TriMP

v0.1

Generated by Doxygen 1.8.17

Chapter 1

TriMP - Trivial Multilayer Perceptron

1.1 Project Description

1.1.1 Why Another MLP Implementation?

There are numerous implementations of artificial neural nets. In my opinion, however, it is much more interesting, and much more practical (in certain cases) to give it a bit more personalized approach. After all, it is always good to know what goes behind the scene.

1.1.2 Multilayer Perceptron

The decision was made to begin with the type of Artificial Neural Nets known as Multilayer Perceptron as the simplest one and the easiest to start with. There is no intention to fill this document with all the painful math stuff here. You may easily find it on the Internet.

1.1.3 Future Plans

There are plans to keep up the development by adding more types of Artificial Neural Nets, as well as adding genetic algorithms for finding the optimal topology and for finding the optimal fundamental parameters, such as the learing rate and momentum.

Author

Alexey Lyashko

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Module Documentation

3.1 Neuron Activation

Neuron activation type definitions.

Enumerations

- enum activation_type { ,
  Exponential, HTangent, Gauss, Softsign,
  Sinusoid, Binary }

  Enumeration of the available neuron activation types.

3.1.1 Detailed Description

Neuron activation type definitions.

3.1.2 Enumeration Type Documentation

3.1.2.1 activation_type

```
enum activation_type
```

Enumeration of the available neuron activation types.

Enumerator

Exponential	Denotes the exponential (logistic) activation function. Range [0, 1].
HTangent	Denotes the hyperbolic tangent activation function. Range [-1, 1].
Gauss	Denotes the Gauss activation function. Range [0, 1].
Softsign	Denotes the Softsign activation function. Range [-1, 1].
Sinusoid	Denotes the Sinusoid activation function. Range [-1, 1].
Binary	Denotes the Binary Step activation funciton. Range [0, 1].

3.2 Multilayer Perceptron

Enumerations

- enum net_type { MLP, Elman }

Functions

- DLL_EXPORT pnet_t Net_new (int numLayers, int ∗neuronsPerLayer, enum activation_type ∗neuron↩ActivationType)
- DLL_EXPORT pnet_t Net_new_ex (int numLayers, int ∗neuronsPerLayer, enum activation_type ∗∗neuron↩ActivationType)
- DLL_EXPORT void Net_free (pnet_t net)
- DLL_EXPORT double NetTrain (pnet_t net, double ∗inputData, double ∗outputData, double ∗expectedResult)
- DLL_EXPORT void NetExec (pnet_t net, double ∗inputData, double ∗outputData)
- DLL_EXPORT void NetSetType (pnet_t net, enum net_type type)
- DLL_EXPORT void NetSetRate (pnet_t net, double learningRate)
- DLL_EXPORT void NetSetMomentum (pnet_t net, double momentum)

3.2.1 Detailed Description

3.2.2 Enumeration Type Documentation

3.2.2.1 net_type

```
enum net_type
```

MLP type

Enumerator

MLP	The net is a regular Multilayer Perceptron.
Elman	The net is an implementation of the Elman recurrent net.

3.2.3 Function Documentation

3.2.3.1 Net_free()

```
DLL_EXPORT void Net_free (
            pnet_t net )
```

Releases all the resources allocated for the net and destroys the net itself.

Parameters

net	Pointer to a net to be freed.

3.2.3.2 Net_new()

```
DLL_EXPORT pnet_t Net_new (
             int numLayers,
             int * neuronsPerLayer,
             enum activation_type * neuronActivationType )
```

Allocates new multilayer perceptron

Parameters

numLayers	Number of layers in the net (including the input and output layers).
neuronsPerLayer	Pointer to an array of int holding number of neurons for each layer.
neuronActivationType	Pointer to an array of enum activation_type values for each layer.

Returns

Pointer to the newly allocated net or NULL.

See also

activation_type

3.2.3.3 Net_new_ex()

```
DLL_EXPORT pnet_t Net_new_ex (
             int numLayers,
             int * neuronsPerLayer,
             enum activation_type ** neuronActivationType )
```

Allocates new multilayer perceptron with the ability to set different activation types for each neuron (even withing a single layer).

Parameters

numLayers	Number of layers in the net (including the input and output layers).
neuronsPerLayer	Pointer to an array of int holding number of neurons for each layer.
neuronActivationType	Pointer to array of pointers to arrays of activation type values for each neuron in each layer.

Returns

Pointer to the newly allocated net or NULL.

See also

activation_type

3.2.3.4 NetExec()

```
DLL_EXPORT void NetExec (
            pnet_t net,
            double * inputData,
            double * outputData )
```

Runs a single iteration of data processing. The function does not have a way to verify the capacities of the input↵ Data, outputData and the expectedResult arrays, therefore, it is user's responsibility to guarantee that the capacity of the inputData array at least equals the amount of the input neurons, and that the capacity of the outputData array at least equals the amount of the output neurons.

Parameters

net	Pointer to the net.
inputData	Array of input values.
outputData	Array that receives the output values.

3.2.3.5 NetSetMomentum()

```
DLL_EXPORT void NetSetMomentum (
            pnet_t net,
            double momentum )
```

Sets the value of the momentum.

Parameters

net	Pointer to the net.
momentum	Value of the momentum.

3.2.3.6 NetSetRate()

```
DLL_EXPORT void NetSetRate (
            pnet_t net,
            double learningRate )
```

Sets the learning rate for the net.

Parameters

net	Pointer to the net.
learningRate	Value of the learning rate.

3.2.3.7 NetSetType()

```
DLL_EXPORT void NetSetType (
            pnet_t net,
            enum net_type type )
```

Sets the type of the net

Parameters

net	Pointer to the net.
type	Type of the network (MLP or Elman).

See also

net_type

3.2.3.8 NetTrain()

```
DLL_EXPORT double NetTrain (
            pnet_t net,
            double * inputData,
            double * outputData,
            double * expectedResult )
```

Runs a single iteration of the network training cycle. The function does not have a way to verify the capacities of the inputData, outputData and the expectedResult arrays, therefore, it is user's responsibility to guarantee that the capacity of the inputData array at least equals the amount of the input neurons, and that the capacities of the outputData and the expectedResult arrays at least equal the amount of the output neurons.

Parameters

net	Pointer to the net.
inputData	Array of input values.
outputData	Array that receives the output values.
expectedResult	Array of values containing the expected results.

Returns

Net's error value.

Index