuron () `[virtual]`

Neuron class desctructor.

3.3.3 Member Data Documentation

3.3.3.1 double Neuron::value

The value of the neuron.

3.3.3.2 double Neuron::bias

The bias of the neuron.

3.3.3.3 double Neuron::biasWeight

The weight of the bias on the neuron.

3.3.3.4 double Neuron::error

The error associated with the computation of the network versus the desired output.

The documentation for this class was generated from the following files:

- src/Neuron.h
- src/Neuron.cpp

Chapter 4

BreakingCaptcha File Documentation

4.1 src/GenericLayer.cpp File Reference

```
#include "GenericLayer.h"
#include <omp.h>
#include <cstdlib>
#include <math.h>
```

4.1.1 Detailed Description

Author:

Ben Snider

Version:

0.1

Defines the GenericLayer class for the implementation of the network.

4.2 src/GenericLayer.h File Reference

```
#include "Neuron.h"
```

Classes

- class GenericLayer

4.2.1 Detailed Description

Author:

Ben Snider

Version:

0.1

Header file for the GenericLayer class.

4.3 src/NeuralNet.cpp File Reference

```
#include "NeuralNet.h"
#include <cstdlib>
```

4.3.1 Detailed Description

Author:

Ben Snider

Version:

0.1

Implements the NeuralNet class.

4.4 src/NeuralNet.h File Reference

```
#include "GenericLayer.h"
```

Classes

- class NeuralNet

4.4.1 Detailed Description

Author:

Ben Snider

Version:

0.1

Defines the NeuralNet class.

4.5 src/Neuron.cpp File Reference

```
#include "Neuron.h"
```

4.5.1 Detailed Description

Author:

Ben Snider

Version:

0.1

Implements the Neuron class.

4.6 src/Neuron.h File Reference

Classes

- class Neuron

4.6.1 Detailed Description

Author:

Ben Snider

Version:

0.1

Defines the Neuron class.

Index