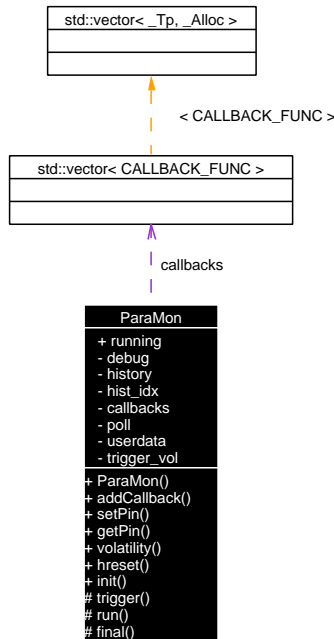


## 6.1 ParaMon Class Reference

```
#include <hwinter.h>
```

Collaboration diagram for ParaMon:



### 6.1.1 Detailed Description

This class provides a high level interface to the IEEE1284 parallel port.

It enumerates the pins and provides a layer of abstraction for debouncing inputs and indicating when they are fluctuating. This implementation uses polling - assumptions about the hardware would be required in order to implement an interrupt-driven approach.

Definition at line 35 of file `hwinter.h`.

### Public Types

- typedef `int(* CALLBACK_FUNC)(void *userdata, unsigned char pin, PINSTATE newState)`

*This is the type used for callbacks when a pin state has changed.*

- enum `PINSTATE { LOW, HIGH, VOLATILE, CHANGING }`

*These are the possible states reported by get `getPin()` and `CALLBACK_FUNC`.*

## Public Member Functions

- [ParaMon](#) (void \*userdata=0, bool trigger\_vol=false, bool debug=false, unsigned long micropoll=20000)  
*Create (don't start) a parallel port monitor.*
- void [addCallback](#) (unsigned char pin, [CALLBACK\\_FUNC](#) callback)  
*Register a callback function for a specific pin.*
- void [setPin](#) (unsigned char pin, [PINSTATE](#) state=HIGH)  
*Set the state of an output pin.*
- [PINSTATE](#) [getPin](#) (unsigned char pin)  
*Get the state of an input pin.*
- int [volatility](#) (unsigned char pin, unsigned window=HIST\_LEN, unsigned offset=0, unsigned target\_offset=0)  
*Query the history for a pin, and return a winding number representing its volatility.*
- void [hreset](#) ()  
*Reset the history.*
- void [init](#) (int pinmask)  
*Set all pins according to pinmask.*

## Public Attributes

- volatile bool [running](#)  
*True while the port is being polled. Set to false to stop.*

## Protected Member Functions

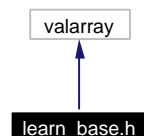
- void [trigger](#) (unsigned char pin, [PINSTATE](#) state)  
*Trigger all the callbacks for pin with the new state.*
- virtual void [run](#) (void)  
*The polling thread.*
- virtual void [final](#) (void)  
*Cleanup - currently does nothing.*

### 6.1.2 Member Typedef Documentation

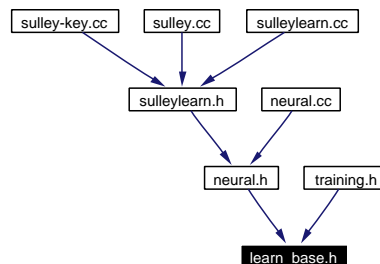
## 6.2 learn\_base.h File Reference

```
#include <valarray>
```

Include dependency graph for learn\_base.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Classifier](#)  
*A Classifier uses an Estimator of type `GenericEstimator<DOMAIN, AttributeSet>` with binary-encoded classes to classify a set of attributes, giving meaningful error, uncertainty and confidence reporting.*
- class [GenericEstimator](#)  
*A `GenericEstimator` attempts to learn a function `DOMAIN -> RANGE`.*
- class [GenericExample](#)  
*This is simply a wrapper for a `<DOMAIN, RANGE>` example pair.*
- class [GenericExampleSet](#)  
*A set of examples, for batch learning, and utility functions.*

### Typedefs

- typedef `std::valarray< double >` [AttributeSet](#)  
*This is what you would use for domain/range for most learning tasks.*

- typedef [GenericEstimator](#)< [AttributeSet](#) > [Estimator](#)  
*Most learners will extend Estimator.*
- typedef int [CLASS\\_TYPE](#)  
*The type used to enumerate classes for classification tasks.*
- typedef [GenericExampleSet](#)< [AttributeSet](#) > [ExampleSet](#)
- typedef [GenericExampleSet](#)< [AttributeSet](#), [CLASS\\_TYPE](#) > [ClassesSet](#)

## 6.2.1 Typedef Documentation

### 6.2.1.1 typedef [std::valarray](#)<[double](#)> [AttributeSet](#)

This is what you would use for domain/range for most learning tasks.

Definition at line 6 of file `learn_base.h`.

Referenced by `Classifier< DOMAIN >::learn()`, and `Classifier< DOMAIN >::prob_classify()`.

### 6.2.1.2 typedef int [CLASS\\_TYPE](#)

The type used to enumerate classes for classification tasks.

Definition at line 83 of file `learn_base.h`.

Referenced by `Classifier< DOMAIN >::classify()`, `Classifier< DOMAIN >::operator()`, `Classifier< DOMAIN >::prob_classify()`, and `Classifier< DOMAIN >::toClass()`.

### 6.2.1.3 typedef [GenericExampleSet](#)<[AttributeSet](#), [CLASS\\_TYPE](#)> [ClassesSet](#)

Definition at line 253 of file `learn_base.h`.

### 6.2.1.4 typedef [GenericEstimator](#)<[AttributeSet](#)> [Estimator](#)

Most learners will extend Estimator.

Definition at line 81 of file `learn_base.h`.

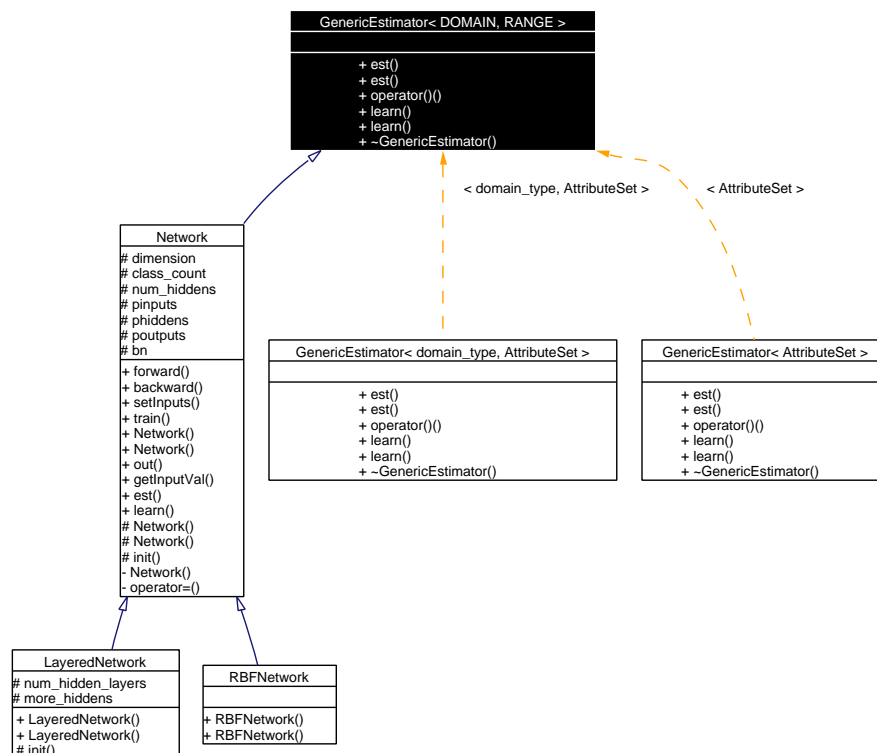
### 6.2.1.5 typedef [GenericExampleSet](#)<[AttributeSet](#)> [ExampleSet](#)

Definition at line 252 of file `learn_base.h`.

## 6.3 GenericEstimator< DOMAIN, RANGE > Class Template Reference

```
#include <learn_base.h>
```

Inheritance diagram for GenericEstimator< DOMAIN, RANGE >:



### 6.3.1 Detailed Description

```
template<class DOMAIN, class RANGE = DOMAIN> class GenericEstimator< DOMAIN, RANGE >
```

A GenericEstimator attempts to learn a function  $\text{DOMAIN} \rightarrow \text{RANGE}$ .

Definition at line 26 of file learn\_base.h.

### Public Types

- typedef DOMAIN [domain\\_type](#)
- typedef RANGE [range\\_type](#)
- typedef [GenericExample](#)< [domain\\_type](#), [range\\_type](#) > [example\\_type](#)

## Public Member Functions

- virtual double `est` (const `domain_type` &dom, `range_type` &ra)=0  
*Estimate dom into ra.*
- virtual `range_type est` (const `domain_type` &dom, double \*confidence=0)  
*Estimate the value dom.*
- `range_type operator()` (const `domain_type` &dom)  
*Convenience function: just does*  
`return est(dom);`  
.
- virtual double `learn` (const `domain_type` &dom, const `range_type` &ra, `range_type` \*actual=0, double \*confidence=0)=0  
*Learn from an example.*
- double `learn` (const `example_type` &ex, `range_type` \*actual=0, double \*confidence=0)
- virtual `~GenericEstimator` ()  
*Virtual destructor: does nothing.*

## 6.3.2 Member Typedef Documentation

### 6.3.2.1 `template<class DOMAIN, class RANGE = DOMAIN> typedef DOMAIN GenericEstimator< DOMAIN, RANGE >::domain_type`

Definition at line 28 of file `learn_base.h`.

Referenced by `GenericEstimator< DOMAIN, RANGE >::est()`.

### 6.3.2.2 `template<class DOMAIN, class RANGE = DOMAIN> typedef GenericExample<domain_type, range_type> GenericEstimator< DOMAIN, RANGE >::example_type`

Definition at line 30 of file `learn_base.h`.

### 6.3.2.3 `template<class DOMAIN, class RANGE = DOMAIN> typedef RANGE GenericEstimator< DOMAIN, RANGE >::range_type`

Definition at line 29 of file `learn_base.h`.

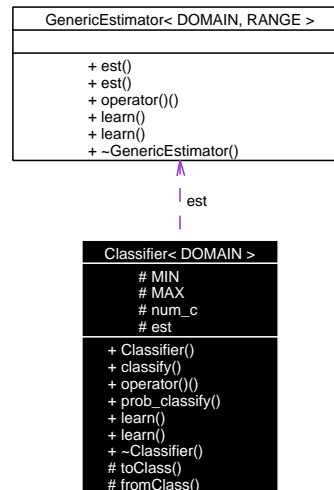
Referenced by `Network::backward()`, `GenericEstimator< DOMAIN, RANGE >::est()`, `Network::forward()`, and `Network::learn()`.

## 6.3.3 Constructor & Destructor Documentation

## 6.4 Classifier< DOMAIN > Class Template Reference

```
#include <learn_base.h>
```

Collaboration diagram for Classifier< DOMAIN >:



### 6.4.1 Detailed Description

```
template<class DOMAIN = AttributeSet> class Classifier< DOMAIN >
```

A Classifier uses an Estimator of type `GenericEstimator<DOMAIN, AttributeSet>` with binary-encoded classes to classify a set of attributes, giving meaningful error, uncertainty and confidence reporting.

#### Parameters:

*OUTPUT\_MIN* the minimum value the estimator assigns to an entry in the RANGE

*OUTPUT\_MAX* the maximum value the estimator assigns to an entry in the RANGE

*DOMAIN* the domain of the Classifier

Definition at line 95 of file `learn_base.h`.

#### Public Types

- typedef `DOMAIN` `domain_type`
- typedef `CLASS_TYPE` `range_type`
- typedef `GenericExample< domain_type, range_type >` `Example`
- typedef `GenericEstimator< domain_type, AttributeSet >` `est_type`

## Public Member Functions

- [Classifier](#) ([CLASS\\_TYPE](#) num\_classes, [est\\_type](#) \*estimator, double min=0.0, double max=1.0)  
*Construct a classifier for num\_classes using estimator.*
- virtual [CLASS\\_TYPE](#) [classify](#) (const [domain\\_type](#) &dom, double \*uncertainty=0, double \*confidence=0, double \*lrn\_conf=0)  
*Determine the most likely class for dom.*
- [CLASS\\_TYPE](#) [operator\(\)](#) (const [domain\\_type](#) &dom)  
*Does*  
return [classify](#)(dom);  
.
- template<class RNG> [CLASS\\_TYPE](#) [prob\\_classify](#) (const [domain\\_type](#) &dom, RNG &rng, [CLASS\\_TYPE](#) \*actual=0, double \*uncertainty=0, double \*confidence=0, double \*lrn\_conf=0)  
*Like classify, but returns a class probabilistically determined by the [learnt] probability distribution and rng.*
- virtual double [learn](#) (const [domain\\_type](#) &dom, const [CLASS\\_TYPE](#) &c1, double \*uncertainty=0, double \*confidence=0, double \*lrn\_conf=0, [CLASS\\_TYPE](#) \*actual=0)
- double [learn](#) (const [Example](#) &ex, double \*uncertainty=0, double \*confidence=0, double \*lrn\_conf=0, [CLASS\\_TYPE](#) \*actual=0)
- virtual [~Classifier](#) ()  
*Virtual destructor.*

## Protected Member Functions

- virtual [CLASS\\_TYPE](#) [toClass](#) (const [AttributeSet](#) &vals, double \*uncertainty=0, double \*confidence=0)  
*Converts an entry in RANGE to CLASS\_TYPE, calculating uncertainty and confidence.*
- virtual void [fromClass](#) ([CLASS\\_TYPE](#) c, [AttributeSet](#) &vals)  
*Converts a CLASS\_TYPE to an entry in the RANGE for learning.*

## Protected Attributes

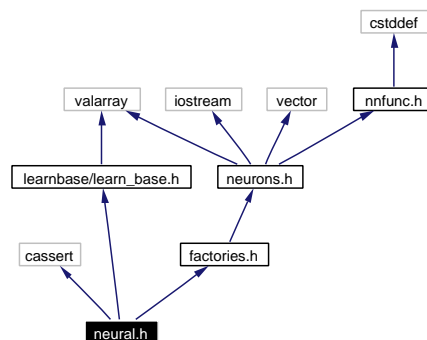
- const double [MIN](#)
- const double [MAX](#)
- unsigned [num\\_c](#)  
*The number of classes.*
- [est\\_type](#) \* [est](#)  
*The estimator being used for classifications.*

## 6.4.2 Member Typedef Documentation

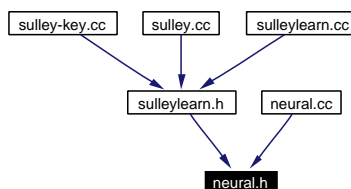
## 6.5 neural.h File Reference

```
#include <cassert>
#include "learnbase/learn_base.h"
#include "factories.h"
```

Include dependency graph for neural.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [LayeredNetwork](#)  
*LayeredNetwork implements Estimator to provide a Neural Network with a multiple hidden layers.*
- class [Network](#)  
*Network implements Estimator to provide a Neural Network with a single hidden layer.*
- class [RBFNetwork](#)  
*RBFNetwork extends Network to force RBF neurons in the (single) hidden layer.*

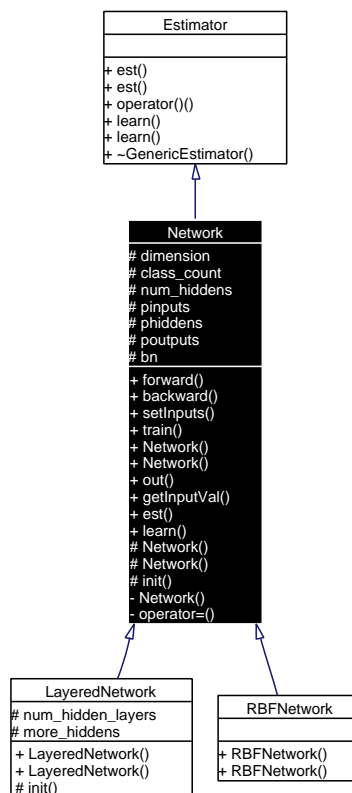
## Functions

- `std::ostream & operator<< (std::ostream &o, const Network &n)`

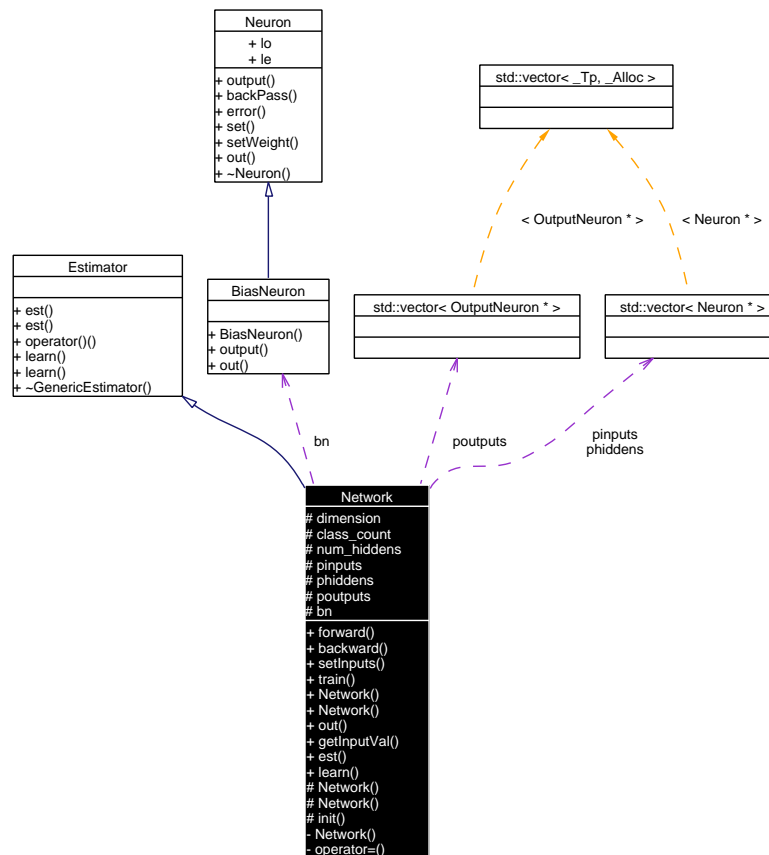
## 6.6 Network Class Reference

```
#include <neural.h>
```

Inheritance diagram for Network:



Collaboration diagram for Network:



### 6.6.1 Detailed Description

Network implements Estimator to provide a Neural Network with a single hidden layer.

This flavour of a neural network restricts all neurons to the same **type** of initialisation function and transfer function. Of course, these can be different functions or different instantiations of objects of the same type.

Definition at line 23 of file neural.h.

#### Public Member Functions

- virtual [range\\_type](#) & [forward](#) ()  
*Forward pass (legacy).*
- virtual double [backward](#) (const [range\\_type](#) &targets, bool learn=true)  
*Backward pass (legacy).*
- virtual void [setInputs](#) (const [domain\\_type](#) &attrs)  
*Set the values of the input neurons to attrs [legacy].*
- virtual double [train](#) (const [example\\_type](#) &ex)  
*Train a single example.*

- `Network` (size\_t domain\_dim, size\_t range\_dim, size\_t num\_hidden, TRANSFER\_FUNC tfunc=&logsigmoidTF, INITIALISATION\_FUNC init\_func=&rand\_init)  
*Create the Neural Network.*
- `template<class TRAN_FUNC, class INIT_FUNC> Network` (size\_t domain\_dim, size\_t range\_dim, size\_t num\_hidden, const `InputNeuronFactory`< INIT\_FUNC > &input\_factory, const `HiddenNeuronFactory`< INIT\_FUNC, TRAN\_FUNC > &hidden\_factory, const `OutputNeuronFactory`< INIT\_FUNC, TRAN\_FUNC > &output\_factory)  
*Create the Neural Network from factories.*
- virtual `std::ostream & out` (`std::ostream &o`) const  
*Formatted output operation.*
- virtual double `getInputVal` (size\_t idx)  
*Utility function for querying.*
- virtual double `est` (const `domain_type` &dom, `range_type` &ra)  
*Estimate dom into ra.*
- virtual double `learn` (const `domain_type` &dom, const `range_type` &ra, `range_type` \*actual=0, double \*confidence=0)  
*Learn from an example.*

## Protected Member Functions

- `Network` (size\_t dimension, size\_t class\_count, TRANSFER\_FUNC tfunc)
- `Network` (size\_t dimension, size\_t num\_hidden, int dummy)
- `template<class TRAN_FUNC, class INIT_FUNC> void init` (const `InputNeuronFactory`< INIT\_FUNC > &input\_factory, const `HiddenNeuronFactory`< INIT\_FUNC, TRAN\_FUNC > &hidden\_factory, const `OutputNeuronFactory`< INIT\_FUNC, TRAN\_FUNC > &output\_factory)  
*Initialisation function, from factories.*

## Protected Attributes

- size\_t `dimension`  
*The number of inputs.*
- size\_t `class_count`  
*The number of outputs [legacy name].*
- size\_t `num_hidden`  
*The [total] number of hidden neurons.*
- `std::vector< Neuron * > pinputs`  
*Pointer input neurons.*
- `std::vector< Neuron * > phiddens`

*Pointer hidden neurons [for 1 hidden layer].*

- `std::vector< OutputNeuron * > poutputs`

*Pointer output neurons.*

- `BiasNeuron bn`

*"The" bias neuron (only ever need one)*

## 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 `Network::Network (size_t dimension, size_t class_count, TRANSFER_FUNC tfunc)` [protected]

Definition at line 11 of file neural.cc.

References `TRANSFER_FUNC`.

### 6.6.2.2 `Network::Network (size_t dimension, size_t num_hidden, int dummy)` [protected]

Definition at line 17 of file neural.cc.

### 6.6.2.3 `Network::Network (size_t domain_dim, size_t range_dim, size_t num_hidden, TRANSFER_FUNC tfunc = &logsigmoidTF, INITIALISATION_FUNC init_func = &rand_init)`

Create the Neural Network.

#### Parameters:

*domain\_dim* the dimension (number of attributes) in the domain

*range\_dim* the dimension (number of attributes) in the range

*num\_hidden* the number of hidden neurons to use

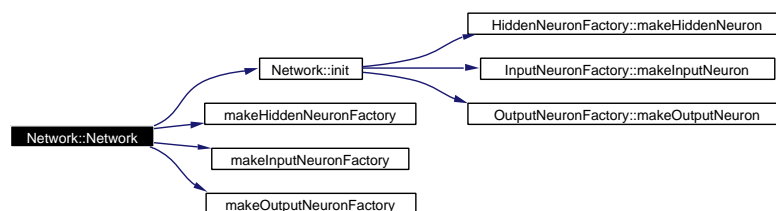
*tfunc* the default transfer function to use for neurons

*init\_func* the default initialisation function to use for neurons

Definition at line 21 of file neural.cc.

References `init()`, `INITIALISATION_FUNC`, `makeHiddenNeuronFactory()`, `makeInputNeuronFactory()`, `makeOutputNeuronFactory()`, and `TRANSFER_FUNC`.

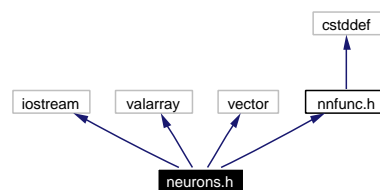
Here is the call graph for this function:



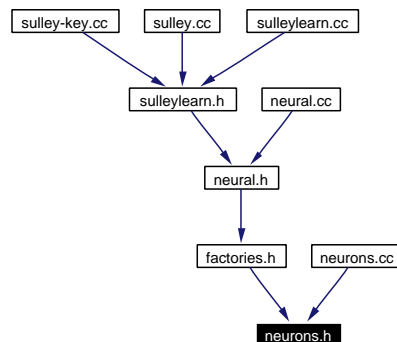
## 6.7 neurons.h File Reference

```
#include <iostream>
#include <valarray>
#include <vector>
#include "nnfunc.h"
```

Include dependency graph for neurons.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [BiasNeuron](#)  
*The simplest neuron - a bias "Neuron".*
- class [HiddenNeuron](#)  
*A hidden neuron.*
- class [InputNeuron](#)  
*An Input "Neuron" (in the 0-th layer).*
- class [LayerNeuron](#)  
*A neuron in a layer (other than layer 0).*

- class `Neuron`  
*A neuron in a neural network.*
- class `OutputNeuron`  
*An output neuron.*
- class `RBFNeuron`  
*A neuron using a radial basis function for 'transfer' and learning.*

## Enumerations

- enum `TRACE_OPTS` {  
`TO_NONE = 0, TO_HIDNEURON_UPDATE = 0x1, TO_HIDNEURON_FORWARD = 0x2,`  
`TO_OUTNEURON_UPDATE = 0x4,`  
`TO_OUTNEURON_FORWARD = 0x8, TO_HIDNEURON_BACKWARD = 0x10,`  
`TO_OUTNEURON_BACKWARD = 0x20, TO_NEURON_INIT = 0x40,`  
`TO_NETWORK_INIT = 0x1000, TO_NETWORK_FORWARD = 0x2000,`  
`TO_NETWORK_TARGET = 0x4000, TO_NETWORK_SET = 0x8000,`  
`TO_NETWORK_ERRORS = 0x10000, TO_NETWORK_ERROR = 0x20000,`  
`TO_CLIENT_LEARN = 0x40000000, TO_CLIENT_EST = 0x80000000,`  
`TO_ALL = 0xffffffff }`

## Functions

- `std::ostream & operator<<` (`std::ostream &o, const Neuron &n`)  
*Output n to o.*

## Variables

- bool `DO_MOMENTUM`
- double `MOMENTUM`
- double `LEARNING_RATE`
- int `TRACE_LEVEL`

## 6.7.1 Enumeration Type Documentation

### 6.7.1.1 enum `TRACE_OPTS`

#### Enumeration values:

*TO\_NONE*

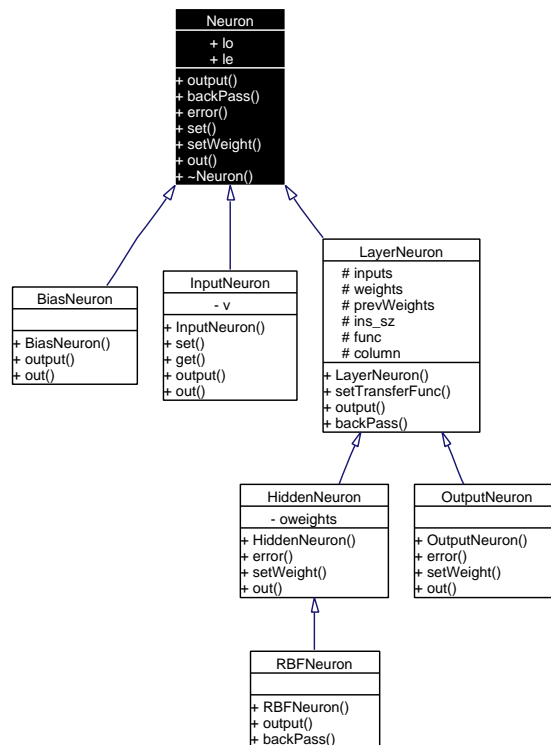
*TO\_HIDNEURON\_UPDATE*

*TO\_HIDNEURON\_FORWARD*

## 6.8 Neuron Class Reference

```
#include <neurons.h>
```

Inheritance diagram for Neuron:



### 6.8.1 Detailed Description

A neuron in a neural network.

Represents a neuron at any layer and for any type of neural network.

Definition at line 45 of file neurons.h.

#### Public Member Functions

- virtual double `output ()` const =0  
*Calculate output of this neuron, with its current inputs.*
- virtual void `backPass ()`  
*Calculate error and the next weights set.*
- virtual double `error (const std::valarray< double > &)`

*Return the error on a target; defaults to 0 error.*

- virtual void [set](#) (double val)  
*Override the output of this Neuron (or set an input neuron).*
- virtual void [setWeight](#) (int idx, double newWeight)  
*Commit the last calculated weights.*
- virtual std::ostream & [out](#) (std::ostream &o) const =0  
*Formatted output function.*
- virtual [~Neuron](#) ()  
*Virtual destructor – does nothing.*

## Public Attributes

- double [lo](#)  
*Last output (buffered for use in error calculation).*
- double [le](#)  
*Last "error" (actually the prospective change before momentum/learning rate are applied).*

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 virtual [Neuron::~~Neuron](#) () [inline, virtual]

Virtual destructor – does nothing.

Definition at line 83 of file neurons.h.

## 6.8.3 Member Function Documentation

### 6.8.3.1 virtual void [Neuron::backPass](#) () [inline, virtual]

Calculate error and the next weights set.

Not all layers will implement this - the default does nothing.

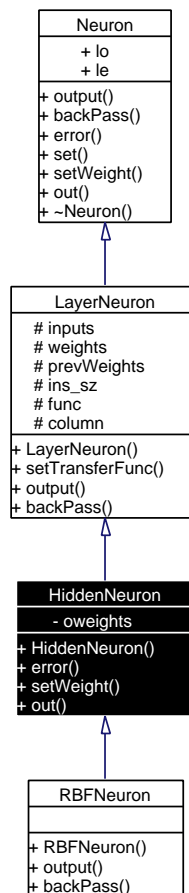
Reimplemented in [LayerNeuron](#), and [RBFNeuron](#).

Definition at line 61 of file neurons.h.

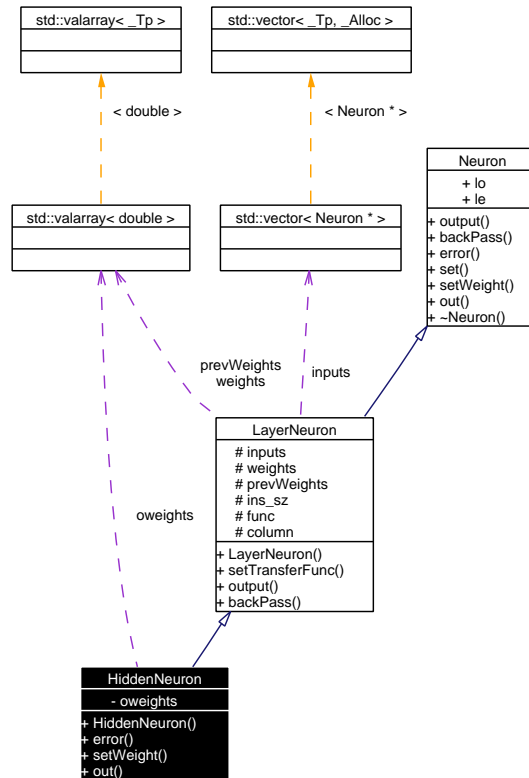
## 6.9 HiddenNeuron Class Reference

```
#include <neurons.h>
```

Inheritance diagram for HiddenNeuron:



Collaboration diagram for HiddenNeuron:



### 6.9.1 Detailed Description

A hidden neuron.

These have a more complicated learning rule based on backpropagation, gradient descent and the transfer function.

Definition at line 201 of file neurons.h.

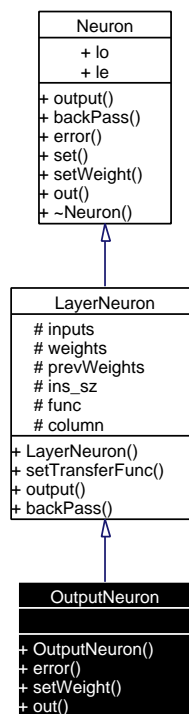
#### Public Member Functions

- [HiddenNeuron](#) ([std::vector< Neuron \\* >](#) \*ins, int col, int outputs, [TRANSFER\\_FUNC](#) f=&logsigmoidTF, [INITIALISATION\\_FUNC](#) init\_func=&rand\_init)  
*Create a hidden neuron.*
- double [error](#) (const [std::valarray< double >](#) &ierrors)  
*Calculate the error for the next weight update in [backPass\(\)](#).*
- void [setWeight](#) (int idx, double newWeight)  
*Apply the learning rule.*
- [std::ostream & out](#) ([std::ostream &o](#)) const  
*Formatted output function.*

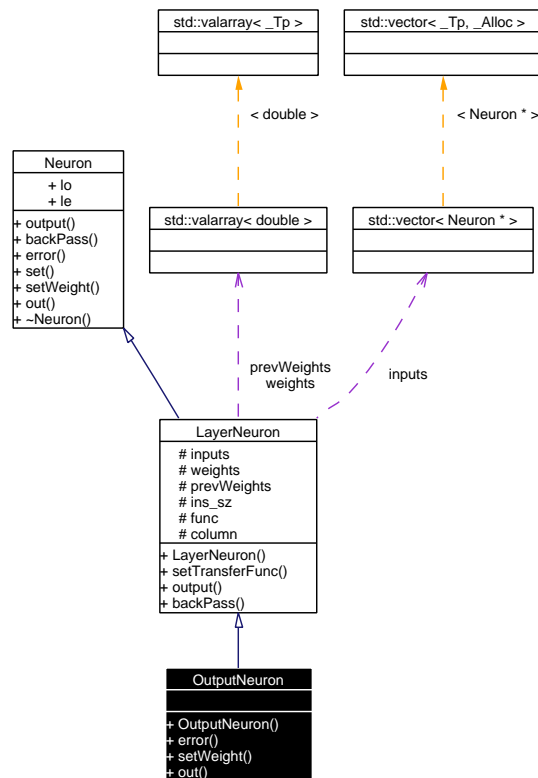
## 6.10 OutputNeuron Class Reference

```
#include <neurons.h>
```

Inheritance diagram for OutputNeuron:



Collaboration diagram for OutputNeuron:



### 6.10.1 Detailed Description

An output neuron.

These have a relatively simple learning rule, based only on the derivative of the transfer function.

Definition at line 171 of file neurons.h.

#### Public Member Functions

- **OutputNeuron** (**std::vector< Neuron \* >** \*ins, int col, **TRANSFER\_FUNC** f=logsigmoidTF, **INITIALISATION\_FUNC** init\_func=&rand\_init)

*Create an output neuron.*

- double **error** (double target)

*Calculate the error for the next weight update in **backPass()**.*

- void **setWeight** (int, double)

*For output neurons, the weight is updated in **backPass**.*

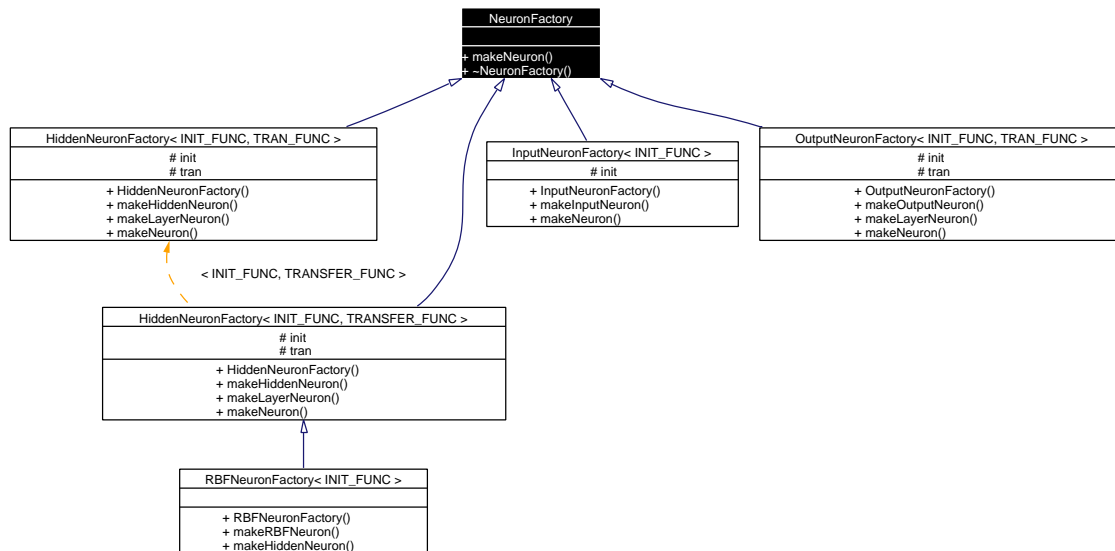
- **std::ostream & out** (**std::ostream &o**) const

*Formatted output function.*

## 6.11 NeuronFactory Class Reference

```
#include <factories.h>
```

Inheritance diagram for NeuronFactory:



### 6.11.1 Detailed Description

Superclass for all neuron factories.

A neuron factory allows [Neuron](#) specifics to be determined by the client of a learning algorithm by inheriting from the appropriate factory and constructing a subclass of [Neuron](#) to their desires

Definition at line 12 of file factories.h.

#### Public Member Functions

- virtual [Neuron](#) \* [makeNeuron](#) () const =0
- virtual [~NeuronFactory](#) ()

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 virtual NeuronFactory::~~NeuronFactory () [inline, virtual]

Definition at line 15 of file factories.h.

## 6.12 nnfunc.h File Reference

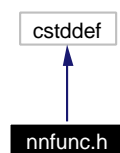
### 6.12.1 Detailed Description

Contains declarations for neural network transfer functions.

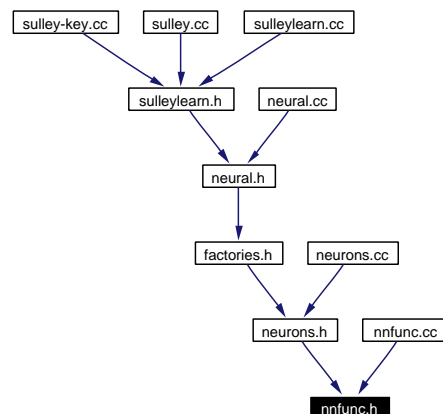
Definition in file [nnfunc.h](#).

```
#include <cstdint>
```

Include dependency graph for nnfunc.h:



This graph shows which files directly or indirectly include this file:



### Typedefs

- typedef double(\* [TRANSFER\\_FUNC](#))(double)  
*Type of a transfer function (for use in Neural Networks).*
- typedef double(\* [INITIALISATION\\_FUNC](#))(void)  
*Type of a neuron initialisation function.*

### Functions

- double [logsigmoidTF](#)(double n)

*Log-sigmoid transfer function:*  $f(n) = \frac{1}{1 + e^{-n}}$ .

- double [tansigmoidTF](#) (double n)

*Tan-sigmoid transfer function:*  $f(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}}$ .

- double [linearTF](#) (double n)

*Pure Linear transfer function:*  $f(n) = n$ .

- double [hardlimitTF](#) (double n)

*Hard Limit transfer function:*  $f(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$ .

- double [gaussianTF](#) (double n)

*Gaussian "transfer" function:*  $f(n) = e^{-\frac{d^2}{2\sigma}}$ .

- template<class ATTRSET> double [euclideanSq](#) (const ATTRSET &a, const ATTRSET &b)

*Euclidean distance function: returns the square of the euclidean distance between vectors a and b.*

- double [rand\\_init](#) (void)

*Random initialisation function.*

- double [zero\\_init](#) (void)

*Zero initialisation function.*

- std::size\_t [tozero\\_sum](#) (std::size\_t \*a)

*Return the sum of all elements in the array a until a 0 entry is reached.*

## Variables

- double [GAUSSIAN\\_TF\\_SIGMA](#)

*Default SIGMA to use for gaussian transfer function (RBF networks).*

## 6.12.2 Typedef Documentation

### 6.12.2.1 typedef double(\* [INITIALISATION\\_FUNC](#))(void)

Type of a neuron initialisation function.

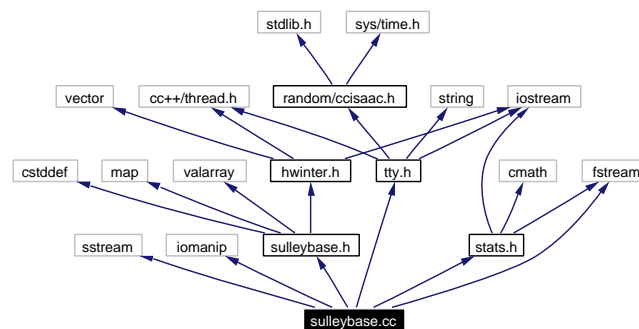
Definition at line 14 of file nfunc.h.

Referenced by `HiddenNeuron::HiddenNeuron()`, `LayeredNetwork::LayeredNetwork()`, `LayerNeuron::LayerNeuron()`, `Network::Network()`, `OutputNeuron::OutputNeuron()`, `RBFNetwork::RBFNetwork()`, and `RBFNeuron::RBFNeuron()`.

## 6.13 sulleybase.cc File Reference

```
#include <sstream>
#include <iomanip>
#include "sulleybase.h"
#include "tty.h"
#include "stats.h"
#include <fstream>
```

Include dependency graph for sulleybase.cc:



### Functions

- `std::ofstream logfile` ("sulleybase.log")
- `ostream & operator<<` (ostream &o, const ATTR\_SET &at)

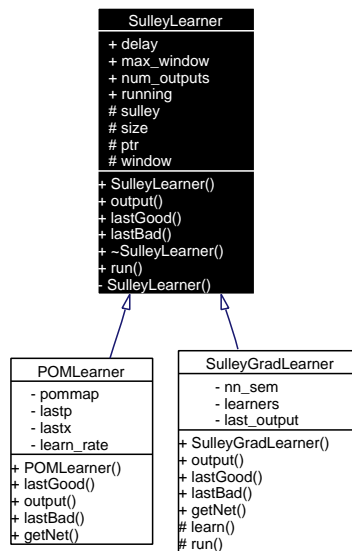
### Variables

- const `Sensor SENSOR` [MAX\_PIN+1]  
*The sensor setup in Sulley.*
- const int `JOIN_23` = 1  
*1 or 0, if 1 2/3 will be (2 || 3) (chest squeeze sensor)*
- const `size_t SULLEY_INPUTS` [] = {2, 3, 7, 8}  
*Sulley's inputs being used.*
- const `size_t NUM_SULLEY_INPUTS` = `sizeof(SULLEY_INPUTS) / sizeof(*SULLEY_INPUTS) - JOIN_23`  
*the number of items in SULLEY\_INPUTS*
- const double `HIGHVAL` = 1.0 `LOWVAL` = 0.0 `CHANGEVAL` = 0.5  
*values to use for learning from pins*

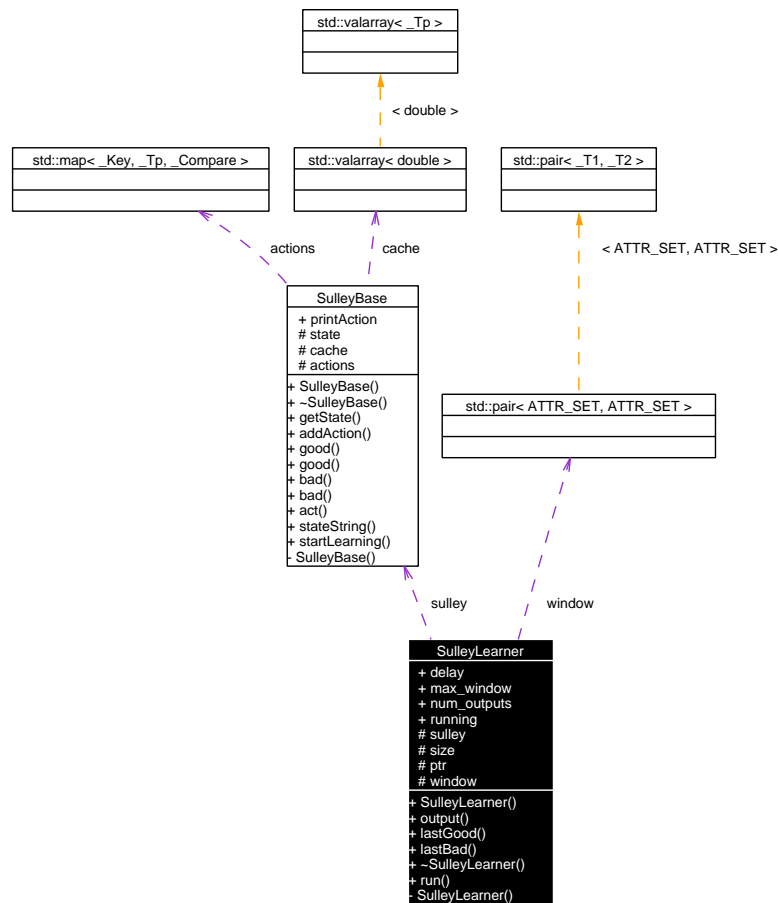
## 6.14 SulleyLearner Class Reference

```
#include <sulleybase.h>
```

Inheritance diagram for SulleyLearner:



Collaboration diagram for SulleyLearner:



### 6.14.1 Detailed Description

A superclass for learners using [SulleyBase](#) for their state.

Subclasses can utilise a learning window, so that recent learning examples can be re-used to assist learning and give smooth regression.

It is intended that subclasses use some classification technique customised to the way the state derived from the [Sulley](#) is used.

Definition at line 46 of file `sulleybase.h`.

#### Public Member Functions

- [SulleyLearner](#) ([SulleyBase](#) \*`sulley`, `size_t` `num_outputs`=1, `size_t` `max_window`=50, unsigned `delay`=1000000)  
*SulleyLearner constructor.*
- virtual int `output` (double \*`uncertainty`=0, double \*`confidence`=0, bool `save`=true)=0  
*Select an output from this learner.*
- virtual double `lastGood` ()=0

*The last.*

- virtual double `lastBad ()`=0
- virtual `~SulleyLearner ()`
- virtual void `run ()`

## Public Attributes

- size\_t `delay`  
*Delay (microseconds) between re-applying learning examples.*
- const size\_t `max_window`  
*The maximum window size.*
- const size\_t `num_outputs`  
*The number of outputs => the maximum value that output will return.*
- bool `running`  
*True if this is actively learning.*

## Protected Attributes

- `SulleyBase * sulley`  
*The source for the state from which to learn.*
- size\_t `size`  
*Number of cached learning instances (examples).*
- size\_t `ptr`  
*The index of the most recent learning instance.*
- `std::pair< ATTR_SET, ATTR_SET > * window`  
*window of input|output learning examples*

## 6.14.2 Constructor & Destructor Documentation

### 6.14.2.1 SulleyLearner::SulleyLearner (`SulleyBase * sulley`, `size_t num_outputs = 1`, `size_t max_window = 50`, `unsigned delay = 1000000`)

SulleyLearner constructor.

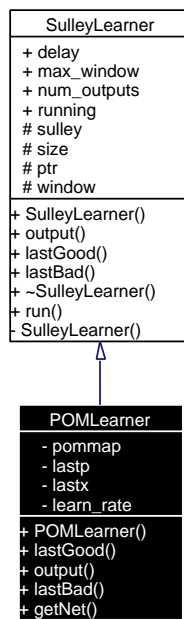
#### Parameters:

- sulley* a pointer to the `SulleyBase` for which this learner is learning
- num\_outputs* the number of outputs desired for this learner

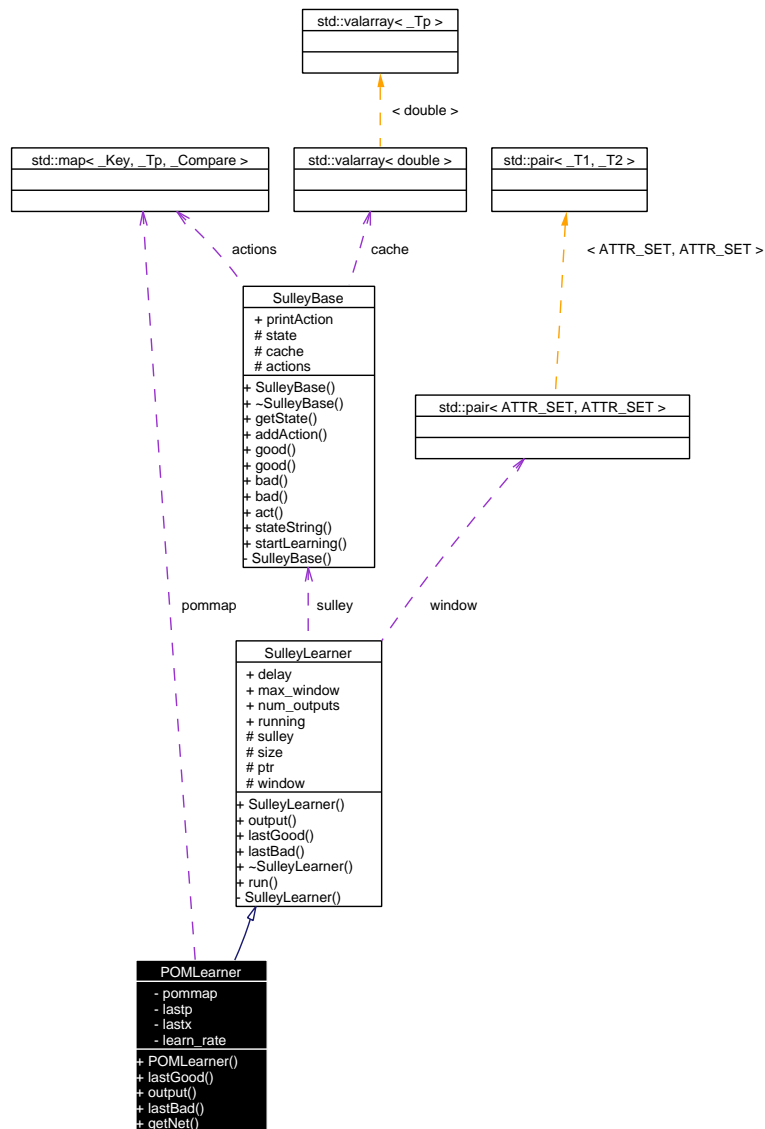
## 6.15 POMLearner Class Reference

```
#include <sulleylearn.h>
```

Inheritance diagram for POMLearner:



Collaboration diagram for POMLearner:



## Public Member Functions

- **POMLearner** (**SulleyBase** \*`sulley`, `size_t num_outputs`, `double learning_rate=0.3`, `size_t window_sz=0`)

- `double lastGood ()`

*The last.*

- `int output (double *uncertainty=0, double *confidence=0, bool save=true)`

*Select an output from this learner.*

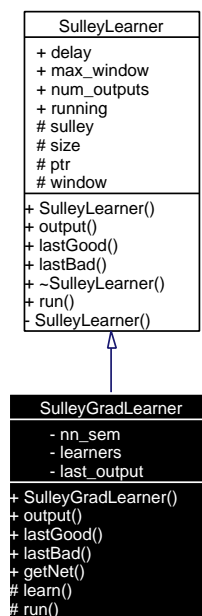
- `double lastBad ()`

- **SulNet** \* `getNet ()`

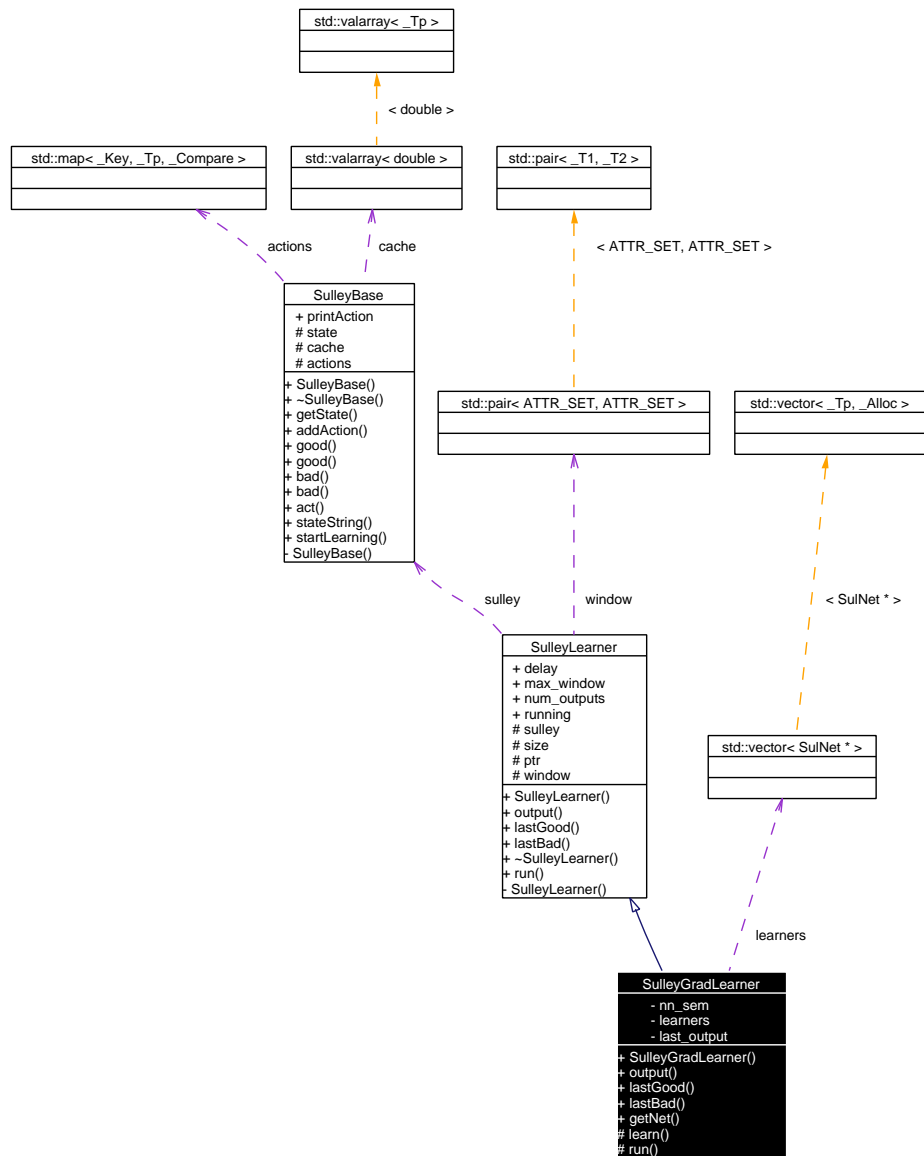
## 6.16 SulleyGradLearner Class Reference

```
#include <sulleylearn.h>
```

Inheritance diagram for SulleyGradLearner:



Collaboration diagram for SulleyGradLearner:



## Public Member Functions

- **SulleyGradLearner** (**SulleyBase** \***sulley**, `size_t num_outputs=1`, `size_t window_sz=50`)
- `int output` (`double *uncertainty=0`, `double *confidence=0`, `bool save=true`)

*Select an output from this learner.*

- `double lastGood` ()

*The last.*

- `double lastBad` ()
- `virtual SulNet * getNet` (`size_t idx=0`)