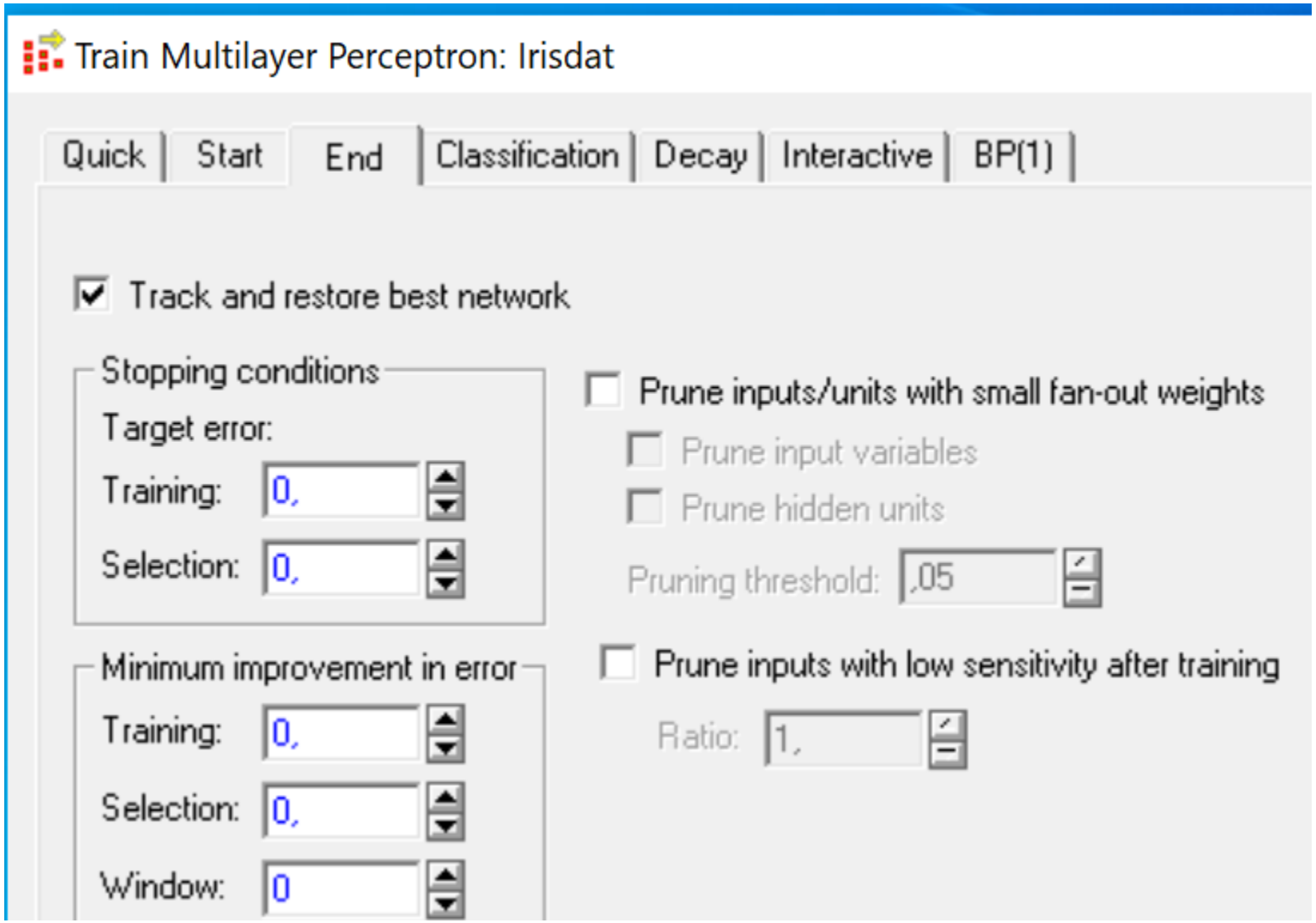
SNN Example 13 – Stopping Conditions

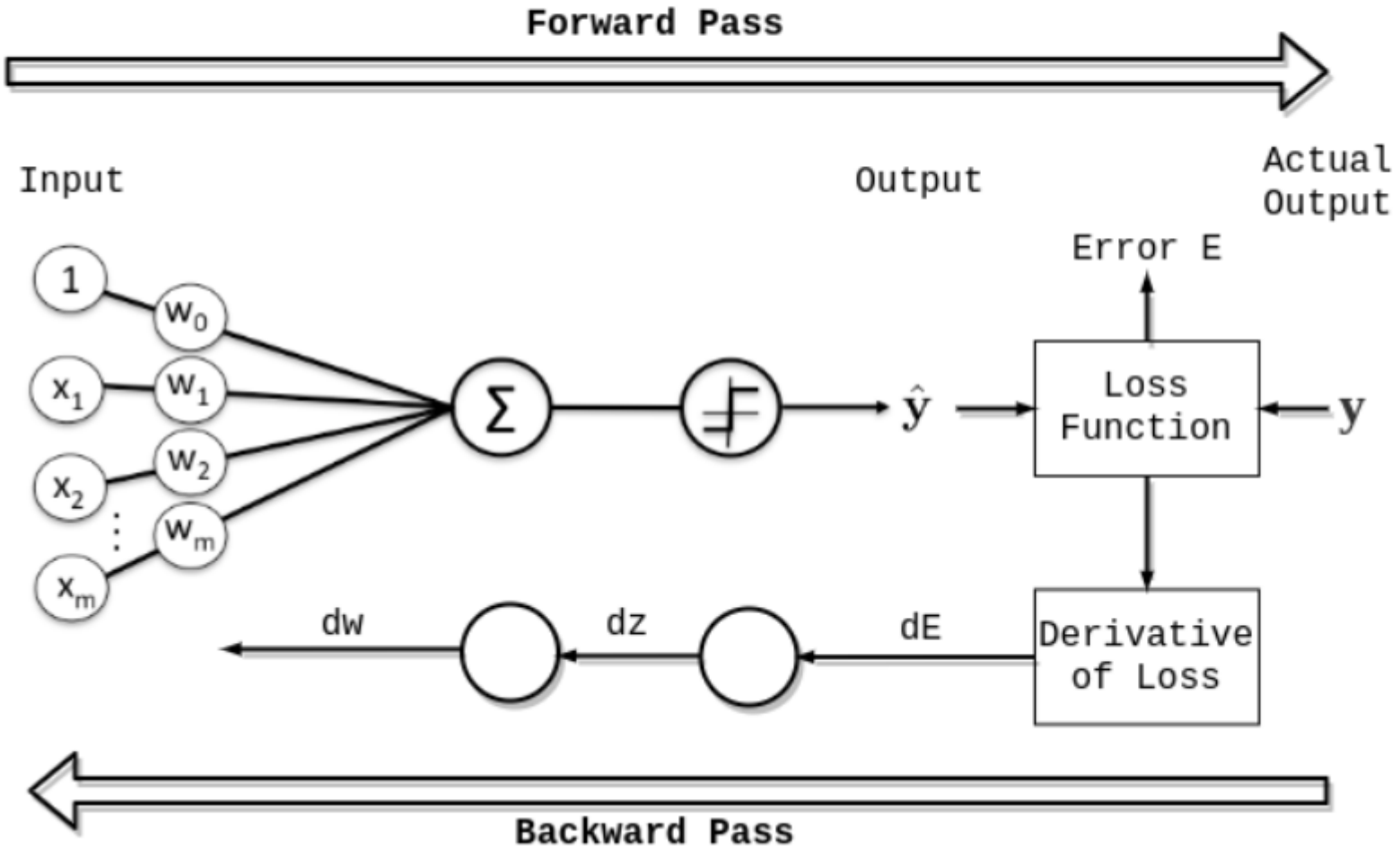


So far we have trained networks for a set number of epochs (see Example 12: Training the Network).

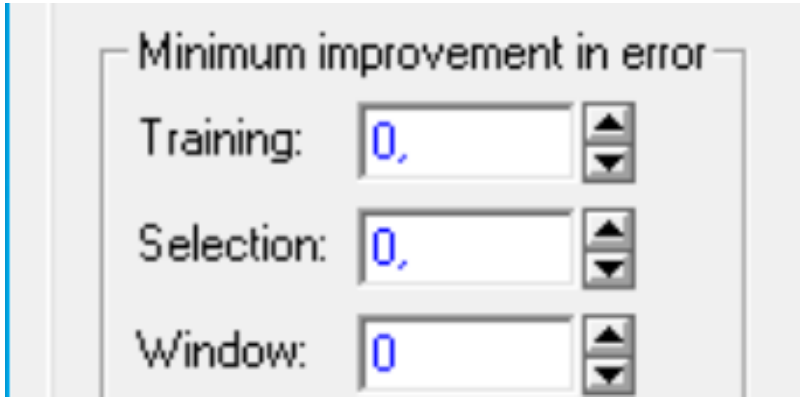
This is a reasonable approach, especially if you are sitting and watching the network train, in which case you can always click the *Finish* button or the *Cancel* button to abort training if you decide it is not doing well.

However, for longer training runs there may be better ways to specify when training should stop. You can do this in *STATISTICA Neural Networks* on the *End tab of the Train Multilayer Perceptron dialog.*

An epoch means training the neural network with all the training data for one cycle. In an epoch, we use all of the data exactly once. A forward pass and a backward pass together are counted as one pass:

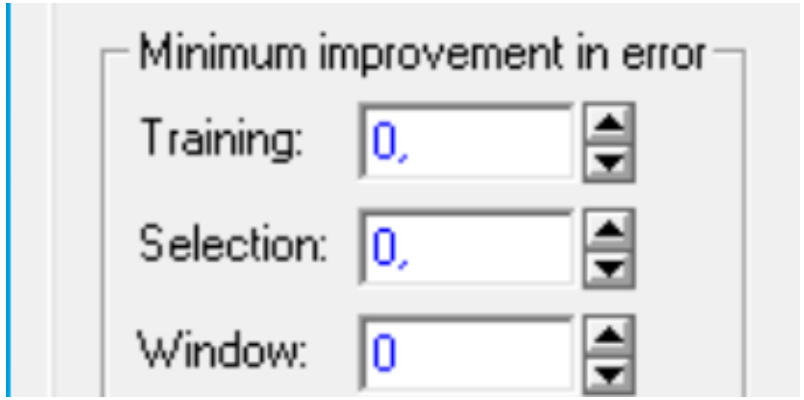


MLP



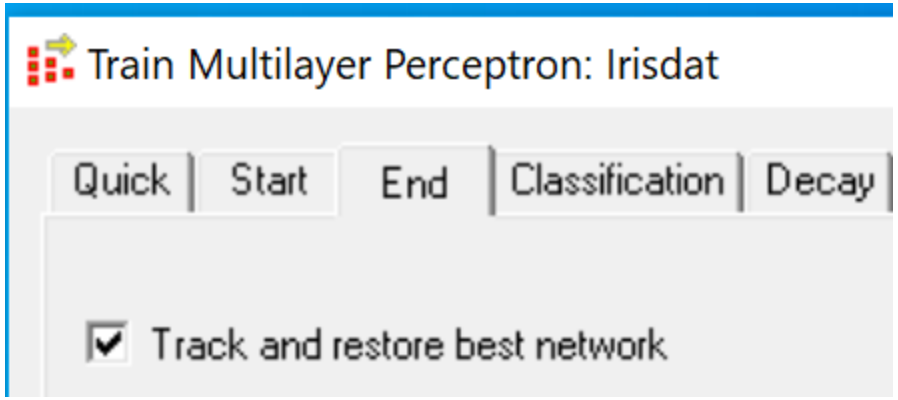
The stopping conditions are actually used by all the iterative training algorithms in *STATISTICA Neural Networks*, including back propagation and conjugate gradient descent. Besides the maximum number of epochs, you can also specify a target error level at which training should stop, and/or a minimum level of improvement over a given number of epochs.
The most useful option is perhaps the ***Minimum improvement in error***. This states that if the training and selection errors do not improve by at least the amount given over a set number of epochs (the *Window*) then training should stop.

For example, set the ***Window* to one epoch** and click *OK* to train the network. If you have left the *Minimum improvement in error Training* and *Selection* thresholds both as 0, *STATISTICA Neural Networks* will stop training if either the training or selection errors deteriorate. (Click *Cancel* on the *Results* dialog to return to the *Train Multilayer Perceptron* dialog.)
You can check for **deterioration** in the selection error only by setting the *Minimum improvement in error* threshold for ***Training* to *-1***, and leaving the *Selection* threshold at 0. This means that even a wild deterioration in training error is acceptable, but no deterioration in selection error is acceptable - effectively checking selection error only. When using back propagation, the training error may sometimes deteriorate. When using conjugate gradient descent the training error will not deteriorate, but the selection error may do so.



One problem with this approach is that the error often fluctuates during training (especially if using back propagation with a high learning rate and shuffle option, but also to a lesser extent for other algorithms), sometimes rising only to fall again soon afterwards. You can adjust the *Window* to take account of this, so that training only ceases if performance is unsatisfactory for a number of epochs. With a window of ten epochs, for example, training only stops if the error deteriorates, and then fails to improve over the last best value for ten epochs. This allows you to stop when a clear trend of deterioration has set in (probably indicating the onset of over-learning), without stopping prematurely because of random noise in the training process.

The *Intelligent Problem Solver* employs stopping conditions on both training and selection, with a relatively long **window (typically 50 epochs**) to ensure that learning is not terminated prematurely. For most problem domains, you can afford to be a little less conservative than that, and a window as low as 20 epochs is acceptable (provided that the back propagation learning rate, which can cause fluctuation, is kept low).



Tracking the best network. There remains a problem. When training is stopped due to deterioration in the error rate, the final network is (by definition) not the best one discovered during the training run. The network we should probably treat as the output of the training process is the one with the lowest selection error discovered during the training run - not the final network.

By default, *STATISTICA Neural Networks* keeps a copy of the best network found during training (i.e. the network with the ***lowest selection error***), and when training finishes, it is this network that is preserved. The problem with this approach is that it imposes **some computational expense** - a copy of the network must be made at the end of every epoch on which an improvement in selection error is made. Consequently, if you do not want to keep the best network (e.g. if you are conducting experiments with network architectures, sampling or learning parameters, but do not at this stage intend to keep resulting networks) then you can turn off the feature. Simply clear the *Track and restore best network* check box on the *End tab of the Train Multilayer Perceptron dialog.*