train

Train a neural network

Syntax

```
[net,tr,Y,E,Pf,Af] = train(net,P,T,Pi,Ai,VV,TV)
```

To Get Help

Type `help network/train`

Description

`train` trains a network `net` according to `net.trainFcn` and `net.trainParam`.

`train(NET,P,T,Pi,Ai,VV,TV)` takes,

- `net` -- Neural Network
- `P` -- Network inputs
- `T` -- Network targets, default = zeros
- `Pi` -- Initial input delay conditions, default = zeros
- `Ai` -- Initial layer delay conditions, default = zeros
- `VV` -- Structure of validation vectors, default = []
- `TV` -- Structure of test vectors, default = []

and returns,

- `net` -- New network
- `TR` -- Training record (`epoch` and `perf`)
- `Y` -- Network outputs
- `E` -- Network errors.
- `Pf` -- Final input delay conditions
- `Af` -- Final layer delay conditions

Note that `T` is optional and need only be used for networks that require targets. `Pi` and `Pf` are also optional and need only be used for networks that have input or layer delays.

Optional arguments `VV` and `TV` are described below.

`train`'s signal arguments can have two formats: cell array or matrix.

The cell array format is easiest to describe. It is most convenient for networks with multiple inputs and outputs, and allows sequences of inputs to be presented:

- `P` -- `Ni x TS` cell array, each element `P{i,ts}` is an `Ri x Q` matrix

`T` -- `Nt` x `TS` cell array, each element `P{i,ts}` is an `Vi` x `Q` matrix

`Pi` -- `Ni` x `ID` cell array, each element `Pi{i,k}` is an `Ri` x `Q` matrix

`Ai` -- `Nl` x `LD` cell array, each element `Ai{i,k}` is an Si x `Q` matrix

`Y` -- `NO` x `TS` cell array, each element `Y{i,ts}` is an `Ui` x `Q` matrix

`E` -- `Nt` x `TS` cell array, each element `P{i,ts}` is an `Vi` x `Q` matrix

`Pf` -- `Ni` x `ID` cell array, each element `Pf{i,k}` is an `Ri` x `Q` matrix

`Af` -- `Nl` x `LD` cell array, each element `Af{i,k}` is an `Si` x `Q` matrix

where

`Ni` = `net.numInputs`

`Nl` = `net.numLayers`

`Nt` = `net.numTargets`

`ID` = `net.numInputDelays`

`LD` = `net.numLayerDelays`

`TS` = Number of time steps

`Q` = Batch size

`Ri` = `net.inputs{i}.size`

`Si` = `net.layers{i}.size`

`Vi` = `net.targets{i}.size`

The columns of `Pi`, `Pf`, `Ai`, and `Af` are ordered from the oldest delay condition to the most recent:

`Pi{i,k}` = input `i` at time `ts=k--ID`.

`Pf{i,k}` = input `i` at time `ts=TS+k--ID`.

`Ai{i,k}` = layer output `i` at time `ts=k--LD`.

`Af{i,k}` = layer output `i` at time `ts=TS+k--LD`.

The matrix format can be used if only one time step is to be simulated (`TS` = 1). It is convenient for networks with only one input and output, but can be used with networks that have more.

Each matrix argument is found by storing the elements of the corresponding cell array argument into a single matrix:

`P` -- `(sum of Ri)` x `Q` matrix

`T` -- `(sum of Vi)` x `Q` matrix

`Pi` -- `(sum of Ri)` x `(ID*Q)` matrix

`Ai` -- `(sum of Si)` x `(LD*Q)` matrix

`Y` -- `(sum of Ui)` x `Q` matrix

`E` -- (`sum of Vi`) x Q matrix

`Pf` -- (`sum of Ri`) x (`ID*Q`) matrix

`Af` -- (`sum of Si`) x (`LD*Q`) matrix

If `VV` and `TV` are supplied they should be an empty matrix [] or a structure with the following fields:

`VV.P, TV.P` -- Validation/test inputs

`VV.T, TV.T` -- Validation/test targets, default = zeros

`VV.Pi, TV.Pi` -- Validation/test initial input delay conditions, default = zeros

`VV.Ai, TV.Ai` -- Validation/test layer delay conditions, default = zeros

The validation vectors are used to stop training early if further training on the primary vectors will hurt generalization to the validation vectors. Test vector performance can be used to measure how well the network generalizes beyond primary and validation vectors. If `VV.T`, `VV.Pi`, or `VV.Ai` are set to an empty matrix or cell array, default values will be used. The same is true for `TV.T, TV.Pi, TV.Ai`.

Examples

Here input `P` and targets `T` define a simple function which we can plot:

```
p = [0 1 2 3 4 5 6 7 8];
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];
plot(p,t,'o')
```

Here newff is used to create a two-layer feed-forward network. The network will have an input (ranging from 0 to 8), followed by a layer of 10 tansig neurons, followed by a layer with 1 purelin neuron. trainlm backpropagation is used. The network is also simulated.

```
net = newff([0 8],[10 1],{'tansig' 'purelin'},'trainlm');
y1 = sim(net,p)
plot(p,t,'o',p,y1,'x')
```

Here the network is trained for up to 50 epochs to a error goal of 0.01, and then resimulated.

```
net.trainParam.epochs = 50;
net.trainParam.goal = 0.01;
net = train(net,p,t);
y2 = sim(net,p)
plot(p,t,'o',p,y1,'x',p,y2,'*')
```

Algorithm

`train` calls the function indicated by `net.trainFcn`, using the training parameter values indicated by `net.trainParam`.

Typically one epoch of training is defined as a single presentation of all input vectors to the network. The network is then updated according to the results of all those presentations.

Training occurs until a maximum number of epochs occurs, the performance goal is met, or any other stopping condition of the function `net.trainFcn` occurs.

Some training functions depart from this norm by presenting only one input vector (or sequence) each epoch. An input vector (or sequence) is chosen randomly each epoch from concurrent input vectors (or sequences). `newc` and `newsom` return networks that use `trainr`, a training function that presents each input vector once in random order.

See Also

sim, init, adapt, revert