

ANN_toolbox

- ANN_toolbox
 - ANN — Toolbox for neural networks
 - ann_FF — Algorithms for feedforward nets.
 - ANN_GEN — General utility functions
 - ann_FF_ConjugGrad — Conjugate Gradient algorithm.
 - ann_FF_Hess — computes Hessian by finite differences.
 - ann_FF_INT — internal implementation of feedforward nets.
 - ann_FF_Jacobian — computes Jacobian by finite differences.
 - ann_FF_Jacobian_BP — computes Jacobian through backpropagation.
 - ann_FF_Mom_batch — batch backpropagation with momentum.
 - ann_FF_Mom_batch_nb — batch backpropagation with momentum (without bias).
 - ann_FF_Mom_online — online backpropagation with momentum.
 - ann_FF_Mom_online_nb — online backpropagation with momentum.
 - ann_FF_SSAB_batch — batch SuperSAB algorithm.
 - ann_FF_SSAB_batch_nb — batch SuperSAB algorithm (without bias).
 - ann_FF_SSAB_online — online SuperSAB training algorithm.
 - ann_FF_SSAB_online_nb — online backpropagation with SuperSAB
 - ann_FF_Std_batch — standard batch backpropagation.
 - ann_FF_Std_batch_nb — standard batch backpropagation (without bias).
 - ann_FF_Std_online — online standard backpropagation.
 - ann_FF_Std_online_nb — online standard backpropagation
 - ann_FF_VHess — multiplication between a "vector" V and Hessian
 - ann_FF_grad — error gradient through finite differences.
 - ann_FF_grad_BP — error gradient through backpropagation
 - ann_FF_grad_BP_nb — error gradient through backpropagation (without bias)
 - ann_FF_grad_nb — error gradient through finite differences
 - ann_FF_init — initialize the weight hypermatrix.
 - ann_FF_init_nb — initialize the weight hypermatrix (without bias).
 - ann_FF_run — run patterns through a feedforward net.
 - ann_FF_run_nb — run patterns through a feedforward net (without bias).
 - ann_d_log_activ — derivative of logistic activation function
 - ann_d_sum_of_sqr — derivative of sum-of-squares error
 - ann_log_activ — logistic activation function
 - ann_pat_shuffle — shuffles randomly patterns for an ANN
 - ann_sum_of_sqr — calculates sum-of-squares error

ANN

Toolbox for neural networks

Description

This ANN toolbox is designed to provide ANN engines for exploration and easy prototyping. This main page gives an overview.

The above main objective have influenced the overall design, i.e.

1. this toolbox will not be translated to C or Fortran for speed (see also the `sci2for` function),
2. except for the main algorithms (described in various books, e.g. see mine "Matrix ANN", it is heavily commented and explained such that it will be easy to adapt to particular needs if necessary,
3. at some points clarity was preferred over speed, when this do not lead to large speed loss.

Components

General support functions See `ANN_GEN` for support functions for ANN. Feedforward networks See `ann_FF` for detailed description.

See Also

- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.

ANN_GEN

General utility functions

Objective

To provide for some functions which are useful for a larger number of architectures. This is their general man page.

User interface: Parameters and functions

In alphabetical order, not all functions require all parameters:

```
x      Matrix containing input patterns, one per column. There shall be a
       unique column correspondence with the matrix of targets t and/or y.

y,z    Matrix containing actual output patterns, one per column. There shall
       be a unique column correspondence with the matrix of targets t and/or x.

t      Matrix containing target patterns, one per column. There shall be a
       unique column correspondence with the matrix of targets x and/or y.

y = ann_log_activ(x)
       Logistic activation function. Here y and x have to be column vectors.

z = ann_d_log_activ(y)
       Derivative of logistic activation function, expressed in terms of actual
       output (not in terms of total input !).

E = ann_sum_of_sqr(y,t)
       Sum-of-squares error function.

err_d = ann_d_sum_of_sqr(y,t)
       Derivative, with respect to network outputs of sum-of-squares error
       function. Here y and t have to be column vectors.

[x,t] = ann_pat_shuffle(x,t)
       Shuffles patterns randomly for better network training, the correspondence
       between x and t is preserved
```

See Also

- [ANN](#) — Toolbox for neural networks

ann_FF

Algorithms for feedforward nets.

Objective

To provide engines for feedforward ANN exploration, testing and rapid prototyping.

Some flexibility is provided (e.g. the possibility to change the activation or error functions).

Architecture description

- The network is visualized as follows: inputs at the left and data (signals) propagating to the right.
- N is a row vector containing the number of neurons per layer, input included.
- First layer is input (despite the fact that it does not process data it makes implementation clearer).

Layer no. 1 2 ... size(N,'c') . -- o o -> V \wedge i . -- o o -> o n \ | \ u p \ =====| > t u | / p t u s / t / s .
- o o -> input first output hidden

Note that connections do not jump over layers, they are only between adjacent layers (fully interconnected).

- The dimension of N is $\text{size}(N,'c')$ so:
 - input layer have $N(1)$ neurons
 - first hidden layer have $N(2)$ neurons, ...
 - the output layer L have $N(\text{size}(N,'c'))$ neurons
- The input vector/matrix is x , each pattern is represented by one column.

Only constant size input patterns are accepted.

NOTE: Internally the patterns will be worked with, individually, as column vectors, i.e. each pattern vector is a column of the form: $x(:,p)$, (p being the pattern order number).

- Each neuron on first hidden layer have $N(1)$ inputs, ... for layer l in $[2, \dots, \text{size}(N,'c')]$ each neuron have $N(l-1)$ inputs from previous layer plus one simulating the bias (where applicable, most algorithms assume existence of bias).
- The network is fully connected but a connection can be canceled by zeroing the corresponding weight

(note that a training procedure may turn it back to a non-zero value, this is one reason for which some "hooks" are provided, see "ex" parameter below).

User interface (1): Parameters

This subsection describes the parameters taken by the various functions defined within the toolbox, not all functions require all parameters.

In alphabetical order: af gives activation function and, if required, its (total) derivative. It is either:
- a string giving the name of activation function - a two element row vector of strings where af(1) is the string with the name of activation function and af(2) is the name of the derivative.

NOTE: Given an activation function $y = f(x)$, the derivative have to be expressed in terms of y not x).

E.g. given the logistic activation function: $y = \frac{1}{1 + \exp(-x)}$ the derivative will be expressed as: $dy/dx = y(1 - y)$ This form reduces the memory usage and, in the particular case of the logistic activation function, increases speed as well.

This parameter is optional, default value is either "ann_log_activ" or ["ann_log_activ","ann_d_log_activ"] (depending on the function using it), i.e. the activation function, described above, and its derivative (the default functions are already defined in this toolbox).

⚠ Be very careful how to define a new activation function.

These functions accept patterns as column vectors within y, each element representing the total input on a neuron, and return a similar matrix representing the activation of the whole layer. I.e. the logistic is defined as: $y = 1 ./ (1 + \exp(-x))$ and note the space between 1 and ./ The derivative of activation function is defined similarly: $z = y .* (1 - y)$ and note the ".*" operator.

Delta_W_old The quantity by which W was changed on previous training pattern.

dW, dW2 the amount of variations of each W element for calculating the error derivatives through a finite difference approach (see ann_FF_grad and ann_FF_Hess for more information). ef error function.

This parameter is optional, default value is "ann_sum_of_sqr", i.e. the sum-of-squares (already defined within this toolbox). err_deriv_y the error derivative with respect to network outputs. Returns a matrix each column containing the error derivative corresponding to the appropriate pattern.

This parameter is optional, default value is "ann_d_sum_of_sqr" (already defined within this toolbox), i.e. the derivative of sum-of-squares error function. ex is a Scilab program sequence, executed after the weight hypermatrix for each training pattern have been updated.

Its main purpose is to provide hooks in order to change the learning function without having to rewrite it. Typical usages would be: checking for a stop criteria, pruning.

This parameter is optional, default value is [" "] or [" "," "] (some functions have two hooks), i.e. empty string, does nothing. l range of layers between which the network is run.

Two component row vector: l(1) layer into which a pattern will be injected, presented as it would have come from previous layer: l(1)-1. l(2) layer from which the outputs are collected. E.g.: l = [3,3] means input is injected into neurons from layer 3 and their outputs (l(2)=3) are collected to give the result. l = [2,3] means input is injected into first hidden layer (exactly as it would have come from input layer) and output is collected at the outputs of neurons on layer 3. This parameter is optional, default value is [2,size(N,'c')] (whole network).

⚠ l(1) = 1 does not make sense as it represents the input layer.

lp represents the learning parameters, is a row vector [lp(1), lp(2), ...]

The actual significance of each component may vary, see the respective man pages for representation and typical values.

N row vector, defines the network, i.e. no. of neurons per layer. N(l) represents the number of neurons on layer l. E.g.: N(1) is the size of input vector, N(size(N),'c') is the size of output vector
r range of random numbers based on which the connection weights (not biases) are initialized.

Is a two component row vector: r(1) gives the lower limit r(2) gives the upper limit

This parameter is optional, default value is [-1,1].

rb range of random numbers based on which the biases (not other weights) are initialized.

Is a two component row vector: rb(1) gives the lower limit rb(2) gives the upper limit

This parameter is optional, default value is [0,0], i.e. biases are initialized with 0.

t matrix of targets, one pattern per column. E.g. t(:,p) represents pattern no. p. x matrix of inputs, one pattern per column. E.g. x(:,p) represents pattern no. p

User interface (2): Functions

The function names are built as follows:

ann	prefix for all function names within this toolbox.
_FF	prefix for all function names designed for FeedForward nets. Defines the type of algorithm: online uses one pattern at a time. batch uses all patterns at once.
_nb	postfix for all function names within this toolbox designed for networks without bias.
ann_FF_init	Build and initialize the weight hypermatrix.
ann_FF_Std	Standard (vanilla, delta rule) backpropagation algorithm.
ann_FF_Mom	Backpropagation with momentum.
ann_FF_run	Runs the network.
ann_FF_grad	Calculate the error gradient through a finite difference approach. It is provided for testing purposes only.
ann_FF_Jacobian	Calculate the Jacobian through a finite difference approach. It is provided for testing purposes only.
ann_FF_Jacobian_BP	Calculate the Jacobian through a Back-Propagation algorithm.
ann_FF_Hess	Calculate the Hessian through a finite difference approach. It is provided for testing purposes only.
ann_FF_VHess	Calculate the multiplication between a vector and the Hessian through an efficient finite difference approach.
ann_FF_ConjugGrad	Conjugate gradients algorithm.
ann_FF_SSAB	Backpropagation with SuperSAB algorithm.

Tips and tricks

- Do not use the no-bias networks unless you know what you are doing.
- The most efficient (by far) algorithm is the "Conjugate Gradient", however it may require bootstrapping with another algorithm (see the examples).
- Reduce as much as possible the number of loops and the number of function calls, use instead as much as possible the matrix manipulation capabilities of Scilab.
- You can do a shuffling of training patterns between two calls to the training procedure, use the "ex" hooks provided.
- Be very careful when defining new activation and error functions and test them to make sure they do what are supposed to do.
- don't use sparse matrices unless they are really sparse (<5%).

Implementation details

- Each layer have associated a hypermatrix of weights.

💡 Most algorithms assume existence of bias by default. For each layer l , except $l=1$, the weight matrix associated with connections from $l-1$ to l is $W(1:N(l),1:N(l-1),l-1)$ for networks without biases and $W(1:N(l),1:N(l-1)+1,l-1)$ for networks with biases, i.e. biases are stored in first column: $W(1:N(l),1,l-1)$.

The total input to a layer l is: $= W(1:N(l),1:N(l-1),l-1)*z(1:N(l-1))$ for network without biases $= W(1:N(l),1:N(l-1)+1,l-1)*[1; z(1:N(l-1))]$ for network with biases where $z(1:N(l-1))$ is output of previous layer (column vector), i.e. bias is simulated as neuron no. 0 on each layer with constant output 1.

W is initialized to $\text{zeros}(\max(N),\max(N),\text{size}(N,'c')-1)$ for networks without biases, and to $\text{hypermat}(\max(N),\max(N)+1,\text{size}(N,'c')-1)$ for networks with biases; the unused entries from W are initialized to zero and left untouched.

- Pattern vectors are passed as columns in a matrix representing a set (of patterns).

Objective

- No sanity checks are performed as this will greatly hurt speed. It is assumed that (as least to some extent) you know what you are doing ;-) You can implement them yourself if you wish.

The following conditions have to be met: + targets have to have the same size as output layer, i.e. $\text{size}(\text{target},'r') = N(\text{size}(N,'c'))$ + inputs have to have the same size as input layer, i.e. $\text{size}(\text{input},'r') = N(1) + \text{all } N(i)$ have to be positive integers of course (am I paranoid here ? :-) + lp parameter is a row-vector of numbers (actual dimension depends on the algorithm used). + af is a row-vector of string(s) defining a valid activation function (and its derivative) as appropriate for the algorithm used. + ex is a string or a two row-vector of strings, representing valid Scilab set of instructions. + $l(1) \leq l(2)$ (see definition of l above) and $l(1) \geq 2 + r(1) \leq r(2)$ and $rb(1) \leq rb(2)$ (see definition of r and rb above), if not then the program may run but you may not get what you would expect when initializing the weights. Warning: In some particular cases this may lead to very subtle errors (e.g. your program may even run without generating any Scilab errors but the results may be meaningless).

- The algorithms themselves are not described here, there are many books which describes them

(e.g. get mine "Matrix ANN" wherever you may find it ;-). - Hypermatrices are slow, however there is no other reasonable way of doing things; tests performed by myself show that using embedded matrices may increase speed but the manipulation of submatrices "by hand" is very tedious and error prone. Of course you may rewrite the algorithms for yourself using embedded matrices if you want to. If you really need speed then go directly to C++ or whatever.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions

ann_FF_ConjugGrad

Conjugate Gradient algorithm.

Syntax

```
W = ann_BP_ConjugGrad(x, t, N, W, T, dW [, ex, af, err_deriv_y])
```

Arguments

- W**
The weight hypermatrix; it have to be initialized with `ann_BP_init` function.
- x**
Matrix of input patterns, one pattern per column.
- t**
Matrix of target patterns, one pattern per column.
- N**
Row vector describing the number of neurons per layer.

N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).
- T**
Number of training cycles (epochs, discrete time, steps).
- dW**
The quantity used to calculate the product between a direction and the Hessian trough a finite differences algorithm. This parameter is passed to `ann_BP_VHess` function (see its man page for details).
- options**
 - ex**
string representing a valid Scilab sequence. It is executed after the weight hypermatrix have been updated (i.e. after each epoch), using "execstr". This parameter is optional, default value: "" (empty string, i.e. does nothing).
 - af**
The name of activation function to be used (string). This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.
 - err_deriv_y**
The name of error function derivative with respect to network outputs (string). This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.

Description

This function performs training of a feedforward net using the conjugate gradients algorithm.

The computation of Hessian is avoided by calculating directly the product between a direction and Hessian trough a finite difference approach which require just two error gradients (see function `ann_BP_VHess`).

The error gradient is calculated across all training patterns at once given in x (respectively t).

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init](#) — initialize the weight hypermatrix.
- [ann_FF_run](#) — run patterns through a feedforward net.
- [ann_FF_grad_BP](#) — error gradient through backpropagation
- [ann_FF_VHess](#) — multiplication between a "vector" V and Hessian

ann_FF_init

initialize the weight hypermatrix.

Syntax

```
W = ann_FF_init(N [, r, rb])
```

Arguments

N

Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).

r

Two component row vector defining the smallest and the largest value for initialization. Weights will be initialized with random numbers between these two values.

r(1) the lower limit

r(2) the upper limit

This parameter is optional, default value is [-1,1].

rb

Same as r but for biases. This parameter is optional, default value is [0,0], i.e. the biases are initialized to zero.

W

The weight hypermatrix, in the format used by ann_FF_Std, ann_BP_run and all other algorithms for feedforward nets (with biases).

Description

This function builds the weight hypermatrix according to network description N. Its format is detailed in ann_FF.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_run

run patterns through a feedforward net.

Syntax

```
y = ann_FF_run(x, N, W [, l, af])
```

Arguments

- y** Matrix of outputs, one pattern per column. Each column have a correspondent column in x.
- x** Matrix of input patterns, one pattern per column.
- N** Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector.
- W** The weight hypermatrix (initialized first through ann_FF_init).
- l** Defines the "injection" layer and the output layer. x patterns are injected into layer l(1) as coming from layer l(1) - 1. y outputs are collected from the outputs of layer l(2).
This parameter is optional, default value is [2,size(N,'c')], i.e. the whole network.
⚠ l(1)=1 does not make sense.
- af** String containing the name of activation function. This parameter is optional, default value is "ann_log_activ", i.e. logistic activation function.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_grad_BP

error gradient through backpropagation

Syntax

```
W = ann_FF_grad_BP(x, t, N, W [, c, af, err_deriv_y])
```

Arguments

- W** The weight hypermatrix (initialized first through `ann_BP_init`).
- x** Matrix of input patterns, one pattern per column.
- t** Matrix of targets, one pattern per column. Each column have a correspondent column in `x`.
- N** N Row vector describing the number of neurons per layer. `N(1)` is the size of input pattern vector, `N(size(N,'c'))` is the size of output pattern vector (and also target).
- c** defines the threshold of error which is backpropagated: a error smaller that `c` (at one neuronal output) is rounded towards zero and thus not propagated. It have to be set to zero for exact calculation of gradient. This parameter is optional, default value 0.
- af** Activation function and its derivative. Row vector of strings: `af(1)` name of activation function. `af(2)` name of derivative.
⚠ Given the activation function $y=f(x)$, the derivative have to be expressed in terms of `y`, not `x`. This parameter is optional, default value is `['ann_log_activ', 'ann_d_log_activ']`, i.e. logistic activation function and its derivative.
- err_deriv_y** the name of error function derivative with respect to network outputs. This parameter is optional, default value is `"ann_d_sum_of_sqr"`, i.e. the derivative of sum-of-squares.

Description

Returns the error gradient hypermatrix (it have the same layout as `W`) of a feedforward ANN, using the whole given training set. The algorithm used is standard backpropagation.

Examples

This function is used as a low level engine in other algorithms (thus the reason for existence of `c` parameter), e.g. see implementation of `ann_FF_ConjugGrad` function.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init](#) — initialize the weight hypermatrix.

ann_FF_VHess

multiplication between a "vector" V and Hessian

Syntax

```
VH = ann_BP_VHess(x, t, N, W, V, dW, [af, err_deriv_y])
```

Arguments

- VH** The result of multiplication - hypermatrix with the same layout as W.
- x** Matrix of input patterns, one pattern per column.
- t** Matrix of target patterns, one pattern per column.
- N** Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).
- W** The weight hypermatrix.
- V** The "vector" by which Hessian have to be multiplied, actually is a hypermatrix with same layout as W (is from same space).
- dW** Size of "finite difference".
- af** The activation function to be used. This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.
- err_deriv_y** the name of error function derivative with respect to network outputs. This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.

Description

This function calculates the product between a vector and Hessian through a (fast) finite difference procedure. The error gradient is calculated (through backpropagation) at $W+dW*V$ and at $W-dW*V$ and then VH is calculated directly from here (this requires only two backpropagations, explicit computation of Hessian is avoided).

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_Hess

computes Hessian by finite differences.

Syntax

```
H = ann_FF_Hess(x, t, N, W, dW, dW2 [,af ,ef])
```

Arguments

- H**
The Hessian hypermatrix, have the same layout as $W \cdot W$ (not $W \cdot W'$, as usually found in literature); $H(n1,i1,l1,n2,i2,l2)$ is the second derivative of error with respect to weights $W(n1,i1,l1)$ and $W(n2,i2,l2)$.
- x**
Matrix of input patterns, one pattern per column.
- t**
Matrix of target patterns, one pattern per column.
- N**
Row vector describing the number of neurons per layer. $N(1)$ is the size of input pattern vector, $N(\text{size}(N,'c'))$ is the size of output pattern vector (and also target).
- W**
The weight hypermatrix.
- dW, dW2**
The quantities used to perturb each W parameter; dW is used for non-diagonal parameters, for diagonal parameters the supplemental perturbations quantities $dW2 \cdot dW$ are also used (note that very small values may easily lead to numerical instabilities).
- af**
The activation function to be used. This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.
- ef**
The error function to be used. This parameter is optional, default value "ann_sum_of_sqr", i.e. the sum-of-squares error function.

Description

This function calculates the Hessian through a finite differences procedure. This process is very slow and is provided for testing purposes only. Internals: for off-diagonal elements: two weights are perturbed at $\pm dW$; for on-diagonal elements the perturbing quantities are: $\pm (1 \pm dW2) \cdot dW$ for one weight; the corresponding Hessian term is calculated from the four resulting errors (at four W points).

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_INT

internal implementation of feedforward nets.

Description

This man page describes the internals of implementation, it is of interest only to those wanting to modify/adapt the functions provided (in case they do not fit the need :-)

Internal variables (in alphabetical order)

The variables described here are used internally only in the functions body. They are:

```
err_dz
    the partial derivative of error with respect to neuronal outputs, on current layer
err_dz_deriv_af
    the element-wise product between err_dz and deriv_af.

deriv_af
    the derivative of activation function for current layer.

grad_E
    gradient of error, same hypermatrix structure as W. E.g. grad_E(n,i,l)
    destined to hold the partial derivative of error with respect to W(n,i,l).
grad_E_mod
    similar to grad_E but it is not the real grad_E because is modified in
    some significant way.
l, ll
    current layer number (different from l(1), l(2)) (if l=[l(1),l(2)] is
    used as parameter then ll is used as layer counter)

L
    total number of layers, including input.

p
    current pattern number.

P
    total number of patterns.

y
    holds the network output(s), one per column.

z
    matrix of neuronal outputs, one column for each layer, e.g.
z(1:N(1),1)
    contains the inputs
z(1:N(2),2)
    contains the outputs of first hidden layer
z(1:N(L),L)
    contains the output

BIAS
    Bias is simulated trough an additional neuron with constant output "1".
    The "bias neuron" is present on all layers, except output, as the first
    neuron.
```

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_Jacobian

computes Jacobian by finite differences.

Syntax

```
J = ann_FF_Jacobian(x, N, W, dx [, af])
```

Arguments

- J**
The Jacobian hypermatrix: each $J(:, :, p)$ have same structure as $z(:, p) * x(:, p)'$, where $z(:, p)$ is the network output given input $x(:, p)$.
- x**
Matrix of input patterns, one pattern per column.
- N**
Row vector describing the number of neurons per layer. $N(1)$ is the size of input pattern vector, $N(\text{size}(N, 'c'))$ is the size of output pattern vector (and also target).
- W**
The weight hypermatrix.
- dx**
The quantity used to perturb each $x(i, p)$ in turn.
- af**
The activation function to be used. This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.

Description

This function calculates the Jacobian through a finite differences procedure, for all patterns presented in x .

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_Jacobian_BP

computes Jacobian through backpropagation.

Syntax

```
J = ann_FF_Jacobian_BP(x, N, W)
J = ann_FF_Jacobian_BP(x, N, W, af)
```

Arguments

- J**
The Jacobian hypermatrix: each $J(:, :, p)$ have same structure as $z(:, p) * x(:, p)'$, where $z(:, p)$ is the network output given input $x(:, p)$.
- x**
Matrix of input patterns, one pattern per column.
- N**
Row vector describing the number of neurons per layer. $N(1)$ is the size of input pattern vector, $N(\text{size}(N, 'c'))$ is the size of output pattern vector (and also target).
- W**
The weight hypermatrix.
- af**
The activation function to be used. This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.

Description

This function calculates the Jacobian through a backpropagation algorithm, for all patterns presented in x.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.

[ANN_toolbox](#) > `ann_FF_Mom_batch`

ann_FF_Mom_batch

batch backpropagation with momentum.

Syntax

```
[W, Delta_W_old] = ..  
ann_FF_Mom_batch(x, t, N, W, lp, T [, Delta_W_old, af, ex, err_deriv_y])
```

Arguments

This function have same parameters as `ann_FF_Mom_online` except for:

ex

string representing a valid Scilab program sequence, executed after each epoch trough `execstr`.

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is batch backpropagation with momentum.

See Also

- [ann_FF_Mom_online](#) — online backpropagation with momentum.

ann_FF_Mom_online

online backpropagation with momentum.

Syntax

```
[W, Delta_W_old] = ann_FF_Mom_online(x, t, N, W, lp, T [, Delta_W_old, af, ex, err_deriv_
```

Arguments

x

Matrix of input patterns, one pattern per column.

t

Matrix of targets, one pattern per column. Each column have a correspondent column in x.

N

Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).

W

The weight hypermatrix (initialized first with ann_FF_init_nb).

lp

Learning parameters [lp(1),lp(2),lp(3),lp(4)].

is the well known learning parameter of standard backpropagation algorithm, W is changed according to the formula:

lp(1) $W(t+1) = W(t) - lp(1) * grad E + momentum term$

where t is the (discrete) time and E is the error. Typical values: 0.1 ... 1. Some networks train faster with even greater learning parameter.

defines the threshold of error which is backpropagated: an error smaller than lp(2) (at one neuronal output) is rounded towards zero and thus not propagated. Typical values: 0 ... 0.1. E.g. assume that output of neuron n have the actual output 0.91 and the target (for that particular neuron, given the corresponding input) is 1. If lp(2) = 0.1 then the error term associated to n is rounded to 0 and thus not propagated.

lp(2)

is the momentum parameter. The momentum term added to W is: momentum term = lp(3) * Delta_W_old.

lp(3)

Typical values: 0 ... 0.9999... (smaller than 1).

is the flat spot elimination constant added to the derivative of activation function, when computing the error gradient, to help the network pass faster over areas where the error gradient is small (flat error surface area). I.e. the derivative of activation is

lp(4)

replaced:

$f'(total\ neuronal\ input) \rightarrow f'(total\ neuronal\ input) + lp(4)$

when computing the gradient. Typical values: 0 ... 0.25.

T

the number of epochs (training cycles trough all pattern set).

Delta_W_old

The previous weight adjusting quantity. This parameter is optional, default value is: Delta_W_old = zeros(W).

NOTE: When calling `ann_FF_Mom_online()` for the first time you should either:

- not give any value to `Delta_W_old`
- initialize it with `Delta_W_old = zeros(W)` On subsequent calls to `ann_FF_Mom_online()`, you should give the value of `Delta_W_old` returned by the previous call.

af

Activation function and its derivative. Row vector of strings:

af(1) name of activation function.

af(2) name of derivative.

⚠ Given the activation function $y = f(x)$, the derivative have to be expressed in terms of y , not x . This parameter is optional, default value is `['ann_log_activ', 'ann_d_log_activ']`, i.e. logistic activation function and its derivative.

err_deriv_y

the name of error function derivative with respect to network outputs. This parameter is optional, default value is `"ann_d_sum_of_sqr"`, i.e. the derivative of sum-of-squares.

ex

two-dimensional row vector of strings representing valid Scilab sequences.

`ex(1)` is executed after the weight hypermatrix have been updated, after each pattern (not whole set), using `execstr`.

`ex(2)`, same as `ex(1)`, but is executed after each epoch. This parameter is optional, default value is `[" "," "]` (do nothing).

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is online backpropagation with momentum. `Delta_W_old` holds the previous W update (usefull for subsequent calls).

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init](#) — initialize the weight hypermatrix.
- [ann_FF_run](#) — run patterns trough a feedforward net.

[ANN_toolbox](#) > `ann_FF_Mom_batch_nb`

ann_FF_Mom_batch_nb

batch backpropagation with momentum (without bias).

Syntax

```
[W, Delta_W_old] = ann_FF_Mom_batch_nb(x, t, N, W, lp, T [, Delta_W_old, af, ex, err_der:
```

Arguments

ex

string representing a valid Scilab program sequence, executed after each epoch trough `execstr`.

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is batch backpropagation with momentum. This function is to be used on networks without biases.

See Also

- [ann_FF_Mom_online_nb](#) — online backpropagation with momentum.

ann_FF_Mom_online_nb

online backpropagation with momentum.

Syntax

```
[W, Delta_W_old] = ann_FF_Mom_online_nb(x, t, N, W, lp, T [, Delta_W_old, af, ex, err_der
```

Arguments

x

Matrix of input patterns, one pattern per column.

t

Matrix of targets, one pattern per column. Each column have a correspondent column in x.

N

Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector. N(size(N,'c')) is the size of output pattern vector (and also target).

W

The weight hypermatrix (initialized first with ann_FF_init_nb).

lp

Learning parameters [lp(1),lp(2),lp(3),lp(4)].

lp(1) is the well known learning parameter of standard backpropagation algorithm, W is changed according to the formula:

$$\mathbf{lp(1)} \quad W(t+1) = W(t) - \mathbf{lp(1)} * \text{grad } E + \text{momentum term}$$

where t is the (discrete) time and E is the error. Typical values: 0.1 ... 1. Some networks train faster with even greater learning parameter.

lp(2) defines the threshold of error which is backpropagated: a error smaller that lp(2) (at one neuronal output) is rounded towards zero and thus not propagated. Typical values: 0 ... 0.1. E.g. assume that output of neuron n have the actual output 0.91 and the target (for that particular neuron, given the corresponding input) is 1. If lp(2) = 0.1 then the error term associated to n is rounded to 0 and thus not propagated.

lp(3) is the momentum parameter. The momentum term added to W is: momentum term = lp(3) * Delta_W_old.

Typical values: 0 ... 0.9999... (smaller than 1).

lp(4) is the flat spot elimination constant added to the derivative of activation function, when computing the error gradient, to help the network pass faster over areas where the error gradient is small (flat error surface area). I.e. the derivative of activation is replaced:

$$f'(\text{total neuronal input}) \rightarrow f'(\text{total neuronal input}) + \mathbf{lp(4)}$$

when computing the gradient. Typical values: 0 ... 0.25.

T

the number of epochs (training cycles trough all pattern set).

Delta_W_old

The previous weight adjusting quantity. This parameter is optional, default value is: Delta_W_old = zeros(W).

NOTE: When calling `ann_FF_Mom_online_nb` for the first time you should either:

- not give any value to `Delta_W_old`
- initialize it with `Delta_W_old = zeros(W)` On subsequent calls to `ann_FF_Mom_online_nb` you should give the value of `Delta_W_old` returned by the previous call.

af

Activation function and its derivative. Row vector of strings:

af(1) name of activation function.

af(2) name of derivative.

⚠ Given the activation function $y = f(x)$, the derivative have to be expressed in terms of y , not x . This parameter is optional, default value is `"['ann_log_activ', 'ann_d_log_activ']"`, i.e. logistic activation function and its derivative.

err_deriv_y

the name of error function derivative with respect to network outputs. This parameter is optional, default value is `"ann_d_sum_of_sqr"`, i.e. the derivative of sum-of-squares.

ex

two-dimensional row vector of strings representing valid Scilab sequences. `ex(1)` is executed after the weight hypermatrix have been updated, after each pattern (not whole set), using `execstr`. `ex(2)`, same as `ex(1)`, but is executed after each epoch. This parameter is optional, default value is `[" "," "]` (do nothing).

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns. The algorithm used is online backpropagation with momentum. `Delta_W_old` holds the previous weight update (useful for subsequent calls). This function is to be used on networks without biases.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init_nb](#) — initialize the weight hypermatrix (without bias).
- [ann_FF_run_nb](#) — run patterns trough a feedforward net (without bias).

ann_FF_init_nb

initialize the weight hypermatrix (without bias).

Syntax

```
W = ann_BP_init_nb(N)
W = ann_BP_init_nb(N, r)
```

Arguments

N
Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).

r
Two component row vector defining the smallest and the largest value for initialization. Weights will be initialized with random numbers between these two values.

r(1) the lower limit

r(2) the upper limit

This parameter is optional, default value is [-1,1].

W
The weight hypermatrix, in the format used by ann_BP_Std_nb, ann_BP_run_nb and other functions working with feedforward nets (without bias).

Description

This function builds the weight hypermatrix according to network description N. The format of it is detailed in ann_FF. This function is to be used on networks without biases.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_run_nb

run patterns through a feedforward net (without bias).

Syntax

```
y = ann_FF_run_nb(x, N, W [, l, af])
```

Arguments

- y** Matrix of outputs, one pattern per column. Each column has a correspondent column in x.
- x** Matrix of input patterns, one pattern per column.
- N** Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector.
- W** The weight hypermatrix (initialized first through ann_BP_init_nb).
- l** Defines the "injection" layer and the output layer. x patterns are injected into layer l(1) as coming from layer l(1) - 1. y outputs are collected from the outputs of layer l(2). This parameter is optional, default value is [2,size(N,'c')], i.e. the whole network.
⚠ l(1)=1 is meaningless.
- af** String containing the name of activation function. This parameter is optional. Default value "ann_log_activ", i.e. logistic activation function.

Description

This function is used to run patterns through a feedforward network as defined by N and W. This function is to be used on networks without biases.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

[ANN_toolbox](#) > `ann_FF_SSAB_batch`

ann_FF_SSAB_batch

batch SuperSAB algorithm.

Syntax

```
[W, Delta_W_old, Delta_W_oldold, mu] = ..  
ann_FF_SSAB_batch(x, t, N, W, lp, Delta_W_old, Delta_W_oldold, T [, mu, af, ex, err_deriv])
```

Arguments

This function have same parameters as `ann_FF_SSAB_online` except for:

ex

string representing a valid Scilab program sequence, executed after each epoch trough `execstr`.

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is batch SuperSAB.

See Also

- [ann_FF_SSAB_online](#) — online SuperSAB training algorithm.

ann_FF_SSAB_online

online SuperSAB training algorithm.

Syntax

```
[W, Delta_W_old, Delta_W_oldold, mu] = ..  
ann_FF_SSAB_online(x, t, N, W, lp, Delta_W_old, Delta_W_oldold, T [, mu, af, ex, err_der:
```

Arguments

- x** Matrix of input patterns, one pattern per column.
- t** Matrix of targets, one pattern per column. Each column have a correspondent column in x.
- N** Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).
- W** The weight hypermatrix (initialized first trough ann_FF_init).
- lp** Learning parameters [lp(1),lp(2),lp(3),lp(4),lp(5)].
 - lp(1)** is the well known learning parameter of vanilla backpropagation algorithm, it is used only to initialize mu if not given (mu being optional) Typical values: 0.1 ... 1. Note that initial values may be strongly amplified in some circumstances.
 - lp(2)** defines the threshold of error which is backpropagated: a error smaller that lp(2) (at one neuronal output) is rounded towards zero and thus not propagated. Typical values: 0 ... 0.1. E.g. assume that output of neuron n have the actual output 0.91 and the target (for that particular neuron, given the corresponding input) is 1. If lp(2) = 0.1 then the error term associated to n is rounded to 0 and thus not propagated.
 - lp(3)** is the momentum parameter, used to "cancel" a previously wrong weight adaptation. Typical values: 0 ... 0.9999... (smaller than 1).
 - lp(4)** is the adaptive increasing factor. Typical values: 1.1 ... 1.3.
 - lp(5)** is the adaptive decreasing factor. Typical values: lp(5) = 1/lp(4) (lp(5) < 1).
- Delta_W_old** The previous weight adjusting quantity. On first call this parameter should be initialized to zero with `Delta_W_old = zeros(w)`. On subsequent calls to `ann_FF_SSAB` you should give the value of `Delta_W_old` returned by the previous call.
- Delta_W_oldold** The weight adjusting quantity used two steps back. Same remarks as for `Delta_W_old` (above) apply.
- T** The number of epochs (training cycles trough all pattern set).
- mu** This is the hypermatrix of learning constants who replaces lp(1) and is adapted at each step. This parameter is optional, default value is `mu = lp(1)*ones(w)`, i.e. it is initialized uniformly to lp(1).
- af**

Activation function and its derivative. Row vector of strings:

af(1) name of activation function (string).

af(2) name of derivative.

⚠ Given the activation function $y=f(x)$, the derivative have to be expressed in terms of y , not x . This parameter is optional, default value is ['ann_log_activ', 'ann_d_log_activ'], i.e. logistic activation function and its derivative.

err_deriv_y

The name of error function derivative with respect to network outputs. This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.

ex

two-dimensional row vector of strings representing valid Scilab sequences. ex(1) is executed after the weight matrix have been updated, after each pattern (not whole set), using execstr.

ex(2) - same as ex(1) - but is executed once after each epoch. This parameter is optional, default value is [" "," "] (do nothing).

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns. The algorithm used is online backpropagation with SuperSAB.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init](#) — initialize the weight hypermatrix.
- [ann_FF_run](#) — run patterns trough a feedforward net.

[ANN_toolbox](#) > `ann_FF_SSAB_batch_nb`

ann_FF_SSAB_batch_nb

batch SuperSAB algorithm (without bias).

Syntax

```
[W, Delta_W_old, Delta_W_oldold, mu] = ..  
ann_FF_SSAB_batch_nb(x, t, N, W, lp, Delta_W_old, Delta_W_oldold, T [, mu, af, ex, err_de
```

Arguments

This function have same parameters as `ann_FF_SSAB_online_nb` except for:

ex

string representing a valid Scilab program sequence, executed after each epoch trough `execstr`.

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is batch SuperSAB.

See Also

- [ann_FF_SSAB_online_nb](#) — online backpropagation with SuperSAB

ann_FF_SSAB_online_nb

online backpropagation with SuperSAB

Syntax

```
[W, Delta_W_old, Delta_W_oldold, mu] = ..  
ann_BP_SSAB(x, t, N, W, lp, Delta_W_old, Delta_W_oldold, T [, mu, af, ex, err_deriv_y])
```

Arguments

- x**
Matrix of input patterns, one pattern per column.
- t**
Matrix of targets, one pattern per column. Each row have a correspondent column in x.
- N**
Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).
- W**
The weight hypermatrix (initialized first trough ann_FF_init_nb).
- lp**
Learning parameters [lp(1),lp(2),lp(3),lp(4),lp(5)].
 - lp(1)** is the well known learning parameter of vanilla backpropagation algorithm, it is used only to initialize mu if not given (mu being optional) Typical values: 0.1 ... 1. Note that initial values may be strongly amplified in some circumstances.
 - lp(2)** defines the threshold of error which is backpropagated: a error smaller that lp(2) (at one neuronal output) is rounded towards zero and thus not propagated. Typical values: 0 ... 0.1. E.g. assume that output of neuron n have the actual output 0.91 and the target (for that particular neuron, given the corresponding input) is 1. If lp(2) = 0.1 then the error term associated to n is rounded to 0 and thus not propagated.
 - lp(3)** is the momentum parameter, used to "cancel" a previously wrong weight adaptation. Typical values: 0 ... 0.9999... (smaller than 1).
 - lp(4)** is the adaptive increasing factor. Typical values: 1.1 ... 1.3.
 - lp(5)** is the adaptive decreasing factor. Typical values: lp(5) = 1/lp(4) (lp(5) < 1).
- Delta_W_old**
The previous weight adjusting quantity. On first call this parameter should be initialized to zero using e.g. Delta_w_old = zeros(w). On subsequent calls to ann_FF_SSAB_online_nb you should give the value of Delta_W_old returned by the previous call.
- Delta_W_oldold**
The weight adjusting quantity used two steps back. Same remarks as for Delta_W_old (above) apply.
- T**
The number of epochs (training cycles trough all pattern set).
- mu**
This is the hypermatrix of learning constants who replaces lp(1) and is adapted at each step. This parameter is optional. Its default value is mu = lp(1)*ones(w), i.e. it is initialized uniformly to lp(1).
- af**

Activation function and its derivative. Row vector of strings:

af(1) name of activation function (string).

af(2) name of derivative.

⚠ Given the activation function $y=f(x)$, the derivative have to be expressed in terms of y , not x . This parameter is optional, default value is ['ann_log_activ', 'ann_d_log_activ'], i.e. logistic activation function and its derivative.

err_deriv_y

the name of error function derivative with respect to network outputs. This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.

ex

two-dimensional row vector of strings representing valid Scilab sequences. ex(1) is executed after the weight matrix have been updated, after each pattern (not whole set), using execstr. ex(2) - same as ex(1) - but is executed after each epoch. This parameter is optional, default value is [" ", " "] (do nothing).

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns. The algorithm used is online backpropagation with SuperSAB. This function is to be used on networks without biases.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init_nb](#) — initialize the weight hypermatrix (without bias).

ann_FF_Std_batch

standard batch backpropagation.

Syntax

```
W = ann_BP_Std_batch(x, t, N, W, lp, T [, af, ex])
```

Arguments

This function have same parameters as `ann_FF_Std_online` except for:

ex

string representing a valid Scilab program sequence, executed after each epoch trough `execstr`.

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is standard batch backpropagation.

See Also

- [ann_FF_Std_online](#) — online standard backpropagation.


ann_FF_Std_online

online standard backpropagation.

Syntax

```
W = ann_BP_Std_online(x, t, N, W, lp, T [, af, ex])
```

Arguments

- x** Matrix of input patterns, one pattern per column.
- t** Matrix of targets, one pattern per column. Each column have a correspondent column in x.
- N** Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector. N(size(N,'c')) is the size of output pattern vector (and also target).
- W** The weight hypermatrix (initialized first with ann_FF_init).
- lp** Learning parameters [lp(1), lp(2)].
lp(1) is the well known learning parameter of standard backpropagation algorithm, W is changed according to the formula:
lp(1) $W(t+1) = W(t) - lp(1) * grad E$
where t is the (discrete) time and E is the error. Typical values: 0.1 ... 1. Some networks train faster with even greater learning parameter.
lp(2) defines the threshold of error which is backpropagated: an error smaller than lp(2) (at one neuronal output) is rounded towards zero and thus not propagated. Typical values: 0 ... 0.1. E.g. assume that output of neuron n have the actual output 0.91 and the target (for that particular neuron, given the corresponding input) is 1. If lp(2) = 0.1 then the error term associated to n is rounded to 0 and thus not propagated.
- T** the number of epochs (training cycles through all pattern set).
- af** Activation function and its derivative. Row vector of strings:
af(1) name of activation function.
af(2) name of derivative.
 Given the activation function $y = f(x)$, the derivative have to be expressed in terms of y, not x. This parameter is optional. Its default value is ['ann_log_activ', 'ann_d_log_activ'], i.e. logistic activation function and its derivative.
- err_deriv_y** the name of error function derivative with respect to network outputs. This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.
- ex** two-dimensional row vector of strings representing valid Scilab sequences.
ex(1) is executed after the weight hypermatrix have been updated, after each pattern (not

whole set), using `execstr`.

`ex(2)`, same as `ex(1)`, but is executed after each epoch. This parameter is optional, default value is `[" "," "]` (do nothing).

Description

Returns the updated weight hypermatrix of a feedforward ANN, after online training with a given set of patterns. The algorithm used is online standard backpropagation.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init](#) — initialize the weight hypermatrix.
- [ann_FF_run](#) — run patterns through a feedforward net.

ann_FF_Std_batch_nb

standard batch backpropagation (without bias).

Syntax

```
W = ann_BP_Std_batch_nb(x, t, N, W, lp, T [, af, ex])
```

Arguments

This function have same parameters as `ann_FF_Std_online_nb` except for:

ex
string representing a valid Scilab program sequence, executed after each epoch trough `execstr`.

Description

Returns the updated weight hypermatrix of a feedforward ANN, after training with a given set of patterns, T times. The algorithm used is standard batch backpropagation. This function is to be used on networks without biases.

See Also

- [ann_FF_Std_online_nb](#) — online standard backpropagation


ann_FF_Std_online_nb

online standard backpropagation

Syntax

```
W = ann_BP_Std_online_nb(x, t, N, W, lp, T [, af, ex, err_deriv_y])
```

Arguments

- x**
Matrix of input patterns, one pattern per column.
- t**
Matrix of targets, one pattern per column. Each column have a correspondent column in x.
- N**
Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector. N(size(N,'c')) is the size of output pattern vector (and also target).
- W**
The weight hypermatrix (initialized first with ann_BP_init_nb).
- lp**
Learning parameters [lp(1), lp(2)].
is the well known learning parameter of standard backpropagation algorithm, W is changed according to the formula:
lp(1) $W(t+1) = W(t) - lp(1) * grad E$
where t is the (discrete) time and E is the error. Typical values: 0.1 ... 1. Some networks train faster with even greater learning parameter.
defines the threshold of error which is backpropagated: an error smaller that lp(2) (at one neuronal output) is rounded towards zero and thus not propagated.
lp(2) values: 0 ... 0.1. E.g. assume that output of neuron n have the actual output 0.91 and the target (for that particular neuron, given the corresponding input) is 1. If lp(2) = 0.1 then the error term associated to n is rounded to 0 and thus not propagated.
- T**
the number of epochs (training cycles through all pattern set).
- af**
Activation function and its derivative. Row vector of strings:
af(1) name of activation function.
af(2) name of derivative.
 Given the activation function $y = f(x)$, the derivative have to be expressed in terms of y, not x. This parameter is optional. Its default value is ['ann_log_activ', 'ann_d_log_activ'], i.e. logistic activation function and its derivative.
- err_deriv_y**
the name of error function derivative with respect to network outputs. This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.
- ex**
two-dimensional row vector of strings representing valid Scilab sequences.
ex(1) is executed after the weight hypermatrix have been updated, after each pattern (not

whole set), using `execstr`.

`ex(2)`, same as `ex(1)`, but is executed after each epoch. This parameter is optional, default value is `[" "," "]` (do nothing).

Description

Returns the updated weight hypermatrix of a feedforward ANN, after online training with a given set of patterns. The algorithm used is online standard backpropagation. This function is to be used on networks without biases.

See Also

- [ANN](#) — Toolbox for neural networks
- [ANN_GEN](#) — General utility functions
- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init_nb](#) — initialize the weight hypermatrix (without bias).
- [ann_FF_run_nb](#) — run patterns through a feedforward net (without bias).

ann_FF_grad

error gradient through finite differences.

Syntax

```
grad_E = ann_FF_grad(x, t, N, W, dW [, af, ef])
```

Arguments

grad_E

The error gradient, same layout as W . x Input patterns, one per column.

t

Target patterns, one per column. Each column have a correspondent column in x .

N

Row vector describing the number of neurons per layer. $N(1)$ is the size of input pattern vector, $N(\text{size}(N,'c'))$ is the size of output pattern vector (and also target).

W

The weight hypermatrix.

dW

The quantity used to perturb each W parameter.

af

The name of activation function to be used (string). This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.

ef

The name of error function to be used (string). This parameter is optional, default value "ann_sum_of_sqr", i.e. the sum-of-squares error function.

Description

Calculates error gradient through a (slow) finite differences procedure. Each element $W(n,i,l)$ is changed to $W(n,i,l)-dW$ then the error is calculated and the process is repeated for $W(n,i,l)+dW$.

From the values obtained the partial derivative of the sum-of-squares error function, with respect to $W(n,i,l)$, is calculated and the value of gradient returned.

This process is very slow (compared to the backpropagation algorithms) so it is to be used only for testing purposes.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_FF_grad_BP_nb

error gradient through backpropagation (without bias)

Syntax

```
W = ann_FF_grad_BP_nb(x, t, N, W [, c, af, err_deriv_y])
```

Arguments

x

Matrix of input patterns, one pattern per column.

t

Matrix of targets, one pattern per column. Each column have a correspondent column in x.

N

Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).

W

The weight hypermatrix (initialized first through ann_BP_init).

c

defines the threshold of error which is backpropagated: a error smaller that c (at one neuronal output) is rounded towards zero and thus not propagated. It have to be set to zero for exact calculation of gradient. This parameter is optional, default value 0.

af

Activation function and its derivative. Row vector of strings:

af(1) name of activation function.

af(2) name of derivative.

⚠ given the activation function $y=f(x)$, the derivative have to be expressed in terms of y, not x. This parameter is optional, default value is "['ann_log_activ', 'ann_d_log_activ']", i.e. logistic activation function and its derivative.

err_deriv_y

the name of error function derivative with respect to network outputs. This parameter is optional, default value is "ann_d_sum_of_sqr", i.e. the derivative of sum-of-squares.

Description

Returns the error gradient hypermatrix (it have the same layout as W) of a feedforward ANN, using the whole given training set. The algorithm used is standard backpropagation. This function is to be used on networks without biases.

Examples

This function is used as a low level engine in other algorithms (thus the reason for existence of c parameter), e.g. see implementation of ann_FF_ConjugGrad function.

See Also

- [ann_FF](#) — Algorithms for feedforward nets.
- [ann_FF_init_nb](#) — initialize the weight hypermatrix (without bias).

ann_FF_grad_nb

error gradient through finite differences

Syntax

```
grad_E = ann_FF_grad_nb(x, t, N, W, dW [, af, ef])
```

Arguments

grad_E

The error gradient, have same layout as W.

x

Input patterns, one per column.

t

Target patterns, one per column. Each column have a correspondent column in x.

N

Row vector describing the number of neurons per layer. N(1) is the size of input pattern vector, N(size(N,'c')) is the size of output pattern vector (and also target).

W

The weight hypermatrix.

dW

The quantity used to perturb each W parameter.

af

The activation function to be used. This parameter is optional, default value "ann_log_activ", i.e. the logistic activation function.

ef

The error function to be used. This parameter is optional, default value "ann_sum_of_sqr", i.e. the sum-of-squares error function.

Description

Calculates error gradient through a (slow) finite difference procedure. Each element $W(n,i,l)$ is changed to $W(n,i,l)-dW$ then the error is calculated and the process is repeated for $W(n,i,l)+dW$. From the values obtained the partial derivative of the sum-of-squares error function, with respect to $W(n,i,l)$, is calculated and the value of gradient returned. This process is very slow (compared to the backpropagation algorithms) so it is to be used only for testing purposes. This function is to be used on networks without biases.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_d_log_activ

derivative of logistic activation function

Syntax

```
z = ann_d_log_activ(y)
```

Description

This function is the derivative of default neuronal activation function.

Any other, user defined, function should have the same input and output format for variables.

Particularly note that the derivative is expressed in terms of activation, not total neuronal input.

Arguments

z

Matrix containing the derivative of activation, one pattern per column.

For each pattern (column): $z(i) = y(i) * [1 - y(i)]$ y Matrix containing the current neuronal activation, one pattern per column.

For each pattern: each $y(i)$ for the corresponding i-th neuron (on the current layer).

Components

General support functions See ANN_GEN for support functions for ANN. Feedforward networks
See ann_FF for detailed description.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_d_sum_of_sqr

derivative of sum-of-squares error

Syntax

```
err_d = ann_d_sum_of_sqr(y, t)
```

Arguments

err_d

Matrix containing the sum-of-squares error derivative, one per column, each column have a correspondent in y and t.

y

Matrix containing the actual network outputs, one per column, each column have a correspondent in t and err_d.

t

Matrix containing the targets, one per column, each column have a correspondent in y and err_d.

Description

This function calculates the derivative of sum-of-squares error given y and t.

Any other, user defined, error function should have the same input and output format for variables.

Note: this function is extremely simple, it just calculates y-t.

The only reason to provide it is to allow the option to change the default error function if so is desired.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_log_activ

logistic activation function

Syntax

```
y = ann_log_activ(x)
```

Arguments

y

Matrix containing the activation, one pattern per column. For each pattern:

$$y(i) = \frac{1}{1 + \exp[-x(i)]}$$

x

Matrix containing the total neuronal input, one pattern per column. For each pattern: each x(i) for the corresponding i-th neuron on the current layer.

Description

This function is the default neuronal activation function. Any other, user defined, function should have the same input and output format for variables.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

ann_pat_shuffle

shuffles randomly patterns for an ANN

Syntax

```
[x, t] = ann_pat_shuffle(x, t)
```

Arguments

- x**
Matrix containing the training inputs, one per column, each column have a correspondent in t.
- t**
Matrix containing the targets, one per column, each column have a correspondent in x.

Description

This function randomly shuffles the columns in matrices x and t, i.e. randomly shuffles the patterns. Some ANN train better if the training set is presented repeatedly but in random order. This function is intended to be called between two epochs. The correspondence between the columns in x and t is preserved. Note that x and t may change place, i.e. you may call this function as "[t,x]=ann_pat_shuffle(t,x)" as long as you keep your order.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.

[ANN_toolbox](#) > `ann_sum_of_sqr`

ann_sum_of_sqr

calculates sum-of-squares error

Syntax

```
E = ann_sum_of_sqr(y, t)
```

Arguments

- y**
Matrix containing the actual network outputs, one pattern per column, each column have a correspondent in t.
- t**
Matrix containing the targets, one pattern per column, each column have a correspondent in y.

Description

This function calculates the sum-of-squares error given y and t. Any other, user defined, error function should have the same input and output format for variables.

See Also

- [ANN](#) — Toolbox for neural networks
- [ann_FF](#) — Algorithms for feedforward nets.