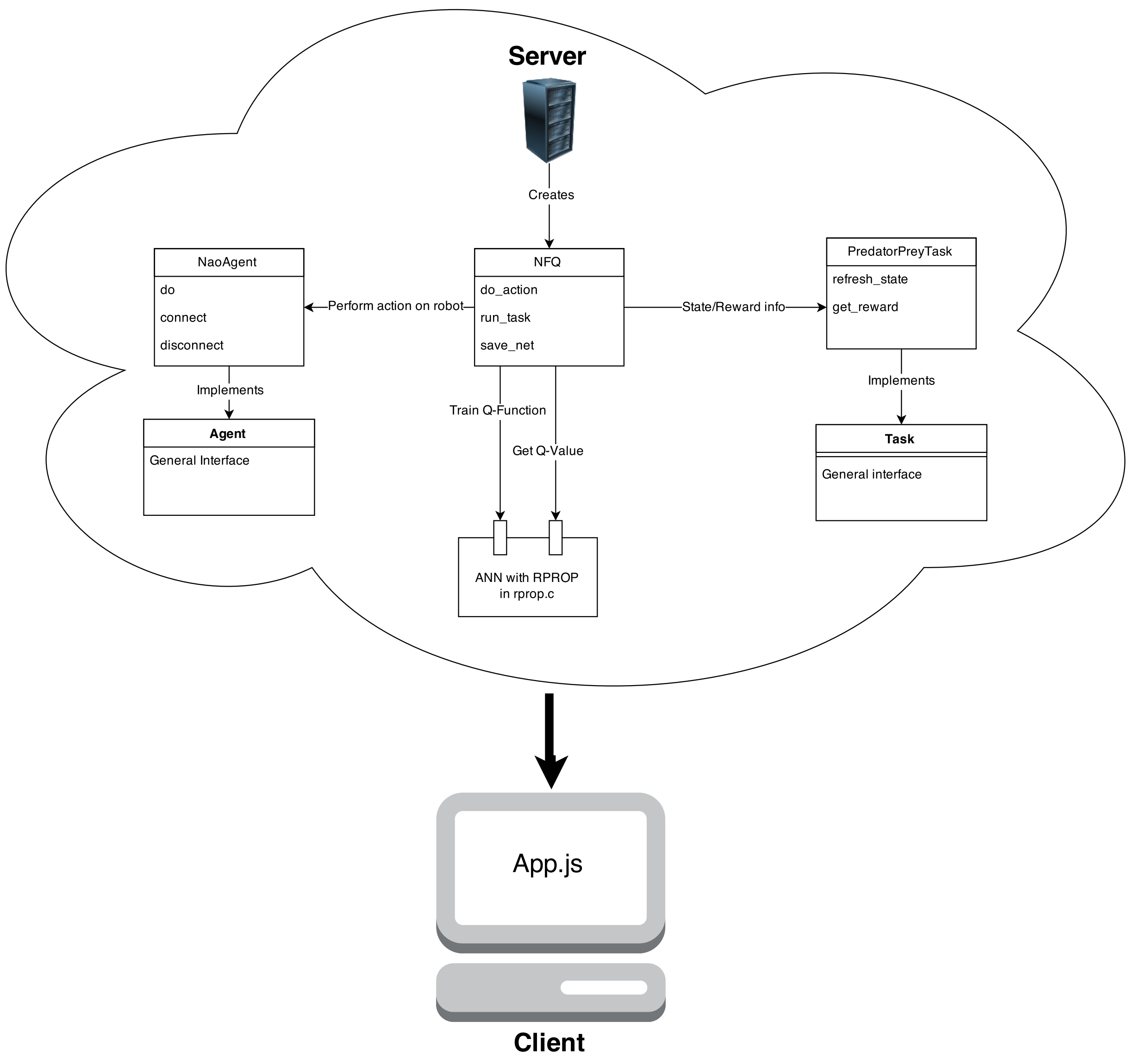

Figure 1: Class/dependency diagram

Contents

1 Data Structure Documentation

1.1 Agent Class Reference

Provides a default interface for Agent class.

Inherited by NaoAgent.

Public Member Functions

- def connect

  This method should handle connection to a real world agent.
- def disconnect

  This methods should handle disconnection from a real world agent.
- def do

  This methods handles performing selected action according to action number on a real world agent.

1.1.1 Detailed Description

Provides a default interface for Agent class.

1.1.2 Member Function Documentation

1.1.2.1 def connect (*self*)

This method should handle connection to a real world agent.

This method must be implemented

1.1.2.2 def disconnect (*self*)

This methods should handle disconnection from a real world agent.

This method must be implemented

1.1.2.3 def do (*self, action_number*)

This methods handles performing selected action according to action number on a real world agent.

This method must be implemented

Parameters

action_number	Number of action to be performed

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

- Agent.py

1.2 ann_t Struct Reference

Struct representing a neural network.

Data Fields

- double x [Nx]
- double y [Nx]
- double delta [Nx]
- double prevGrad [Nx][Nx]
- double currGrad [Nx][Nx]
- double updateValue [Nx][Nx]
- double wDelta [Nx][Nx]
- double w [Nx][Nx]
- double dv [Nou]

1.2.1 Detailed Description

Struct representing a neural network.

Container for a complex MLP structure.

1.2.2 Field Documentation

1.2.2.1 double currGrad[Nx][Nx]

Weight current error gradient

1.2.2.2 double delta[Nx]

Delta value on neurons

1.2.2.3 double dv[Nou]

Target value on output neurons

1.2.2.4 double prevGrad[Nx][Nx]

Weight error gradient from last step

1.2.2.5 double updateValue[Nx][Nx]

Update value according to RPROP algorithm for each weight

1.2.2.6 double w[Nx][Nx]

Weights matrix

1.2.2.7 double wDelta[Nx][Nx]

Delta of weights change

1.2.2.8 double x[Nx]

Input of neurons

1.2.2.9 double y[Nx]

Output of neurons

The documentation for this struct was generated from the following file:

- rprop.c

1.3 MainHandler Class Reference

Operates a web server for web application.

Inherits RequestHandler.

Public Member Functions

- def get

  Handles asynchronous web request and renders a web app.

1.3.1 Detailed Description

Operates a web server for web application.

1.3.2 Member Function Documentation

1.3.2.1 def get (*request*)

Handles asynchronous web request and renders a web app.

Parameters

request	Web request received

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

- server.py

1.4 NaoAgent Class Reference

Implementation of Agent interface on Nao robot in Webots simulator.

Inherits Agent.

Public Member Functions

- def __init__

  Constructor method setups default parameters.
- def connect

  Method handles connection to simulated robot and initialization of its features.
- def disconnect

  Disconnects from simulated Nao.
- def do

  Perform specified action from the set of all available motions.
- def go_forward

  This method performs movements along the X-axis, with 0.25m distance.
- def go_right

  This method performs a rotation 30 degrees to the right.
- def go_left

  This method perform movement 30 degrees left.
- def StiffnessOn

  Sets the stiffnes of Nao's joints to maximal value, to make him ready to perform motions.

Data Fields

- robot_ip

  Default connection IP address.
- robot_port

  Default connection port.

1.4.1 Detailed Description

Implementation of Agent interface on Nao robot in Webots simulator.

1.4.2 Member Function Documentation

1.4.2.1 def connect (*self*, *ip* = `None`, *port* = `None`)

Method handles connection to simulated robot and initialization of its features.

It connection to the robot on specified Ip:port and initializes the robot to the init position, preparing it for motions.

Parameters

ip	IP address of a robot
port	Port number of a robot

Returns

True, if connection was succesful, else False

1.4.2.2 def do (*self*, *action_number*)

Perform specified action from the set of all available motions.

Parameters

action_number	specifies position of action in the action list

1.4.2.3 def StiffnessOn (*self*, *proxy*)

Sets the stiffnes of Nao's joints to maximal value, to make him ready to perform motions.

Parameters

proxy	specifies a proxy(path) for setting the joint's stiffness

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

- NaoAgent.py

1.5 NFQ Class Reference

This class handles mechanism behind NFQ learning and also decision process.

Public Member Functions

- def __init__

  Constructor of the class.
- def save_net

Saves current network to database.
- def load_net

  Loads and update current network weights with the network saved in database.
- def load_tasks

  Loads a list of all previously learned tasks from a database.
- def load_samples

  Loads a previously collected samples from the database and update current ones with them.
- def write_samples

  Upload current training samples set into database in JSON format.
- def sample_to_patterns

  This method provides basic transformation from the training sample into training patterns.
- def get_training_set_from_samples

  Transform whole set of training samples into training patterns according to the proposed algorithm.
- def learn

  The key method of this class, if target state is reached, this method updates a current Q-function.
- def choose_controller

  This method controls the decision process according to current autonomy level.
- def run_task

  Runs a task saved in database with a specific name.
- def make_step_task

  This methods handles steps and reward counting during performing loaded tasks.
- def do_action

  Method representing an elementary step of agent in the environment.
- def getQ

  General method for acquiring the Q-value for current state-action pair.
- def minQ

  Gets a minimum Q-value in current state for available actions.
- def minQ_index

  Gets an index of action with minimal Q-value in current state.
- def maxQ

  Gets a maximal Q-value in current state for available actions.
- def maxQ_index

  Gets an index of action with a maximal Q-value in current state.
- def get_target

  Calculates a target Q-value according to Bellman equation mentioned in the work.

Data Fields

- agent

  An Agent performs actions in the environment.
- task

  A Task object provides reward function and also environment state update.
- connected

  State variable signalizing if Agent is connected to the real robot.
- redis_server

  Connection to the Redis database.
- epoch

  Episode counter.
- discount

  Discount factor constant for learning process.
- network

Holds weights of the neural network.

- samples

  Set of training samples in a form of tuples according to work.
- training_set

  Set of training pattern, which can be applied directly to the neural network.
- state

  Representation of environment state.
- step_counter

  Counts steps made during current episode.

Static Public Attributes

- list action_codes

  Encoding of action number for purposes of neural network input.

1.5.1 Detailed Description

This class handles mechanism behind NFQ learning and also decision process.

It contains definitions for methods that handles communication with database, operating the neural network, performing tasks and finally the NFQ learning and decision procedure.

```
@author Jan Gamec
@version 1.0
```

1.5.2 Constructor & Destructor Documentation

1.5.2.1 def __init__ (*self*, *discount* = `0.99`)

Constructor of the class.

Handles basic configuration of the NFQ module and its constants. It's arranging connection with Redis database and initializes Agent and Task objects, that are necessary for algorithm. It handles initialization of neural network keeping learned knowledge if no record of network configuration is in database. It also loads information about previous learning process from the database and parses previous samples intro training patterns.

Parameters

discount	Discount factor constant used during learning

1.5.3 Member Function Documentation

1.5.3.1 def choose_controller (*self*)

This method controls the decision process according to current autonomy level.

It either asks a humna operator for action or decides according to the policy

1.5.3.2 def do_action (*self*, *action_num*, *autonomy* = `False`)

Method representing an elementary step of agent in the environment.

Given the action number, method make an Agent do selected action and collect a sample information. Samples are collected in a tuples according to work.

[state at t, action number, state at t+1, epoch, step]

If total autonomy (in a case of performing leaded task) is off, sample is appended to current sample set. This method then delegates decision process back to the learn() method in a recursive way

Parameters

action_num	Number of action in the action set
autonomy	Default value is False. During autonomy samples are not stored

1.5.3.3 def get_target (*self*, *state* = `None`, *step* = `None`)

Calculates a target Q-value according to Bellman equation mentioned in the work.

Parameters

state	Represents an environment state. Default is current state.
step	Time step. Default value is step count in current episode

Returns

Returns the target Q-value for a neural network training

1.5.3.4 def get_training_set_from_samples (*self*)

Transform whole set of training samples into training patterns according to the proposed algorithm.

This method updates current training set used for training with the generated one

1.5.3.5 def getQ (*self*, *state*, *operator*)

General method for acquiring the Q-value for current state-action pair.

This method runs a neural network forward in order to get Q-value for current state

Parameters

state	Represents state of environment
operator	Function handle, e.g. we want minimal Q-value in current state for available actions, we use function min()

Returns

A pair, where first value corresponds to the actual Q-value and second is the index of the action having this value

1.5.3.6 def learn (*self*)

The key method of this class, if target state is reached, this method updates a current Q-function.

Update is realised training a neural network with a training patternset generated from samples collected during previous episodes. It also updates information about the learning progress to the database (Episode no., Number of steps) It delegates recursive control process to the choose_controller

See also

choose_controller method

1.5.3.7 def load_net (*self*, *name* = `'net'`)

Loads and update current network weights with the network saved in database.

Parameters

name	Specifies the name of network in the database

1.5.3.8 def load_tasks (*self*)

Loads a list of all previously learned tasks from a database.

Returns

List of all learned tasks

1.5.3.9 def make_step_task (*self*, *reward* = 0)

This methods handles steps and reward counting during performing loaded tasks.

It works in the recursive way. If target area or step limit is not reached, agent make a step and cumulate the reward. The method calls itself until target state or step limit is reached.

Parameters

reward	Represents current cumulative reward

Returns

Reward after end of task performance

1.5.3.10 def maxQ (*self*, *state*)

Gets a maximal Q-value in current state for available actions.

Parameters

state	Represents state of environment

Returns

maximal Q-value for current state

1.5.3.11 def maxQ_index (*self*, *state*)

Gets an index of action with a maximal Q-value in current state.

Parameters

state	Represents state of environment

Returns

Index of the action with a maximal Q-value

1.5.3.12 def minQ (*self*, *state*)

Gets a minimum Q-value in current state for available actions.

Parameters

state	Represents state of environment

Returns

Minimal Q-value for current state

1.5.3.13 def minQ_index (*self*, *state*)

Gets an index of action with minimal Q-value in current state.

Parameters

state	Represents state of environment

Returns

Index of the action with minimal Q-value

1.5.3.14 def run_task (*self*, *name*)

Runs a task saved in database with a specific name.

The learned task is represented by its NN weights configuration which are loaded and replaces current network configuration during the task performance. Original network is restored after then.

Parameters

name	Specifies a name of task saved in database

Returns

Returns cumulative reward after task was performed

1.5.3.15 def sample_to_patterns (*self*, *sample*)

This method provides basic transformation from the training sample into training patterns.

Training patterns are generated according to the algorithm in the work, where for action selected in sample pattern with minimal Q-value is generated and for all other actions patterns with maximal Q are generated

Parameters

sample	Training sample in the form of tuple

Returns

List of training patterns

1.5.3.16 def save_net (*self*, *name* = `'net'`)

Saves current network to database.

Parameters

name	Specifies name for the network

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

- NFQ.py

1.6 PredatorPreyTask Class Reference

Implementation of Task interface for a so called Predator-Prey task.

Inherits Task.

Public Member Functions

- def __init__

  Constructor method.
- def get_reward

  A reward function.
- def refresh_state

  Update current state loading the data from database.

Data Fields

- database

  Handles connection to the Redis database.
- state

  Stores current state.

1.6.1 Detailed Description

Implementation of Task interface for a so called Predator-Prey task.

1.6.2 Member Function Documentation

1.6.2.1 def get_reward (*self*, *state*)

A reward function.

If agent is closer to target than 0.5m cca 0.15 normalized and rotated less than 30 degrees, its considered to be a target state so the reward is 0. Else reward is equal 0.01

Parameters

state	Current environment state

Returns

Returns a reward for a give state

1.6.2.2 def refresh_state (*self*)

Update current state loading the data from database.

Returns

Returns current environment state

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

- PredatorPreyTask.py

1.7 Task Class Reference

Provides a default interface for Tasks used in NFQ.

Inherited by PredatorPreyTask.

Public Member Functions

- def get_reward

  Reward function.
- def refresh_state

  Method for updating environment state.

1.7.1 Detailed Description

Provides a default interface for Tasks used in NFQ.

1.7.2 Member Function Documentation

1.7.2.1 def get_reward (*self*, *state*)

Reward function.

This method has to be implemented

1.7.2.2 def refresh_state (*self*)

Method for updating environment state.

This method has to be implemented

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

- Task.py

1.8 WSHandler Class Reference

Handles websocket communication with web app.

Inherits WebSocketHandler.

Public Member Functions

- def open

  Handles all new connections.
- def on_message

  Handles various type of messages received over WS.
- def on_close

  Handles close of WS connection from client.

1.8.1 Detailed Description

Handles websocket communication with web app.

1.8.2 Member Function Documentation

1.8.2.1 def on_message (*self, json_message*)

Handles various type of messages received over WS.

It distinc 3 main commands:

- Performing an action.
- Saving current task.
- Running loaded task.

It also responds with corresponding data

Parameters

json_message	Data received in WS request (like action number)

1.8.2.2 def open (*self*)

Handles all new connections.

Initializes a new NFQ instance for each new WS connection

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

- server.py

2 File Documentation

2.1 _rprop.c File Reference

File containing python wrapper for a C MLP with RPROP.

```
#include <Python.h>
#include <numpy/arrayobject.h>
#include "rprop.c"
```

Functions

- static PyObject ∗ rprop_learn2 (PyObject ∗self, PyObject ∗args)

  Function wrapping a learning process of a neural network.
- static PyObject ∗ rprop_run2 (PyObject ∗self, PyObject ∗args)

  Function wrapping a function that runs a neural network forward.
- static PyObject ∗ rprop_init (PyObject ∗self, PyObject ∗args)

  Function wrapping a initialization of a network.
- PyMODINIT_FUNC init_rprop (void)

  Initializes python wrapping functions.

Variables

- static PyMethodDef module_methods [ ]

2.1.1 Detailed Description

File containing python wrapper for a C MLP with RPROP.

Author

Jan Gamec

Date

24 May 2015 This file serves as a wrapper for functionality in rprop.c . After building this script, it can be imported as a standalone python module. The module can be build by following command: python setup.py build_ext –inplace

The setup.py file **is required** to be in the same directory as this script!. In order to change configuration of network, please see documentation for rprop.c file.

See also

rprop.c

2.1.2 Function Documentation

2.1.2.1 static PyObject ∗ rprop_init (PyObject ∗ *self,* PyObject ∗ *args*) `[static]`

Function wrapping a initialization of a network.

Function in python has no input arguments. It just creates new network with random weights initialization and return this weights as a numpy array.

Parameters

self	Object pointer
args	Holds all arguments that can be passed to function in python

Returns

A numpy array holding new weights of the created network

2.1.2.2 static PyObject ∗ rprop_learn2 (PyObject ∗ *self,* PyObject ∗ *args*) `[static]`

Function wrapping a learning process of a neural network.

Function in python accepts following arguments:

- **num_of_epochs** Number of training epochs
- **patternSet** A numpy array of training patterns
- **weights** A numpy array of neural network weights

Parameters

self	Object pointer
args	Holds all arguments that can be passed to function in python

Returns

A numpy array holding new weights

2.1.2.3 static PyObject * rprop_run2 (PyObject * *self*, PyObject * *args*) [static]

Function wrapping a function that runs a neural network forward.

It runs network forward and return values on output neurons. Function in python accepts following arguments:

- **pattern** A numpy array representing one input pattern
- **weights** A numpy array of neural network weights

Parameters

self	Object pointer
args	Holds all arguments that can be passed to function in python

Returns

An Q-value on the output neuron with double precission

2.1.3 Variable Documentation

2.1.3.1 PyMethodDef module_methods[ ] [static]

Initial value:

```
= {
  {"learn", rprop_learn2, METH_VARARGS, learn_docstring},
  {"run", rprop_run2, METH_VARARGS, run_docstring},
  {"init", rprop_init, METH_VARARGS, init_docstring},
  {NULL, NULL, 0, NULL}
}
```

Definition of all functions available after the build.

2.2 app.js File Reference

Javascript application for visualisation of NFQ algorithm.

Functions

- function connectRobot ()

  Initializes new websocket connection to server.
- function serverResponse (response)

  Parses received message object from websocket connection.
- function showIntro (msg)

  Displays a introduction page.
- function showControls ()

  Displays a NFQ controls and state information.
- function saveTask ()

  Request saving current task progress to the database.
- function loadTasks (tasks)

  Creates a table of all available tasks from DB.
- function runTask (task)

  Requests a server to perform a task on a robot.
- function doAction (action_num)

  Order a robot to do selected action.
- function enableAllButtons ()

Enables all buttons on page.
- function disableAllButtons ()

  Disables all buttons on the page to block request until the current action is finished.
- function updateState (distance, angle)

  Updates current state information with a new, received over Websocket.

2.2.1 Detailed Description

Javascript application for visualisation of NFQ algorithm.

Author

Jan Gamec

Date

24 May 2015

This application is served as a webpage when running the server application.

2.2.2 Function Documentation

2.2.2.1 function connectRobot ( )

Initializes new websocket connection to server.

Establishes a connection to robot through opening a new websocket connection to server. In addition it binds corresponding callbacks for websocket events. When a new message is received, function serverResponse is called, with a received object.

2.2.2.2 function disableAllButtons ( )

Disables all buttons on the page to block request until the current action is finished.

2.2.2.3 function doAction (*action_num*)

Order a robot to do selected action.

Handles click events on each movement arrows in order to move the "real" robot.

Parameters

action_num	Number of action from action set

2.2.2.4 function enableAllButtons ( )

Enables all buttons on page.

2.2.2.5 function loadTasks (*tasks*)

Creates a table of all available tasks from DB.

Parameters

tasks	List of task names from database

2.2.2.6 function runTask (*task*)

Requests a server to perform a task on a robot.

Parameters

task	Name of the task to be performed

2.2.2.7 function saveTask ( )

Request saving current task progress to the database.

Takes a name from corresponding input box, requests a server to save the task in DB.

2.2.2.8 function serverResponse (*response*)

Parses received message object from websocket connection.

This function handles basically 4 types of responses and reactions to them:

- Confirmation about succesful connection to robot - display a control UI.
- Warning about connection error - warns about an error.
- State update - refreshes the UI.
- Result of the performed task - displays final cumulative reward after completion of the task.

Parameters

response	A message object received over websocket

2.2.2.9 function showControls ( )

Displays a NFQ controls and state information.

2.2.2.10 function showIntro (*msg*)

Displays a introduction page.

Parameters

msg	If any warning is received, display it

2.2.2.11 function updateState (*distance, angle*)

Updates current state information with a new, received over Websocket.

Parameters

distance	New distance state
angle	New angle state

2.3 rprop.c File Reference

File containing implementation of MLP with RPROP learning.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/timeb.h>
```

Data Structures

- struct ann_t

  Struct representing a neural network.

Macros

- #define Nin 5
- #define Nh1 10
- #define Nh2 10
- #define Nou 1

Functions

- int sign (double x)

  Returns the sign of given double.
- void shuffle (double ∗∗array, int n)

  Shuffles the given 2D array.
- void ann_initRprop (ann_t ∗ann)

  Rprop variables initialization. Current weights gradient is set to 0 and udpate value to 0.1.
- void ann_resetDelta (ann_t ∗ann)

  Resets all delta values on neurons.
- void ann_rndinit (ann_t ∗ann, double min, double max)

  Randomly initializes weights matrix withit given interval.
- void ann_init (ann_t ∗ann, double ∗∗weights)

  Initializes a network from a give weights matrix.
- static void layer_run (blk_t(ann))

  Calculates an output on one layer using output from previous.
- void MLP2_run (ann_t ∗ann)

  Simple runs of network in a forward direction Calculate output running whole network forward.
- void calculate_gradients (blk_t(ann), int out)

  Calculate weights gradients between 2 layers.
- double ∗ rprop_run (ann_t ∗ann, double ∗pattern)

  Runs a network in a forward direction with a given input pattern Calculate output running whole network forward.
- void rprop_update (blk_t(ann))

  Implementation of RPROP learning algorithm according to paper Update rules and equations are described in work. This function updates weights between 2 layers.
- void rprop_learning_step (ann_t ∗ann, int num_of_patterns, double ∗∗patternSet)

  RPROP learning step. Implementation of RPROP algorithm according to paper. This method makes one forward run throught network calculating learning variables and updating weights after then.
- void test_net (ann_t ∗ann, int num_of_patterns, double ∗∗patternSet)

  Tests a network for an error against training set.
- void rprop_learn (ann_t ∗ann, int num_of_epochs, int num_of_patterns, double ∗∗patternSet)

  Manages a learning process Repeats learning procedure for the given number of epochs and shuffles the training set. It tests the network after then.

2.3.1 Detailed Description

File containing implementation of MLP with RPROP learning.

Author

Jan Gamec

Date

24 May 2015 This module contain functions for initialization, running and training Multilayer Perceptron with RPROP learning algorithm. This file cannot be run independently, but is used as a library for python wrapper. File can be combiled: gcc -Wall -std=gnu99 -O3 -ffast-math -funroll-loops -s -o rprop_standalone rprop.c -lm

2.3.2 Macro Definition Documentation

2.3.2.1 #define Nh1 10

Defines number of neurons in first hidden layer. This needs to be changed according to task.

2.3.2.2 #define Nh2 10

Defines number of neurons in second hidden layer. This needs to be changed according to task.

2.3.2.3 #define Nin 5

Defines number of input neurons. This needs to be changed according to task.

2.3.2.4 #define Nou 1

Defines number of output neurons. This needs to be changed according to task.

2.3.3 Function Documentation

2.3.3.1 void ann_init (ann_t ∗ *ann*, double ∗∗ *weights*)

Initializes a network from a give weights matrix.

Parameters

`in,out`	*ann*	Neural network structure
	weights	Initializatioon weights matrix

2.3.3.2 void ann_initRprop (ann_t ∗ *ann*)

Rprop variables initialization. Current weights gradient is set to 0 and udpate value to 0.1.

Parameters

`in,out`	*ann*	Neural network structure

2.3.3.3 void ann_resetDelta (ann_t ∗ *ann*)

Resets all delta values on neurons.

Parameters

in,out	*ann*	Neural network structure

2.3.3.4 void ann_rndinit (ann_t * *ann*, double *min*, double *max*)

Randomly initializes weights matrix withit given interval.

Parameters

in,out	*ann*	Neural network structure
	min	Bottom weight bound
	max	Upper weight bound

2.3.3.5 void calculate_gradients (blk_t(ann) *,* int *out*)

Calculate weights gradients between 2 layers.

Parameters

blk_t(ann)	Macro representing separated 2 layers
out	Signalizes whether the layer is hidden or output/input

2.3.3.6 static void layer_run (blk_t(ann)) [static]

Calculates an output on one layer using output from previous.

Parameters

in,out	*blk_t(ann)*	Macro separating 2 layers from ann structure

2.3.3.7 void MLP2_run (ann_t * *ann*)

Simple runs of network in a forward direction Calculate output running whole network forward.

Parameters

in,out	*ann*	Neural network structure

2.3.3.8 void rprop_learn (ann_t * *ann*, int *num_of_epochs*, int *num_of_patterns*, double ** *patternSet*)

Manages a learning process Repeats learning procedure for the given number of epochs and shuffles the training set. It tests the network after then.

Parameters

in,out	*ann*	Neural network structure
	num_of_epochs	Number of training epochs
	num_of_patterns	Number of training patterns
	patternSet	Training set represented by a 2D array

2.3.3.9 void rprop_learning_step (ann_t * *ann*, int *num_of_patterns*, double ** *patternSet*)

RPROP learning step. Implementation of RPROP algorithm according to paper. This method makes one forward run throught network calculating learning variables and updating weights after then.

Parameters

`in,out`	*ann*	Neural network structure

2.3.3.10 double∗ rprop_run (ann_t ∗ *ann,* double ∗ *pattern*)

Runs a network in a forward direction with a given input pattern Calculate output running whole network forward.

Parameters

`in,out`	*ann*	Neural network structure
	pattern	Training pattern

Returns

Vector of values on the output neurons

2.3.3.11 void rprop_update (blk_t(ann))

Implementation of RPROP learning algorithm according to paper Update rules and equations are described in work. This function updates weights between 2 layers.

Parameters

blk_t(ann)	Macro separating 2 following layers

2.3.3.12 void shuffle (double ∗∗ *array,* int *n*)

Shuffles the given 2D array.

Parameters

`in,out`	*array*	Array to be shuffled
	n	length of an array

2.3.3.13 int sign (double *x*)

Returns the sign of given double.

Parameters

x	Double precission number

2.3.3.14 void test_net (ann_t ∗ *ann,* int *num_of_patterns,* double ∗∗ *patternSet*)

Tests a network for an error against training set.

Parameters

`in,out`	*ann*	Neural network structure
	num_of_pattern	Number of pattern in training set
	patternSet	Training set represented by 2D array