Contents

Chapter 1

TODOs

1.1 General

Description, Header (File-Header too), Comments

1.2 class Neuron

class NeuronFloat implements class Neuron?

1.3 class Connection

connectionFloat

1.4 class MultiLayerPerceptron

Constructor mit Connections fuer input layer: each: NrConnections = hidden, twoGroups: NrConnections = hidden/2, one: NrConnections = inputNeur
Constructor mit + Connections fuer hidden Layer: each, cross und zigzag
Constructor mit + Output topologies each und groups
Constructor mit hidden neurons rising from inputs to variable / layer
Constructor mit Float

Chapter 2

class Neuron

2.1 Attributes

int inputs For all inputs one integer variable is used. So the neuron can handle up to 32 inputs, which can just be 0 or 1.

int nrInputs Specifies how many inputs (respectively bits of the variable `inputs`) shall be used.

int netInput The result of the net input function:
$netInput = input.1 * weights[0] + ... + input.32 * weights[31]$
So far also used as output.

int weights[] The weights of the inputs. Array size depends on the number of inputs.

int threshold The threshold when output is activated.

Boolean activated Boolean output, true if the calculated output is >= the threshold.

2.2 Constructors

Neuron(int nrInputs) Sets `nrInputs`, size of array `weights` and initialzes `inputs`, `netInput`, and `weights` to 0, `activated` to false, and `threshold` to max ($0x7fffffff$).

2.3 Methods

Boolean setInput(int nrInput) Sets given input to 1 by shifting a 1 to the corresponding bit in `inputs`.
Returns true if input was set, returns false if input number is to high.

Boolean unsetInput(int nrInput) Sets given input to 0 by shifting a 1 to the corresponding bit in `inputs` and bitwise inverting it.
Returns true if input was unset, returns false if input number is to high.

Boolean setWeight(int nrInput, int weight) Puts the given weight into the for the given input corresponding field in the weights array.
Returns true if weight was set, returns false if input number is to high.

void setThreshold(int threshold) Sets threshold for the neuron.

int getOutput() Calculates the `netInput`, the result of the net input function, and uses the activation function for returning the output.
So far, there is no extra output variable in the class, since the output function is just the identity. So the method returns the variable `netInput`.
So far, the activation function is just a step function, which is implemented as a simple if-else.

Boolean getActivation() Returns `activated`, which is true if the calculated output is >= the threshold.
Note! So far, activated can just be called if `getOutput` was called before

Chapter 3

class NeuronFloat

Same as class Neuron, just using floats for weights, threshold and output.

3.1 Attributes

int inputs

int nrInputs

float netInput

float weights[]

float threshold

Boolean activated

3.2 Constructors

NeuronFloat(int nrInputs) Same as `Neuron(int nrInputs)`.

Methods Same as in `class Neuron`.

Chapter 4

class Connection

Connects the output of a neuron to an input of another neuron. The weight of the following input is the output value times the weight of the connection.

4.1 Attributes

Neuron neuronFrom Neuron the output is taken from.

Neuron neuronTo Neuron the input is set.

int input Input of `neuronTo` which is set.

int connWeight Weight of the connection.

int weightToSet Weight set in `neuronTo`. Output of `neuronFrom` $*$ weight of connection.

4.2 Constructors

Connection(Neuron neuronFrom, Neuron neuronTo, int input, int connWeight)
Just sets `neuronFrom`, `neuronTo`, `input`, and `connWeight`.

4.3 Methods

void run() Calculates the weight for `neuronTo`. The weight, which is set, is calculated by `getOutput()` from `neuronFrom` $*$ the weight of the connection.for `neuronTo`. The calculated weight is set to the given input.
The method uses `getActivation()` from `neuronFrom` to set or unset the given input of `neuronTo`.

Chapter 5

class MultiLayerPerceptron

Automatically builds a multi-layer perceptron. The number of input neurons and output neurons can be set in any case, more possibilities depends on the constructors.
The maximum number of neurons for one layer is 32.

5.1 Attributes

Neuron inputNeuron[] Input layer, size depends on definition.

Neuron hiddenNeuron[][] Hidden layer(s), dimension 1 depends on how many hidden layers are defined (or calculated). Dimension 2 depends on how many hidden neurons are defined (or calculated).

Neuron outputNeuron[] Output layer, size depends on definition.

int hiddenLayers Size of 1st dimension of array `hiddenNeuron`. Depending on the constructor this value may be settable or be calculated in the constructor.

int inputConnWeights[] Weights of the connections between input layer and 1st hidden layer.

int hiddenConnWeights[][] Weights of the connections between the hidden layers (if more than 1 is defined or calculated). The size of the 1st dimension of this array is set by `hiddenLayers`.

int outputConnWeights[] Weights of the connections between the last hidden layer and the output layer.

int outputVector[] The result of the multi layer perceptron, is returned by the method `run()`. The size of the array depends on the number of output neurons.

Connection inputConnection[] Connections between input layer and 1st hidden layer.

Connection hiddenConnection[][] Connections between the hidden layers (if more than 1 is defined or calculated). The size of the 1st dimension of this array is set by `hiddenLayers`.

Connection outputConnection[] Connections between the last hidden layer and the output layer.

5.2 Constructors

MultiLayerPerceptron(int inputNeurons, int hiddenNeuronsPerLayer, int hiddenLayers, int outputNeurons) The constructor creates a multi-layer perceptron with the given number of input neurons, number of hidden neurons per layer, number of hidden layers and number of output neurons.
Each input neuron is implemented with one input. All neurons will be connected automatically to each neuron of the following layer, except of the output layer. So the number of inputs for the hidden neurons is calculated by the number of input neurons for the first layer and by the number of hidden neurons for all hidden layers and, in case of just one output neuron, for the output layer. If there are more than one output neuron the outputs of the last hidden layer are split to the output neurons, the lower hidden neurons to the lower output neurons etc., so the number of inputs will be calculated by dividing the number of hidden neurons by the number of output neurons. Therefore it is not possible to have more output neurons than hidden neurons and there must be at least one hidden layer. In case the division has a remainder one more input is added to each output neuron.
The weights and the thresholds of all neurons and the weights of all connections will be initialised with 1.
The constructor calculates the amount of weights for each layer, stored in the corresponding arrays and then sets the connections.
The output vector is set to 0.

5.3 Methods

int[] run(int inputVector) The method just executes the built multi-layer perceptron. The inputs of the input neurons are set with the given input vector. Than the connections of the different layers are executed one after another, starting with the first connection of the input layer, by calling the `run()`-methods of the connections. Finally, the `getOutput()`-methods of the output-neurons are called and the output vector returned.

Chapter 6

Prospective Enhancements

6.1 class Neuron64 with type long for inputs

Using type long instead of int so neuron can handle up to 64 inputs.

6.2 Activation function and Output function settable

(Into extra classes so class Neuron remains as small as possible)
So far neuron just uses step function.
(siehe ComputaionalIntelligence S.52)
Sprungfunktion: if (netInput ¿= threshold) return netInput; else return 0;
semi-lineare Funktion: if (netInput ¿ threshold + 1/2) return netInput;
else if ((netInput ¡ threshold + 1/2) && (netInput ¿ threshold - 1/2)) output = (netInput - threshold) + 1/2;
else return 0;
Sinus bis Saettigung: if (netInput ¿ threshold + pi/2) return netInput;
else if ((netInput ¡ threshold + pi/2) && (netInput ¿ threshold - pi/2)) output = (sin(netInput - threshold) + 1)/2;
else return 0;
logistische Funktion: output = 1 / (1 + $e^{(-(netInput-threshold))}$) // geht nur von 0 - 1
radiale Basis Funktionen
enum fÃ¼r Funktionen in Klasse Ã¼ber Neuron

Chapter 7

Test results

7.1 Unit test for NeuronFloat