train

Train neural network

Syntax

```
[net,tr] = train(net,X,T,Xi,Ai,EW)
[net,__] = train(__,'useParallel',__)
[net,__] = train(__,'useGPU',__)
[net,__] = train(__,'showResources',__)
[net,__] = train(Xcomposite,Tcomposite,__)
[net,__] = train(Xgpu,Tgpu,__)
net = train(__,'CheckpointFile','path/name','CheckpointDelay',numDelays)
```

Description

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

`[net,tr] = train(net,X,T,Xi,Ai,EW)` takes

`net`	Network
`X`	Network inputs
`T`	Network targets (default = zeros)
`Xi`	Initial input delay conditions (default = zeros)
`Ai`	Initial layer delay conditions (default = zeros)
`EW`	Error weights

and returns

`net`	Newly trained network
`tr`	Training record (`epoch` and `perf`)

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

`train` arguments can have two formats: matrices, for static problems and networks with single inputs and outputs, and cell arrays for multiple timesteps and networks with multiple inputs and outputs.

The matrix format is as follows:

`X`	`R`-by-`Q` matrix
`T`	`U`-by-`Q` matrix

The cell array format is more general, and more convenient for networks with multiple inputs and outputs, allowing sequences of inputs to be presented.

`X`	`Ni`-by-`TS` cell array	Each element `X{i,ts}` is an `Ri`-by-`Q` matrix.
`T`	`No`-by-`TS` cell array	Each element `T{i,ts}` is a `Ui`-by-`Q` matrix.
`Xi`	`Ni`-by-`ID` cell array	Each element `Xi{i,k}` is an `Ri`-by-`Q` matrix.
`Ai`	`Nl`-by-`LD` cell array	Each element `Ai{i,k}` is an `Si`-by-`Q` matrix.
`EW`	`No`-by-`TS` cell array	Each element `EW{i,ts}` is a `Ui`-by-`Q` matrix

where

Ni	=	net.numInputs
Nl	=	net.numLayers
No	=	net.numOutputs
ID	=	net.numInputDelays
LD	=	net.numLayerDelays
TS	=	Number of time steps
Q	=	Batch size
Ri	=	net.inputs{i}.size
Si	=	net.layers{i}.size
Ui	=	net.outptus{i}.size

The columns of Xi and Ai are ordered from the oldest delay condition to the most recent:

Xi{i,k}	=	Input i at time ts = k - ID
Ai{i,k}	=	Layer output i at time ts = k - LD

The error weights EW can also have a size of 1 in place of all or any of No, TS, Ui or Q. In that case, EW is automatically dimension extended to match the targets T. This allows for conveniently weighting the importance in any dimension (such as per sample) while having equal importance across another (such as time, with TS=1). If all dimensions are 1, for instance if EW = {1}, then all target values are treated with the same importance. That is the default value of EW.

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 in a single matrix:

X	(sum of Ri)-by-Q matrix
T	(sum of Ui)-by-Q matrix
Xi	(sum of Ri)-by-(ID*Q) matrix
Ai	(sum of Si)-by-(LD*Q) matrix
EW	(sum of Ui)-by-Q matrix

As noted above, the error weights EW can be of the same dimensions as the targets T, or have some dimensions set to 1. For instance if EW is 1-by-Q, then target samples will have different importances, but each element in a sample will have the same importance. If EW is (sum of Ui)-by-Q, then each output element has a different importance, with all samples treated with the same importance.

The training record TR is a structure whose fields depend on the network training function (net.NET.trainFcn). It can include fields such as:

- Training, data division, and performance functions and parameters
- Data division indices for training, validation and test sets
- Data division masks for training validation and test sets
- Number of epochs (num_epochs) and the best epoch (best_epoch).
- A list of training state names (states).
- Fields for each state name recording its value throughout training
- Performances of the best network (best_perf, best_vperf, best_tperf)

`[net, __ ] = train( __ ,'useParallel', __ )`, `[net, __ ] = train( __ ,'useGPU', __ )`, or `[net, __ ] = train( __ ,'showResources', __ )` accepts optional name/value pair arguments to control how calculations are performed. Two of these options allow training to happen faster or on larger datasets using parallel workers or GPU devices if Parallel Computing Toolbox is available. These are the optional name/value pairs:

`'useParallel','no'`	Calculations occur on normal MATLAB thread. This is the default `'useParallel'` setting.
`'useParallel','yes'`	Calculations occur on parallel workers if a parallel pool is open. Otherwise calculations occur on the normal MATLAB thread.
`'useGPU','no'`	Calculations occur on the CPU. This is the default 'useGPU' setting.
`'useGPU','yes'`	Calculations occur on the current gpuDevice if it is a supported GPU (See Parallel Computing Toolbox for GPU requirements.) If the current gpuDevice is not supported, calculations remain on the CPU. If `'useParallel'` is also `'yes'` and a parallel pool is open, then each worker with a unique GPU uses that GPU, other workers run calculations on their respective CPU cores.
`'useGPU','only'`	If no parallel pool is open, then this setting is the same as `'yes'`. If a parallel pool is open then only workers with unique GPUs are used. However, if a parallel pool is open, but no supported GPUs are available, then calculations revert to performing on all worker CPUs.
`'showResources','no'`	Do not display computing resources used at the command line. This is the default setting.
`'showResources','yes'`	Show at the command line a summary of the computing resources actually used. The actual resources may differ from the requested resources, if parallel or GPU computing is requested but a parallel pool is not open or a supported GPU is not available. When parallel workers are used, each worker's computation mode is described, including workers in the pool that are not used.
`'reduction',N`	For most neural networks, the default CPU training computation mode is a compiled MEX algorithm. However, for large networks the calculations might occur with a MATLAB calculation mode. This can be confirmed using `'showResources'`. If MATLAB is being used and memory is an issue, setting the reduction option to a value `N` greater than 1, reduces much of the temporary storage required to train by a factor of `N`, in exchange for longer training times.

`[net, __ ] = train(Xcomposite,Tcomposite, __ )` takes Composite data and returns Composite results. If Composite data is used, then `'useParallel'` is automatically set to `'yes'`.

`[net, __ ] = train(Xgpu,Tgpu, __ )` takes gpuArray data and returns gpuArray results. If gpuArray data is used, then `'useGPU'` is automatically set to `'yes'`.

`net = train( __ ,'CheckpointFile','path/name','CheckpointDelay',numDelays)` periodically saves intermediate values of the neural network and training record during training to the specified file. This protects training results from power failures, computer lock ups, Ctrl+C, or any other event that halts the training process before `train` returns normally.

The value for `'CheckpointFile'` can be set to a filename to save in the current working folder, to a file path in another folder, or to an empty string to disable checkpoint saves (the default value).

The optional parameter `'CheckpointDelay'` limits how often saves happen. Limiting the frequency of checkpoints can improve efficiency by keeping the amount of time saving checkpoints low compared to the time spent in calculations. It has a default value of 60, which means that checkpoint saves do not happen more than once per minute. Set the value of `'CheckpointDelay'` to 0 if you want checkpoint saves to occur only once every epoch.

Note Any `NaN` values in the inputs `X` or the targets `T`, are treated as missing data. If a column of `X` or `T` contains at least one `NaN`, that column is not used for training, testing, or validation.

Examples

Train and Plot Networks

Here input `x` and targets `t` define a simple function that you can plot:

```
x = [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(x,t,'o')
```

Here `feedforwardnet` creates a two-layer feed-forward network. The network has one hidden layer with ten neurons.

```
net = feedforwardnet(10);
net = configure(net,x,t);
y1 = net(x)
plot(x,t,'o',x,y1,'x')
```

The network is trained and then resimulated.

```
net = train(net,x,t);
y2 = net(x)
plot(x,t,'o',x,y1,'x',x,y2,'*')
```

Train NARX Time Series Network

This example trains an open-loop nonlinear-autoregressive network with external input, to model a levitated magnet system defined by a control current `x` and the magnet's vertical position response `t`, then simulates the network. The function `preparets` prepares the data before training and simulation. It creates the open-loop network's combined inputs `xo`, which contains both the external input `x` and previous values of position `t`. It also prepares the delay states `xi`.

```
[x,t] = maglev_dataset;
net = narxnet(10);
[xo,xi,~,to] = preparets(net,x,{},t);
net = train(net,xo,to,xi);
y = net(xo,xi)
```

This same system can also be simulated in closed-loop form.

```
netc = closeloop(net);
view(netc)
[xc,xi,ai,tc] = preparets(netc,x,{},t);
yc = netc(xc,xi,ai);
```

Train a Network in Parallel on a Parallel Pool

Parallel Computing Toolbox™ allows Neural Network Toolbox™ to simulate and train networks faster and on larger datasets than can fit on one PC. Parallel training is currently supported for backpropagation training only, not for self-organizing maps.

Here training and simulation happens across parallel MATLAB workers.

```
parpool
[X,T] = vinyl_dataset;
net = feedforwardnet(10);
net = train(net,X,T,'useParallel','yes','showResources','yes');
Y = net(X);
```

Use Composite values to distribute the data manually, and get back the results as a Composite value. If the data is loaded as it is distributed then while each piece of the dataset must fit in RAM, the entire dataset is limited only by the total RAM of all the workers.

```
[X,T] = vinyl_dataset;
Q = size(X,2);
Xc = Composite;
Tc = Composite;
numWorkers = numel(Xc);
ind = [0 ceil((1:4)*(Q/4))];
for i=1:numWorkers
    indi = (ind(i)+1):ind(i+1);
    Xc{i} = X(:,indi);
    Tc{i} = T(:,indi);
end
net = feedforwardnet;
net = configure(net,X,T);
net = train(net,Xc,Tc);
Yc = net(Xc);
```

Note in the example above the function configure was used to set the dimensions and processing settings of the network's inputs. This normally happens automatically when train is called, but when providing composite data this step must be done manually with non-Composite data.

Train a Network on GPUs

Networks can be trained using the current GPU device, if it is supported by Parallel Computing Toolbox. GPU training is currently supported for backpropagation training only, not for self-organizing maps.

```
[X,T] = vinyl_dataset;
net = feedforwardnet(10);
net = train(net,X,T,'useGPU','yes');
y = net(X);
```

To put the data on a GPU manually:

```
[X,T] = vinyl_dataset;
Xgpu = gpuArray(X);
Tgpu = gpuArray(T);
net = configure(net,X,T);
net = train(net,Xgpu,Tgpu);
Ygpu = net(Xgpu);
Y = gather(Ygpu);
```

Note in the example above the function configure was used to set the dimensions and processing settings of the network's inputs. This normally happens automatically when train is called, but when providing gpuArray data this step must be done manually with non-gpuArray data.

To run in parallel, with workers each assigned to a different unique GPU, with extra workers running on

CPU:

```
net = train(net,X,T,'useParallel','yes','useGPU','yes');
y = net(X);
```

Using only workers with unique GPUs might result in higher speed, as CPU workers might not keep up.

```
net = train(net,X,T,'useParallel','yes','useGPU','only');
Y = net(X);
```

Train Network Using Checkpoint Saves

Here a network is trained with checkpoints saved at a rate no greater than once every two minutes.

```
[x,t] = vinyl_dataset;
net = fitnet([60 30]);
net = train(net,x,t,'CheckpointFile','MyCheckpoint','CheckpointDelay',120);
```

After a computer failure, the latest network can be recovered and used to continue training from the point of failure. The checkpoint file includes a structure variable `checkpoint`, which includes the network, training record, filename, time, and number.

```
[x,t] = vinyl_dataset;
load MyCheckpoint
net = checkpoint.net;
net = train(net,x,t,'CheckpointFile','MyCheckpoint');
```

Another use for the checkpoint feature is when you stop a parallel training session (started with the `'UseParallel'` parameter) even though the Neural Network Training Tool is not available during parallel training. In this case, set a `'CheckpointFile'`, use Ctrl+C to stop training any time, then load your checkpoint file to get the network and training record.

More About

collapse all

Algorithms

`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 for each epoch from concurrent input vectors (or sequences). `competlayer` returns networks that use `trainru`, a training function that does this.

See Also

`adapt` | `init` | `revert` | `sim`

Introduced before R2006a