OXlearn

user manual

Dr. Nicolas Ruh

Oxford Brookes University

The development of OXlearn was supported by ESRC grant Res-061-23-0129 to Gert Westermann.

Table of contents:

Introduction

OXlearn is a neural network simulation software that enables you to build, train, test and analyse connectionist neural network models. Because OXlearn is implemented in MATLAB you can run it on all operation systems (Windows, Linux, MAC, etc.), provided you have a recent version of MATLAB installed (R2006b or later). This also has the additional advantage that it makes it easy to 'look under the hood' so you can inspect the calculations performed by the program or adapt the program to your specific needs.

First and foremost, OXlearn is designed as an educational tool that provides a quick and easy start to neural network modelling. OXlearn provides a Graphical User Interface (GUI) that enables access to most of its functionality, no programming is needed. In line with the principal aim to foster understanding of neural network models, OXlearn is set up for maximum transparency. By using native MATLAB components such as the workspace browser or the array editor, you can easily retrace most of the manipulations implemented in the GUI – or you can choose to manipulate data (changing parameters, plotting data, etc.) directly in MATLAB. The main part of this manual describes OXlearn's general structure and functionality and provides a detailed description of all the parameters that are internally manipulated.

The secondary aim of OXlearn, also aided by the above mentioned transparency, is to facilitate extension of the existing functionality. For example, if you wanted to implement a different learning algorithm or a network architecture that is not (yet) included, you could just add in the dedicated function(s) while still making use of OXlearn's inbuilt utilities and interface. This latter step requires some proficiency in writing MATLAB code, of course, although it should usually be possible to take existing program files as a starting point.

Using OXlearn

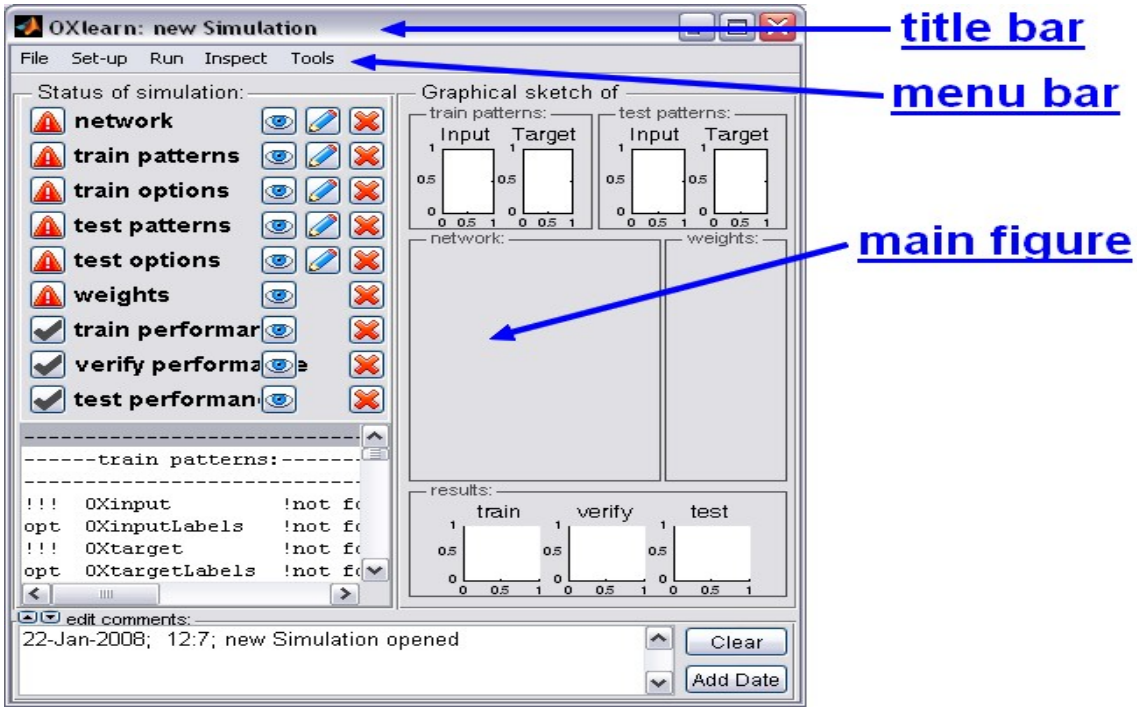
Starting OXlearn

Before you can start OXlearn, you have to make sure that Matlab knows where to find it. Because Matlab always knows the content of its "current directory", you can use the standard browsing interface at the top of MATLAB's main window to browse to the location on your hard drive where you have saved the (unzipped) OXlearn folder. Alternatively, the OXlearn folder can be added to the MATLAB path (i.e., a list of locations known to MATLAB), e.g., through the interface under `File -> Set Path`. Provided that you have administrator rights, this latter solution has the advantage of adding the new location permanently to the search path so that MATLAB will still know where to find OXlearn when you restart it.

To start OXlean, type *'OXlearn'* in the MATLAB command window and press return. As an alternative to always starting OXlearn from the command line, you can also create a shortcut. To do so, select the string *"OXlearn"* that you just typed in the command window and drag it over the shortcuts toolbar (located at the top of your MATLAB window, the bottom most toolbar). From now on, a single click on this shortcut will suffice to start OXlearn.

The command (or shortcut) "OXlearn" will open an empty simulation in the OXlearn GUI. If you want OXlearn to load a specific simulation right away, simply add the name of the file (in parentheses and quotes) to the command, e.g., *"OXlearn('XOR.mat')"*.

Layout of the graphical user interface (GUI)



Starting OXlearn will open the OXlearn GUI, which entails three parts:

(1) In the title bar of the window you will find a label, consisting of the string "OXlearn:" and the name of the current simulation file (e.g. "OXlearn: new Simulation" if you have opened an empty simulation, or "OXlearn: XOR.mat" if you have opened, loaded or saved (as) a simulation with that name).

(2) The menu bar which represents the main means to interact with the OXlearn GUI (on a MAC, as usual, the menu bar will appear at the top of your screen). The menu bar has five main menus ('File', 'Set-up', 'Run', 'Inspect' and 'Tools') with a number of selectable sub menus each.

(3) The main part of the figure contains various displays, dependent on your selection in the 'Inspect' or 'Analysis' menus. Initially, an overview concerning the status of the current simulation will be shown. To return to this display at any time, select Inspect -> Simulation.

Workflow

Every simulation project involves three general steps: (1) preparing a simulation by defining the exact set-up, (2) running the simulation (training the network, verifying that it has learned and, possibly, testing the networks performance with specific or novel stimuli) and, finally, (3) analysing the networks behaviour/performance. This succession of steps is roughly mirrored in the layout of the menu bar, where the first two elements ('File' and 'Set-up') are mainly used to define the exact set-up of your simulation, the 'Run' menu in the middle accesses training, verifying and testing actions, and the last two menus ('Inspect' and 'Tools') provide useful tools for visualisation and analysis of your simulation.

Although this order reflects the general structure of a typical workflow, it is of course not imperative to use the interface in a strictly left to right (and, within each menu, top to bottom) fashion. For example, you might want to use the displays to inspect your set-up before having trained the network. Or you could train a network, have a closer look at its performance during training, and then go back to set-up a generalisation test which is subsequently run and analysed. Also, you will often want to change something in the set-up of an existing simulation and then repeat the cycle of training it and analysing the performance. For the illustrative purpose of this manual, however, we will assume the standard case of creating a simple simulation of the XOR problem from scratch.

Concepts

Before going into further details, it is important that you understand the general concept behind OXlearn. While there is no need to go into any technical details, this will make it much easier for you to find your way around the program and neural network models in general.

OXparams

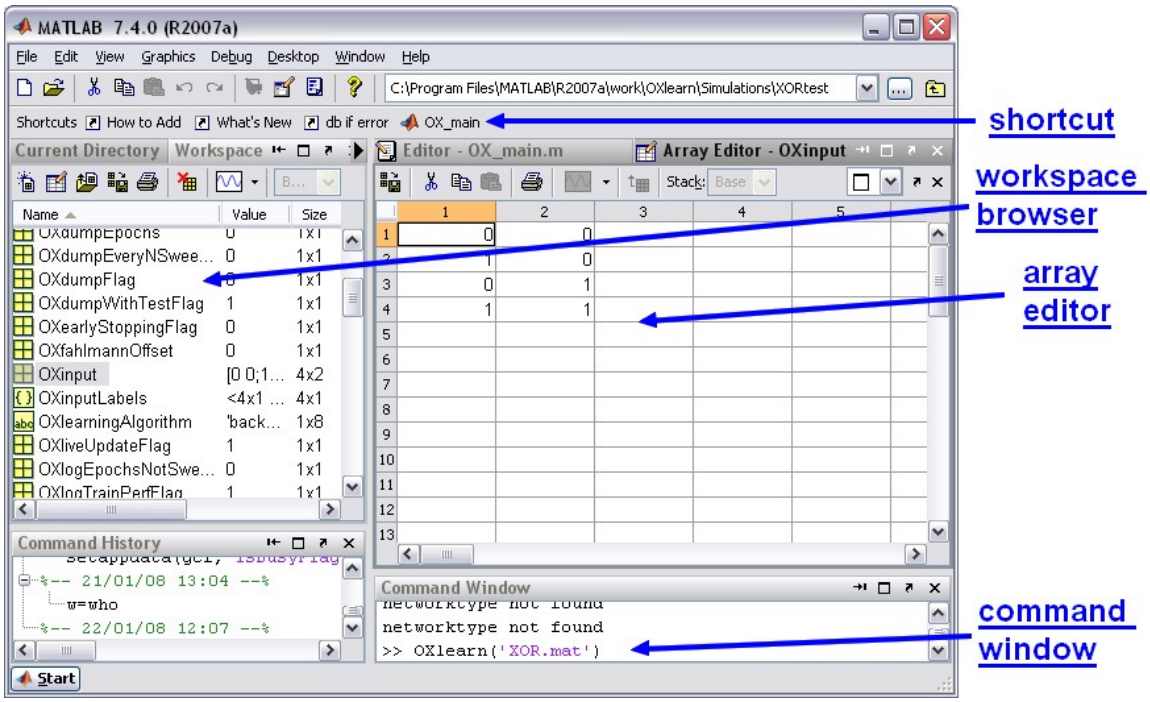
Try to think about it like this: what do we need to fully describe a neural network model?

1. To start with, we need to be explicit about the stimuli the model deals with, sometimes referred to as the *environment*. This includes the number and identity of input and target patterns, as well as labels or grouping information. If the model is to be tested on novel patterns after training (generalisation test), we also need information regarding these test patterns.
2. Then there is the network *architecture* to be defined, for example the number of layers, how many units they have, how they are connected, if there is a bias, etc.
3. And finally, we need to determine which learning algorithm is used, including important parameters such as learning rate or the number of patterns to be presented (= sweeps) before training stops. We will call this the training *paradigm*.

It is clear that all of this information needs to be given before the network can actually be trained (or tested in the case of the test patterns), and therefore the "Set-up" menu provides the means to manipulate such information in appropriately organised pop-up windows. All the information that you can manipulate in these pop-up windows corresponds directly to variables that live in MATLAB's basic workspace. The names of these variables will usually be displayed as tooltips, e.g. when you point with the mouse at the dropdown menu that allows you to select the network type (in Set-up -> Network) you will see that the variable representing this choice is called "OXnetworkType". In a similar vein, every value that you can select, tick or change in the set-up windows has a corresponding variable name, all starting with 'OX' (from now on, these will be referred to as OXparams).

OXparams are just variables, each having a name and value(s) that live in the MATLAB workspace.

The workspace



If you are interested in sneaking a peek under the hood of OXlearn, there is a very simple way of inspecting all the OXparams directly: go back to your main MATLAB window and have a look at the workspace browser (type "*workspace*" in the command window if you do not find such a window on the left hand side). You will see that it contains a number of OXparams (if you are still working on an empty simulation you might only find one, 'OXcomments', which was automatically created upon opening OXlearn. To see more, you could load an existing simulation file or simply open some of the set-up windows, thereby creating the corresponding parameters and initializing them to default values). The MATLAB workspace browser also contains additional information as to the format, value, size, etc. of these OXparams (double-click on a parameter name to have its value displayed in the MATLAB array editor). An experienced user might sometimes find it more efficient to inspect and manipulate OXparams directly in the workspace browser or the array editor.

Note that the underlying workspace is not directly accessible if you are working with the standalone version of OXlearn (OXlearn.exe, which does not require you to have a MATLAB license). Apart from not being able to directly inspect or manipulate variables in the workspace, however, there is no difference between the standalone and the Matlab version of OXlearn.

The content of a simulation file

Now, the important thing to understand is this: the content of an OXlearn simulation file (saved in MATLAB's native data format that ends on '.mat') is nothing but these OXparams (i.e., their names and values). When using the sub-menus under "Set-up", you will automatically create the appropriate OXparams and determine their values. When saving a simulation file (File -> Save Simulation or -> Save Simulation As), all the

existing OXparams will be stored in a '.mat' file. If you load a simulation file (File -> Load Simulation) they will be put back into the workspace.

In addition to the parameters pertaining to the network *architecture*, the training *environment*, and the training *paradigm*, there is one further type of OXparams: the variables that store the results of running an appropriately set-up simulation. Let's have a more detailed look at this: Choosing any of the options available under 'Run' will take all the *set-up OXparams* that specify the current simulation, perform the specified operation – e.g., training or testing the network - and put all changed or novel *performance OXparams* back into the workspace. For example, if you have trained a network (Run -> Train Network), the *OXparams* specifying the weights (e.g. 'OXweightsInputToHidden'), and also some information regarding the networks performance during training (e.g. 'OXtrainError') will be given back. The only difference between *set-up* and *performance OXparams* is that the latter are not meant to be manipulated directly, but rather are the outcome of training, verifying or testing the network. The displays available under 'Inspect' and 'Tools', finally, also operate directly on the *OXparams* (both *set-up* and *performance*) in the workspace but will not change them.

Why is this important? As you will see, the notion of the underlying OXparams that contain all the essential information about an OXlearn simulation will often recur when describing what the individual components do. Secondly, it helps with troubleshooting, as almost all the errors you might encounter when using OXlearn will have to do with particular OXparams being missing or having an incorrect format (e.g. letters where numbers are expected). And thirdly, demystifying the inner workings of OXlearn enables you to use some of MATLAB's native functionality (e.g. the data import wizard or the array editor) if you want to perform specific kinds of data manipulation for which no suitable OXlearn tool exists.

Implications and common problems

(1) You can create/manipulate the parameters pertaining to the networks environment, architecture and learning algorithm by using the 'Set-up' menus or directly in the workspace. In the latter case you need to ensure appropriate naming and format.

(2) All the options under the 'Run' menu require appropriate set-up OXparams to exist and have valid formats. These requirements differ slightly for training/verifying, which does not rely on test patterns being specified, and testing, which obviously does. Which exact OXparams are expected furthermore depends on the network type and the learning algorithm chosen.

(3) The process of training, verifying or testing a network results in specific performance OXparams being created and put back into the workspace. Therefore, each simulation file can hold only one set of these performance OXparams for each of the operations (train/verify/test), respectively. If you want to run a second test with the same network, make sure to have saved the previous version.

(4) If you train several instances of the current network, or if you dump intermediate states during training, each version will be automatically saved under a different file name (OXlearn will add suffixes to the name of your simulation) and only the last one will remain open after training terminates, still bearing the original name. You will need to reopen the automatically saved files in order to inspect or analyze them.

The only way to work with several simulations or states of a simulation at once is by using the network comparison tool (Tools -> Compare Networks).

(5) If you change set-up OXparams in a simulation that already contains performance-OXparams (i.e. the network has already been trained/verified/tested), your simulation might become temporally inconsistent because it contains the result of training a network with a different (the previous) set-up. This inconsistency will be resolved once you have trained/verified/tested the network with the new set-up. To avoid possible confusion, however, it is recommended to clear all performance OXparams (File -> Reset Simulation) before changing anything in the set-up.

(6) We said earlier that .mat files in general, and OXlearn simulation files more specifically, may contain a variable number of name-value pairs. If you import data from a .mat file, the import wizard provides you with a preview of the content and allows you to decide which of the variables you actually want to import. However, OXlearn will only recognize the imported variables if the have valid OXparam names. To ensure this (if necessary) you can change the name of any imported variable by right-clicking on it within the import wizard. Alternatively, you can rename variables in the workspace browser (right-click).

(7) The requirement of assigning valid OXparam names is also given when you import data from other sources, such as text or Excel files. In this case, however, it is difficult to import several variables at once and you should usually import the data corresponding to each OXparam separately.

List of OXparams

Name	Description	Content	Default	Display
OXactFcnH	activation function of nodes in the hidden layer	'sigmoid' 'threshold' 'linear'	'sigmoid'	Set-up -> Network
OXactFcnO	activation function of nodes in the output layer	'sigmoid' 'threshold' 'linear'	'sigmoid'	Set-up -> Network
OXautoTestFlag	toggles automatic testing when end of training is reached or when an simulation file is dumped	0 or 1 (false/true)	0	Set-up -> Training Options
OXautoVerifyFlag	toggles automatic verifying when end of training is reached or when an simulation file is dumped	0 or 1 (false/true)	1	Set-up -> Training Options
OXbH	whether the hidden layer includes a bias node	0 or 1 (false/true)	1	Set-up -> Network
OXbO	whether the output layer includes a bias node	0 or 1 (false/true)	1	Set-up -> Network
OXcomments	user editable comments and time stamps	user defined		Inspect -> Simulation
OXdumpEveryNSweeps	interval (in sweeps) to dump simulation file during training		optional	Set-up -> Training Options
OXdumpFlag	whether to dump simulation files	0 or 1 (false/true)	0	Set-up -> Training Options
OXearlyStoppingFlag	whether to consider early stopping	0 or 1 (false/true)	0	Set-up -> Training Options
OXfahlmannOffset	parameter in some learning algorithms that prevents weights from changing any more when the error is negligable	<0	0	Set-up -> Training Options
OXinput	matrix that represents input patterns (rows) for training (and verifying)	matrix of numbers		Set-up -> Train Patterns
OXinputLabels	labels for the input patterns	eq rows OXinput	optional	Set-up -> Train Patterns
OXlearningAlgorithm	learning algorithm to be used	'backprop' 'quickprop' 'BPTT'	backprop'	Set-up -> Training Options
OXliveUpdateFlag	whether the Oxlearn displays are	0 or 1	1	Set-up ->

	updated during the training process	(false/true)		Training Options
OXlogTrainPerfFlag	whether the networks performance during training is logged	0 or 1 (false/true)		Set-up -> Training Options
OXlogTrainPerfInterval	how often (in sweeps) the training performance is logged	<OXmaxSweeps	1	Set-up -> Training Options
OXlr	learning rate; the magnitude of weight adjustments	small number	0.5	Set-up -> Training Options
OXmaxSweeps	maximum number of sweeps after which training is terminated.	number	1000	Set-up -> Training Options
OXmomentum	parameter in some learning algorithms that governs how large a proportion of the previous weight change is added to the current adjustment of weights	small number	0	Set-up -> Training Options
OXnH	number of units in the hidden layer		0	Set-up -> Network
OXnI	number of units in the input layer	eq columns OXinput	0	Set-up -> Network
OXnO	number of units in the output layer	eq columns OXtarget	0	Set-up -> Network
OXnSweeps	number of sweeps performed, equals maxSweeps at the end of training	<OXmax-Sweeps		
OXnetworkType	architecture of the network	‘2-layer feed-forward’ ‘3-layer feed-forward’ ‘SRN’	‘2-layer feed-forward'	Set-up -> Network
OXpresentationOrder	order in which the input patterns are presented during training	‘sequential’ ‘random with replacement’ ‘random without replacement’	‘sequential'	Set-up -> Training Options
OXrunNr	index when training several instances of a network		optional	
OXseedNr	id of the seed used by the random number generator	<2^32	optional	Set-up -> Training Options
OXsimFileName	name of the current simulation	user defined string		File -> Save Simulation As
OXsimFilePath	location (on the computer) to which the simulation file is saved	user defined	“OXlearn\ Simulations”	File -> Save

		path		Simulation As
OXstopCritType	operator used by the early stopping criterion		optional	Set-up -> Training Options
OXstopCritValue	value against which the early stopping criterion evaluates		optional	Set-up -> Training Options
OXstopCritWindow	number of consecutive sweeps for which the early stopping criterion must be fulfilled		optional	Set-up -> Training Options
OXtarget	matrix that represents target patterns (rows) for training (and verifying)	matrix of numbers		Set-up -> Train Patterns
OXtargetLabels	labels for the target patterns	eq rows OXtarget	optional	Set-up -> Train Patterns
OXtestGroups	group vector for test patterns	eq rows OXtestInput	optional	Set-up -> Test Patterns
OXtestHidden	hidden layer activations during testing	RESULTS	RESULTS	Inspect -> Patterns
OXtestInput	matrix that represents input patterns (rows) for testing	matrix of numbers		Set-up -> Test Patterns
OXtestInputLabels'	labels for the test input patterns	eq rows OXtestInput		Set-up -> Test Patterns
OXtestLogHiddenActFlag	whether to log hidden activation during testing	0 or 1 (false/true)	1	Set-up -> Test Options
OXtestLogOutputActFlag	whether to log output activation during testing	0 or 1 (false/true)	1	Set-up -> Test Options
OXtestOutput	raw activation of the output layer during testing	RESULTS	RESULTS	Inspect -> Patterns
OXtestTarget	matrix that represents target patterns (rows) for testing	matrix of numbers		Set-up -> Test Patterns
OXtestTargetLabels	labels for the test target patterns	eq rows OXtestTarget		Set-up -> Test Patterns
OXtimeStamp	date and time where a simulation file was saved	date and time	now	
OXtrainCorrect	performance during training, evaluated against the correctness	RESULTS	RESULTS	Inspect -> Performanc

	criterion of 'deviation < 0.1'			e
OXtrainError	performance during training measured as mean squared error	RESULTS	RESULTS	Inspect -> Patterns
OXtrainGroups	group vector for train/verify patterns	eq rows OXinput		Set-up -> Train Patterns
OXtrainOrderLog	the actual order in which the train patterns have been presented during training	RESULTS	RESULTS	Inspect -> Performanc e
OXverifyHidden	hidden layer activations during verifying	RESULTS	RESULTS	Inspect -> Patterns
OXverifyOutput	output layer activations during verifying	RESULTS	RESULTS	Inspect -> Patterns
OXwInitMean	mean of the distribution from which initial weight values are drawn	number	0	Set-up -> Training Options
OXwInitRange	range of the distribution from which initial weight values are drawn	number	0.1	Set-up -> Training Options
OXwInitType	whether and how to initialize weight values. A specific seed allows to recreate a specific random pattern	‘random seed’ ‘seed Nr:’ ‘use existing weights’	‘random seed’	Set-up -> Training Options
OXweightsHiddenTo-Output	weights from the hidden to the output layer	RESULTS	RESULTS	Inspect -> Weights
OXweightsInputToHidden	weights from the input to the hidden layer	RESULTS	RESULTS	Inspect -> Weights
OXweightsToHiddenBias	weights from the bias node (always 1) to the hidden layer	RESULTS	RESULTS	Inspect -> Weights
OXweightsToOutputBias	weights from the bias node (always 1) to the output layer	RESULTS	RESULTS	Inspect -> Weights

The OXlearn GUI

Control elements

The different displays within OXlearn enable the set-up and the detailed graphical investigation of various components of a simulation. In most display windows you will find a panel with several control elements at the left hand side. These buttons let you interact with the graphs that occupy the main portion of the display, e.g. by changing the appearance of the displayed data (zooming, coloring, etc.) or by providing additional information (enabling datatips, adding colorbars, etc.). The "Options" panel at the bottom part of most displays influences which data are shown, e.g. information from training, verifying or testing the network. Similarly, you will often find tickboxes to control which parts of the data are shown or hidden.

Most of the functionality provided by the various control elements (buttons, tickboxes, drop-down menus, etc.) should be self-explanatory – note that a short description will be displayed as tooltip when the mouse pointer hovers over a control element. However, in the following you will find a short description of the functionality of all control elements within OXlearn's various displays.

The extract button 

This button opens a new MATLAB figure and extracts a snapshot of the currently displayed data into it. This novel window does not contain any of the control elements and will not be updated in case the underlying data change. Instead of the OXlearn control elements, however, you will find that all the native MATLAB graphics tools are accessible in this new Window (see the menus and toolbars at the top). With their help you can change every part of the appearance of the figure (labels, colors, legends, annotations, etc.), please refer to the MATLAB help for information on how to use these tools. It is also possible to organize several extracted figures with the dock controls in the top right corner of the figure. And finally, you can save the graph in a variety of formats – use the 'save' button or choose File -> Save Simulation or Save Simulation As. Make sure to change the type (choose '.jpg', for example) if you do not want to save the figure in MATLAB's native .fig format (you will find advanced options under File -> Export Setup).

The colorbar button 

This button toggles the display of a colorbar at the right hand side of the current axes (click on a graph to make it current). The colorbar indicates the correspondence between the colors in a graph and the underlying data values.

The datatip button 

This button toggles the datacursormode in which additional information regarding a specific datapoint is displayed when you click on a patch/line/dot in a graph. For some displays this mode is initially enabled.

The zoom in button 

This button toggles the zoom mode. When zoom mode is on, clicking within an axes will zoom in by a specific amount, centered around the region you have clicked on. You can also use the mouse wheel to continuously zoom in or out or specify the region you want to look at by dragging out a rectangle or line with the primary mouse button held down. Double clicking will usually restore the original view (see also the zoom out button). Note that, depending on the nature of the data displayed, zooming might be restricted to one dimension (horizontal or vertical) and several graphs might be coupled with respect to their zooming behaviour. To circumvent these restrictions, extract the graph (extract button) and use MATLAB's zoom tool on the extracted figure.

The zoom out button 

This button will disable the zoom mode and restore the original view.

The pan button 

This button toggles the pan mode in which you can drag the elements within a graph by moving the mouse within the graph with the primary button pressed down. This usually is most useful when you have zoomed into an appropriate level of detail, but now want to inspect neighbouring data points.

The rotate button 

This button toggles the rotate mode in which you can rotate a (usually 3 dimensional) graph by moving the mouse within the graph with the primary button pressed down. Double clicking will usually restore the original view (see also the zoom out button). You might also want to explore the options available in the context menu that appears when you perform a right click on the graph (when in rotate mode), e.g. to select specific dimensions.

The scroll (up/down) buttons 

These buttons are specific to the Inspect/Set-up -> Patterns displays. When you have zoomed into the graph, pressing these buttons will result in the next/previous portion of the data to be shown, similar to the page up/down keys in a text editor or internet browser.

The skip (up/down) buttons 

These buttons are specific to the Inspect -> Activations display. Pressing these buttons will result in the next/previous pattern to be displayed. Note that 'next' in this context usually means the following pattern within the epoch of train or test patterns as determined in the Set-up -> Train Patterns/Test Patterns window. The number displayed between the two skip buttons corresponds to the index of the currently displayed pattern within the (sorted) epoch. With sort by error disabled, this index will always correspond

to the one indicated in the title of the individual bar graphs. You can also set the number in this box directly.

The sort by error button 

This button toggles skipping through patterns in the order of ascending/descending error, i.e., the 'next' pattern would be the one with the next higher/lower error. When sort by error is enabled, the number displayed between the two skip buttons refers to this alternative order. The number one, for example, indicates that the pattern with the highest error is currently displayed.

The group button 

This button is specific to the Inspect -> Performance and the Tools -> Compare Networks displays. This button influences the colouring of the performance display or, more specifically, it toggles the display of groupings within the data in different colours. The exact the way in which the underlying data are split into groups depends on an optional grouping vector (OXtrainGroups/OXtestGroups).

The smootfactor spinner 

This element, comprised of a number in the middle (the smoothfactor) and two buttons by which this number can be increased or decreased, is also specific to the Inspect -> Performance and the Tools -> Compare Networks displays. The number, which can also be edited directly, determines the level of smoothing applied to the data. A setting of 0 or 1 indicates that no smoothing is applied and therefore each datapoint (corresponding to the network's performance in a specific sweep) is displayed exactly as logged. This is a sensible setting when inspection test or verify performance. With regard to the much larger amount of data that arises from training, however, it often makes sense to smooth the (error/correct) curve in order to see the general tendency. A smoothfactor of 4, for example, means that the average over groups of four sweeps is displayed instead of the original data points.

The labels button 

In the Tools -> PCA display, this button toggles whether pattern labels are displayed within the scatterplot or not. In the Tools -> Cluster Analysis display this button switches between two different kinds of labels that you might want to see, namely annotation the cluster plot with either input labels or target labels.

The n clusters spinner 

This element, comprised of a number in the middle (number of clusters to show) and two buttons by which this number can be increased or decreased, is specific to the Tools -> PCA display. The number of clusters, which can also be edited directly, determines the coloring of the scatter plot. If, for example, this number is set to 2, OXlearn will calculate

the two groups of datapoints that are most distant from one another and indicate the membership of each individual point to one or the other cluster by using two different colors.

The change size button 

This button is specific to the Set-up -> Train Patterns/Test Patterns window. Pressing it will raise a pop-up window which lets you choose the new dimensions of your input and target patterns. Note that new simulations always start out with 1x1 patterns and you need to adjust the dimensions to accommodate the specific requirements of your simulation before you can start to enter values.

The graphical edit mode button 

This button is specific to the Set-up -> Train Patterns/Test Patterns window. When toggled, a single click within one of the graphs not only displays additional information with regard to the datapoint you have clicked on (same as the datatip), but it also allows you to enter a new value for this datapoint. Confirm by pressing return. Note that this functionality is also accessible by right clicking on the graph.

The edit in table button 

This button is specific to the Set-up -> Train Patterns/Test Patterns window. Pressing it will open the relevant OXparams (input, output, labels and groupings) in MATLAB's native array editor. Use the tabs at the bottom to switch between OXparams, also note the dock controls in the upper right hand corner of the array editor which let you manage the grouping and display of several variables at once. The functionality of the edit in table button is similar to double clicking on the respective OXparams in the workspace. You can edit all values directly in this table based format. Note that strings (e.g. when editing labels) need to be enclosed in single quotes. It is not

The edit labels button 

This button is specific to the Set-up -> Train Patterns/Test Patterns window. Pressing it will raise a pop-up window which lets you edit pattern labels – after having queried which labels you want to edit. Please make sure that the number of labels and the number of patterns match. Note that this functionality is also available by right clicking on the labels (on the y-axes) of a graph.

The edit groups button 

This button is specific to the Set-up -> Train Patterns/Test Patterns window. Pressing it will raise a pop-up window which lets you edit the grouping vector. Please make sure that the grouping vector has one entry for every pattern. The position of the entry codes for the group membership of the pattern in the same position, the identity if the entry will be taken as a label for the group. Thus, if you enter 'group A' in the first, sixth and seventh position of the group vector, patterns one, six and seven will be deemed to belong to 'group A'.

The File menu

The options under the File menu let you import, export or clear (selected) variables to/from the workspace. If you import from data formats other than OXlearn's native '.mat' simulation files, you might have to ensure proper naming of the parameters, either during the import or in the workspace (choose 'rename' from the context menu after right-click).

File -> New simulation *(shortcut: CTRL-N)*

Choosing this option will erase all existing variables from the workspace, thus creating a clean slate for a novel simulation. Please assign a name to your new simulation (File -> Save Simulation As). If your simulation is still called "new Simulation" when training the network, you will be prompted to assign a name.

File -> Load Simulation *(shortcut: CTRL-L)*

Choosing this option will open a file browser that enables you to select a '.mat' file from anywhere on your computer. Once you confirm your selection, all previously existing variables will be deleted and the contents of the chosen file will be loaded into the workspace instead. OXlearn will not check the contents of the loaded file.

File -> Save Simulation *(shortcut: CTRL-S)*

Choosing this option will save the current status of your simulation to the current filename and location. To change the filename and/or location please choose File -> Save Simulation As – this dialogue will also open automatically if filename and location have not been determined yet.

File -> Save Simulation As *(shortcut: CTRL-Z)*

Choosing this option will raise a dialogue that allows you to determine the location in which you want the current simulation to be saved and the name under which you want to save it. All OXlearn simulation files should have the extension '.mat' which indicates MATLAB's native data format. Use this option to change the name and/or location of the current simulation.

File -> Dump Simulation As *(shortcut: CTRL-D)*

Choosing this option will raise a dialogue that allows you to determine the location in which you want the current simulation to be saved and the name under which you want to save it. In contrast to Save Simulation As, however, the current simulation will remain open and unchanged. Use this option to create safety copies of the current state of the simulation.

File -> Import Selected

Choosing this option will invoke the MATLAB import wizard. The import wizard allows you to import data from many standard formats (e.g. from text files, old t-learn projects or excel worksheets, but also normal .mat files). All selected variables (untick the ones you don't want to import) will be loaded into the workspace with the given names and values, which you can inspect with the import wizard's preview function. Variables already existing in the workspace will remain unchanged, except if your imported

variables have the same name – in which case you will be asked to confirm that they should be overwritten. You can import variables with whichever names you like, but OXlearn will only recognize OXparams when they are named appropriately. You can rename variables by right-clicking on them, either within the import wizard or, after import, in the MATLAB workspace browser.

While it is easy to import multiple OXparams from other '.mat' simulation files (e.g. all the set-up parameters for testing), you should usually only import one variable at a time from other file formats (e.g. from text files or Excel worksheets).

File -> Export Selected

Choosing this option will raise a dialogue that lists all the currently existing variables from which you can select the ones you want to export (use the SHIFT and CTRL keys to select multiple items). Once you have confirmed your selection, another dialogue will let you choose a filename and location under which you want the selected variables to be saved. Note that the drop-down menu at the bottom gives a choice between three different formats in which the selected parameters may be saved: .mat, .txt, or .xls (OXparams will get distributed to several appropriately labelled worksheets).

File -> Clear Selected

Choosing this option will raise a list of all currently existing variables from which you can choose the ones you want to be deleted from the workspace (use the SHIFT and CTRL keys to select multiple items).

File -> Reset Simulation *(shortcut: CTRL-R)*

Choosing this option will clear all performance OXparams, including all weights. Use this option to avoid confusion when changing the set-up of a simulation that already has been trained/verified/tested.

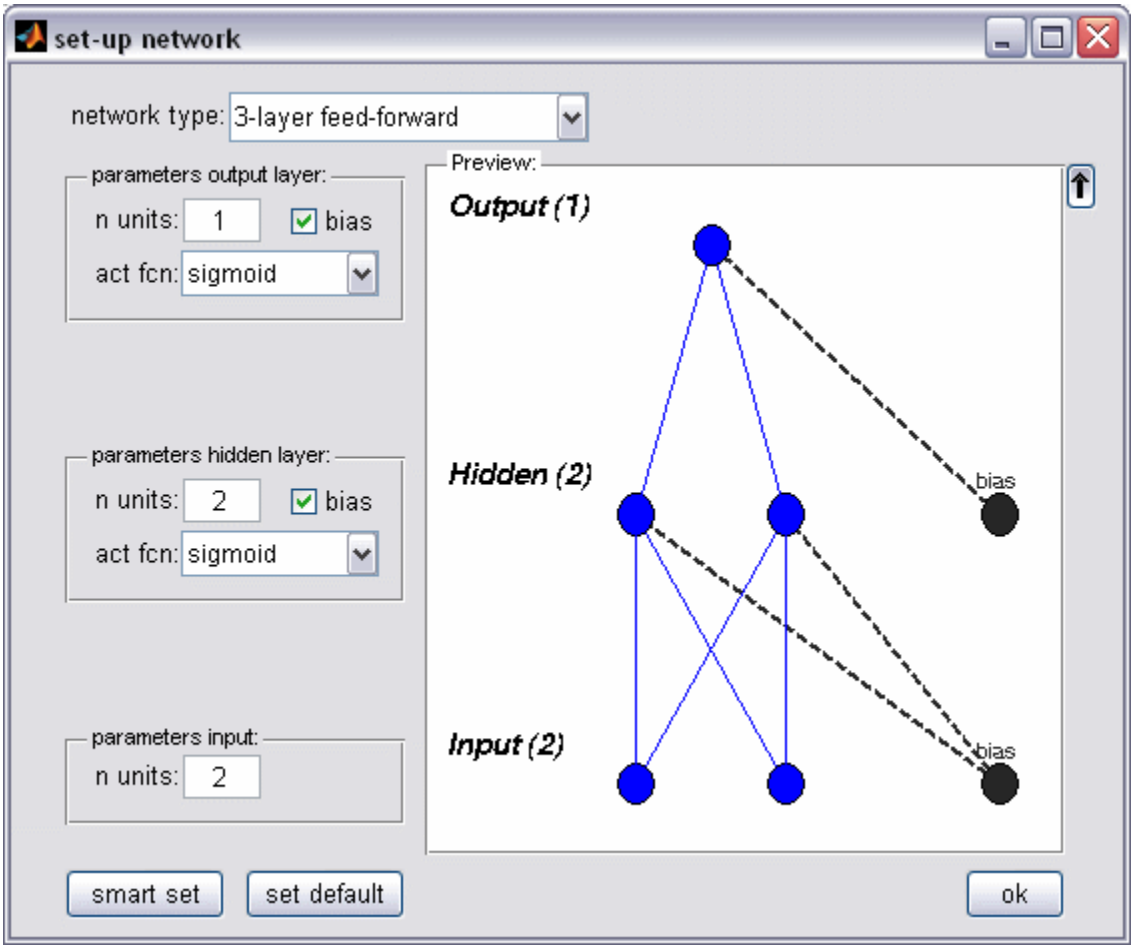
File -> Quit OXlearn *(shortcut: CTRL-Q)*

Choosing this option will close the OXlearn interface, all existing data will be destroyed. Please make sure you have saved you simulation, if appropriate.

The Set-up menu

The options under this menu let you create and manipulate all the OXparams needed to fully determine the set-up of a given simulation – you might also sometimes want to use these windows just to inspect the current values, e.g. to find out which learning algorithm has been used or what the learning rate was in a given simulation. The values shown in the diverse control elements (text boxes, dropdown menus, etc.) will be determined by the values of the corresponding OXparams in the workspace. Similarly, when you change any of these values on the interface, this change will be reflected instantly in the workspace. All corresponding OXparams that do not exist when a set-up window is opened will be created in the workspace and default values will be assigned. If, for example, you click on Set-up -> Network and there is no variable with the name 'OXnetworkType' in the workspace, such a variable will be created with the default value of "2-layer feed-forward". If you now choose "SRN" from the drop-down menu, you can see that the value in the workspace has changed as well.

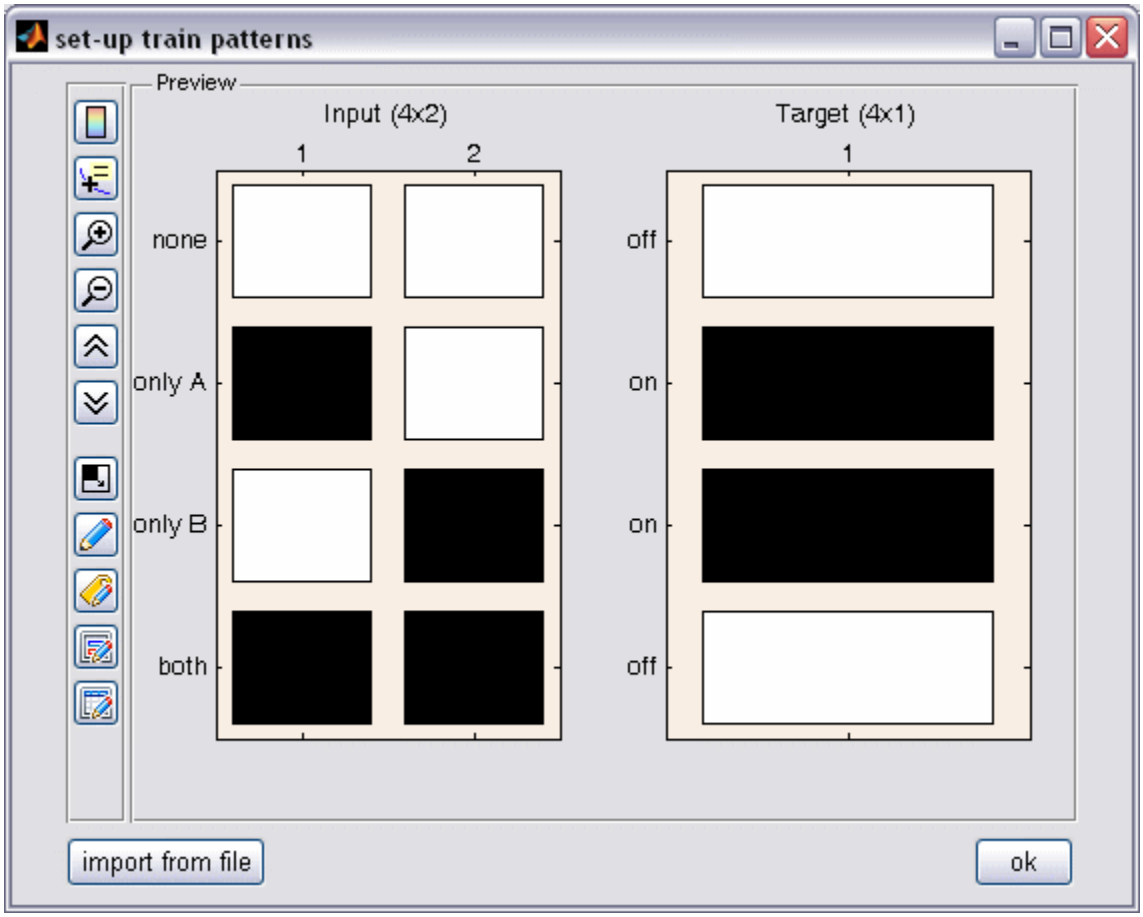
Set-up -> Network



Use this option to determine the *architecture* of the network you want to employ. The interface should be self-explanatory, use the preview at the right hand side to inspect the current architecture. Note that neural networks, traditionally, are displayed with the input at the bottom because they were initially associated with a 'bottom-up' approach to cognition. You can change the orientation by pressing the arrow button to the right of the display if you prefer the input to be on top (more of a flow chart way of looking at things). Note also that individual units and connections will only be shown in small networks. When the layer size exceeds 15 units, a solid slab (for the layer) and a thick arrow, indicating full all-to-all connectivity, will be drawn instead.

The drop-down menu at the top of the figure gives you a choice between several well known network architectures. The value chosen here (OXnetworkType) might also have an impact on other OXparams – everything related to the hidden layer, for example, becomes obsolete with a 2-layer architecture. The other control elements in this window govern, for each required layer, the number of units (OXnI, OXnH, OXnO), whether a bias should be included or not (OXbH, OXbO) and the activation function used (OXactFcnH, OXactFcnO). The smart set button adjusts the number of input and output units to dimensions of the training patterns (s. above), the typical set button initializes all parameters to typical values for this type of network (note that the weights initialisation parameters (OXwInitMearn, OXwInitRange) are influenced as well).

Set-up -> Train Patterns

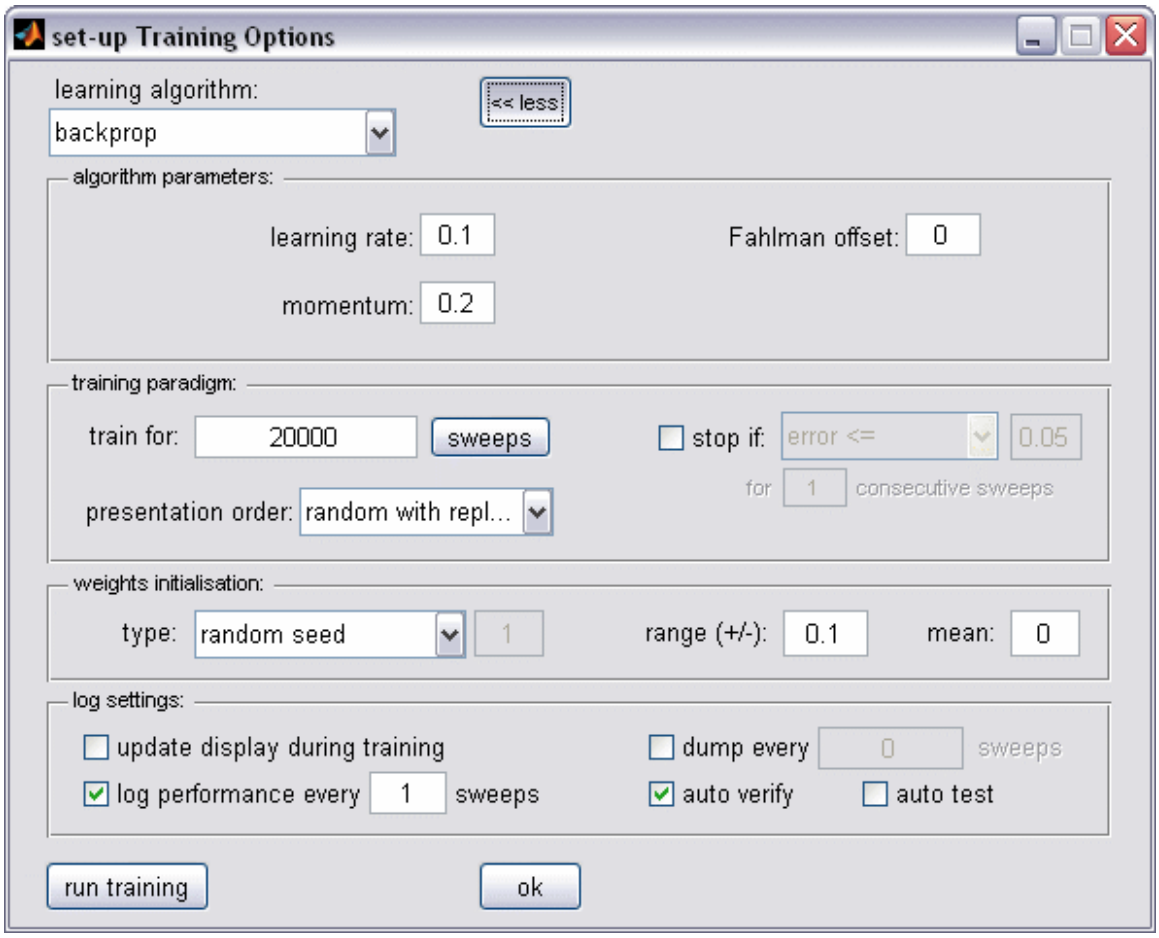


Use this option to determine the exact nature of the stimuli you want the network to deal with, the network's *environment*. At the very least, this requires a matrix of numbers representing the network's input. Each row of this matrix stands for an individual input pattern or stimulus, where the first value represents the activation of the first input unit, the second value holds the activation value of the second input unit, and so on. Evidently, the number of columns in the input patterns and the number of units in the input layer of the network should correspond. How many rows this matrix has determines the number of different input patterns or stimuli the network will have to deal with. As per MATLAB convention, the first dimension is downwards, the second one across (rightwards). Therefore we can say that the input patterns (OXinput) are represented by a n x m matrix of numbers, where n (rows) is the number of patterns and m (columns) is the number of input units.

Many neural network models use supervised learning algorithms which basically means that we need, for each input pattern, a corresponding target pattern which specifies the intended activation in the output layer. Thus we have an additional n x m matrix (OXtarget) where n should correspond to n (number of patterns) in the input, and m equals the number of units in the output layer. Because of the one-to-one correspondence between input and target patterns, OXlearn displays them side by side.

The set-up window enables manipulation of these two matrixes in several, hopefully intuitive ways (see the descriptions of the graphics buttons for details). You can also set-up labels for your input (OXinputLabels) and target (OXtargetLabels) patterns – again, the number of labels and the number of patterns should correspond. Setting custom labels is optional, but it helps keeping track of your simulation. Another optional parameter, again with n (number of patterns) elements, is a grouping vector (OXgroups). With its help you can define groups within your patterns and later differentiate the network's behaviour with regard to these groups of patterns.

Set-up -> Training Options *(shortcut: CTRL-O)*



Use this option to determine the exact way in which your network will be trained, the training *paradigm*. This includes the choice of a learning algorithm (OXlearningAlgorithm) and the associated parameters (e.g. OXlr, OXmomentum, and OXfahlmannOffset for the classical backpropagation algorithm), but also parameters detailing the length of training (OXmaxSweeps), a possible early stopping criterion (OXearlyStoppingFlag, OXstopCritType, OXstopCritValue, OXstopWindow) and the order in which individual patterns are presented during training (OXpresentationOrder). Furthermore, you can specify the details of the weights initialisation (OXwInitType, OXwInitSeed, OXwInitMean, OXwInitRange) and influence which information will be logged (OXlogTrainPerfFlag, OXlogTrainPerfInterval), saved (OXdumpFlag, OXdumpEveryNSweeps, OXautoVerify, OXautoTest) and displayed during training

(OXliveUpdateFlag). With the exception of the learning rate, the momentum and the training length, however, you will usually be fine with the default values for most of these parameters – which is also the reason why some of them will only be displayed when you click on the >> more button. Note as well that some of the parameters determine whether others are enabled or not. For example, it obviously is not necessary to specify an interval for performance logging when you have decided not to log training performance at all.

Set-up -> Test Patterns

Use this option to determine the exact nature of the stimuli you want to test the network with. The test patterns (OXtestInput, OXtestTarget, OXtestInputLabels, OXtestTargetLabels, OXtestGroups) are set-up in exactly the same way as the training patterns, but will be used for testing where the train patterns are used for training and verifying.

Set-up -> Testing Options

Use this option to determine which data will be logged during testing (OXtestLogOutputActFlag, OXtestLogHiddenActFlag). It is rarely necessary to change these parameters.

The Run menu

The options under the run menu let you train network(s), verify network(s) and test network(s). Obviously, a network can only be trained or tested when the simulation is set-up appropriately, please use the Inspect -> Simulation display to determine the status of you simulation. The precondition for testing is that all parts of the simulation are indicated to be set-up correctly. Training and verifying can do without the parts that relate to testing only (Test Patterns, Test Options). Note that all the options under run might clear or overwrite previous results (performance OXparams), to avoid this make sure to save your simulation under a different name before training the next one. Note also that the Set-up -> Train Options window includes two tickboxes (auto verify, auto test) which, when ticked, automatically include one verification and/or one test run at the end of training (and before dumping any weights, in case the dump option is chosen).

Run -> Train Network *(shortcut: CTRL-T)*

Choosing this option will result in the network being trained. Essentially, the different input patterns will be presented to a network with the given architecture, one at a time, in the chosen order of presentation. For each pattern, the activation will be propagated through the net with its (usually) initially random weight configuration. The output produced by the net will be compared to the target for this pattern, and the weights will be adjusted in accordance with the chosen learning algorithm and its associated parameters. Repeatedly doing so will result in a weights configuration that has optimally adapted to the processed stimuli. These weights (e.g. OXweightsInputToHidden, OXweightsHiddenToOutput, etc.) contain all the acquired 'knowledge' of the network, and they will be sent back to the workspace at the end of training. Training ends when either the maximum number of sweeps is reached or when the early stopping criterion is satisfied.

There are three more things that are logged during the process of training: (1) the order in which individual patterns were presented (OXtrainOrderLog), (2) the mean square error for each pattern (OXtrainError), and (3) whether the network's performance was correct or not (OXtrainCorrect). Correctness is determined by evaluating the network's output against a relatively conservative correctness criterion, namely that the activation of none of the output units deviates by more than 0.1 from its target value. As per default, the three values mentioned above are logged for every single sweep during training – thus producing a large amount of data which allows you to inspect the networks training performance in detail. In large simulations (where training goes on for many sweeps) it is useful to increase the logging interval. If, for example, the interval is set to 10, only every tenth sweep will be logged, thus loosing some (usually negligible) detail but, at the same time, reducing the amount of data in you computer's memory (and in the simulation file) by a magnitude.

Apart from giving these performance OXparams back to the current workspace, OXlearn will also automatically save a complete copy of the state of your simulation at the end of training. You will find this file in the same folder that your current simulation resides in. The file will have the same name as your current simulation with a suffix of the form '_sw<number of sweeps>' attached. Thus if your simulation is called 'MySim' and you have trained the network for 3333 sweeps, this file will be named 'MySim_sw3333.mat'. Your current 'MySim' file (having, at this point, the same content as 'MySim_sw3333') will remain open for you to continue working on it.

Run -> Train Several Networks

Choosing this option will open a dialog in which you can specify how many instances you want to train and subsequently repeat the process of training the current network for the given number of times. At the end of each training run, the simulation will be saved under the original name with a suffix of '_run(<N>)', where N is a running index over instances. The simulation file including the results from the last run will remain open under the original name.

This option is useful if you want to explore a network's dependency on the initial weights configuration. For example, you might want to compare the final performance of 10 networks which only differ in terms of the random weights initialisation (and/or the possibly random order of presenting patterns during training). Use the Tools -> Compare Networks display after having trained multiple instances to investigate such comparisons.

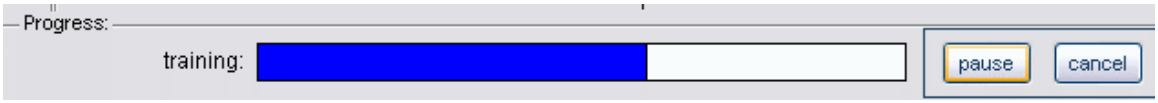
Run -> Resume Training

Choosing this option will use the current weights and continue training from the current point. It is, obviously, a precondition that suitable weights do exist. If the previous training process has been terminated prematurely, training will continue until the original maximum number of sweeps is reached (or until the early stopping criterion, if enabled, is satisfied). If the previous training run has reached the maximum number of sweeps, you will be queried as to how much longer you want to train the network.

Please be cautious when changing any set-up parameters before resuming training. While it sometimes might make sense to, for example, reducing the learning rate before training for a final couple of hundred sweeps, OXlearn will not log this change and you might later be misled to believe that the network was entirely trained on the lower learning rate.

Other changes can induce problems with displays (e.g. changing the logging interval) or training function (e.g. changing the network architecture).

Displaying the training process



When a network is being trained, a blue progress bar will appear in the bottom part of the current display. At the right hand side, you will find two buttons labelled 'pause' and 'cancel'. The latter, naturally, aborts the training process and prevents weights and training performance parameters to be given back to the workspace (alternatively you can press '*c*' on your keyboard). Pressing the pause button (or '*p*' on your keyboard) will freeze the training process and provide five additional buttons that let you interact with the paused simulation. These buttons are:

 resume (or press '*r*'): quit the paused mode and continue training

 skip sweep (or press '*1*'): train for one more sweep, then pause again. You can also press another number (<=9) to train for so many more sweeps.

 skip epoch (or press '*e*'): train for one more epoch (= the number of patterns in the input), then pause again.

 stop (or press '*s*'): stop training at this point, give results (current weights, training performance up to that point) back to the workspace

 cancel (or press '*c*'): abort training, do not give results back.

The functions of these buttons are only useful when the displays are updated during training (this is controlled by a tickbox in the training options window). There exists a good reason for unticking this box: your simulation will run a fair amount faster when the displays do not have to be updated during training – progress will still be indicated by the progress bar and you can, of course, inspect training performance in detail after training is finished. Updating the display, on the other hand, enables you to monitor online how well your network is doing and, using the buttons mentioned above, you may inspect snapshots of the network's development during training. The three displays that may be of interest with respect to this development are the Weights, Pattern, and Performance displays, all found under the Inspect menu.

Run -> Verify Network *(shortcut: CTRL-V)*

Choosing this option will present the current simulation with each of the train patterns once, in sequential order. The weights are not adjusted any more (they are 'frozen'), the verify option thus essentially tests the networks current performance on the patterns used for training it. Because the requirements of training the network and running verify are the same, auto verify (on the training options window) is on per default. Verify returns two performance OXparams to the workspace, containing the output layer activations (OXverifyOutput) and the hidden layer activations (OXverifyHidden) produced in

response to each of the input patterns. This is not usually an amount of data to challenge the capabilities of modern computers, it is thus recommended to leave the auto verify option enabled.

Run -> Verify Several Networks

Choosing this option will run the verification test on several simulation files, you will be prompted to indicate these files. You will only need this option when you have produced several simulations (or dumps) with the auto verify option disabled. See Inspect -> Test Several Networks for more details.

Run -> Test Network *(shortcut: CTRL-K)*

Choosing this option will present the current simulation with each of the test patterns once, in sequential order. The test patterns usually consist of novel patterns that the network has not seen during training, and the outcome of the test can thus inform you about the network's ability to generalize. It is a precondition that the test patterns and other test related parameters are appropriately set. During testing, weights are not adjusted, they are 'frozen'. The test returns two performance OXparams to the workspace, containing the output layer activations (OXtestOutput) and the hidden layer activations (OXtestHidden) produced in response to each of the test input patterns. It is not, strictly speaking, necessary to define test target patterns, but evaluation of the networks performance is often easier when you give them. If the test related parameters are set up prior to training, you can make OXlearn run a test automatically by checking the box next to auto test in the training options box. This is especially useful when you train multiple instances or dump intermediate states, as in each of these cases the auto test (and, possibly, auto verification) will be performed before a run or dump is saved.

Run -> Test Several Networks

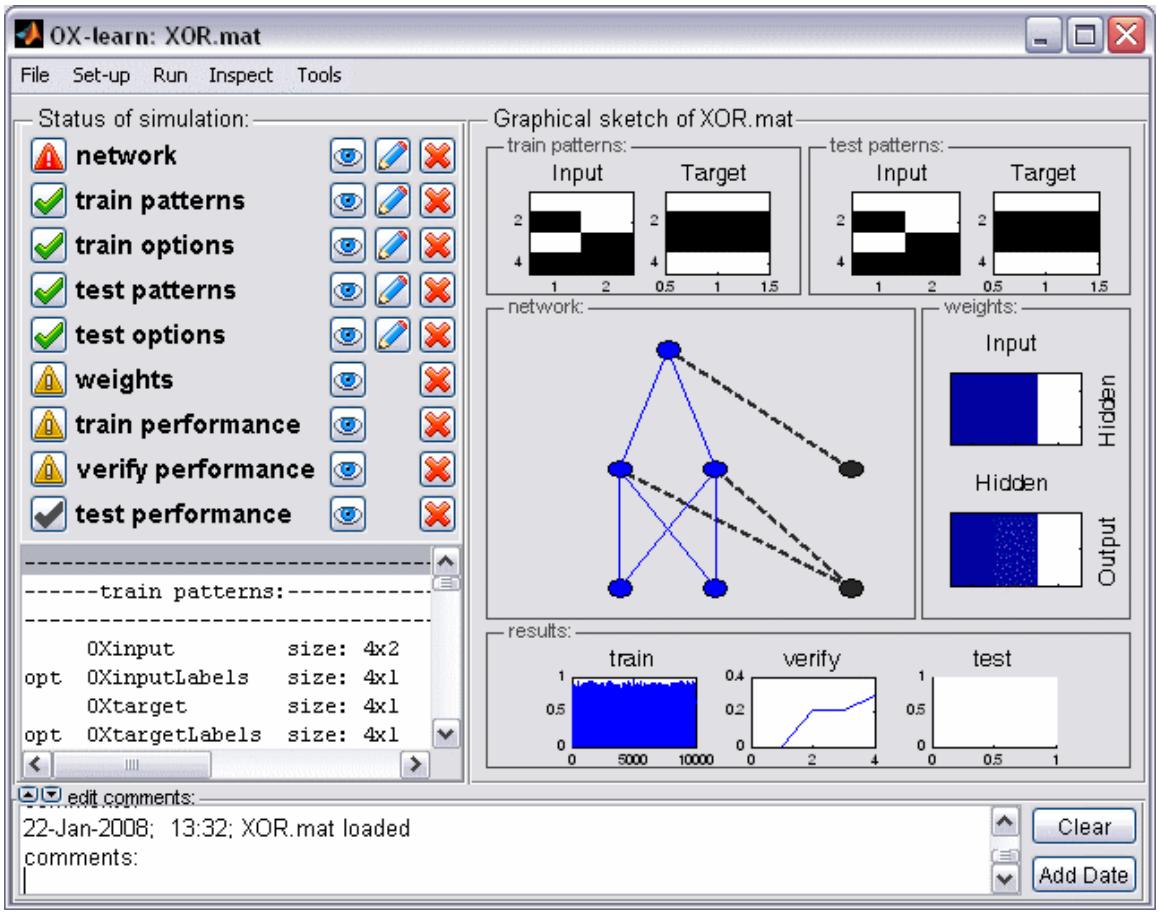
Choosing this option will run the test defined in the currently opened simulation on several simulation files, you will be prompted to indicate these files. You will only need this option when you have produced several simulations (or dumps) with the auto test option disabled, or when you want a novel test to be performed on several runs or dumps. Note that each simulation file can only hold one set of test results, any existing test results will thus be overwritten by the new test (duplicate the simulation files before running the novel test to prevent that).

When testing multiple simulation files, the currently opened simulation will serve as a reference point. Basically, any external simulation that you have selected for testing will be compared against the current one in a few key points that guarantee compatibility, e.g. that the number of input and output units match. Provided this is the case, OXlearn will perform the test defined in the current simulation with the weights defined in the external simulation file, to which the outcome will be saved as well. Only after all the external files have been tested in this way will the test be performed on the current simulation, which remains open.

The Inspect menu

The Inspect menu offers convenient ways of visualizing data within the simulation, i.e. OXparams or parts thereof. As an alternative it is always possible to inspect data directly in the MATLAB workspace browser and array editor.

Inspect -> Simulation *(shortcut: CTRL-I)*



This display provides a graphical overview of the current status of the simulation. Three main parts are distinguishable: the status panel, the graphical sketch panel and the comments panel.

The comments panel allows you to include descriptive comments into the simulation file. This is often helpful in reminding yourself what you have done or attempted to do in a specific simulation when you come back to it at a later time. Nothing is more vexing than having done a simulation some months ago and then not being able to determine which was the final version. As an aid in this respect, OXlearn will automatically add a time stamp each time you open a simulation. However, you can delete/add/edit the comments in whichever way you like without causing any problems. Note that the small triangle buttons at the left hand side allow you to enlarge the comments panel.

The graphical sketch panel summarizes your current simulation graphically without going into details. This is intended to provide, at a glance, information such as whether target patterns have been specified, if the network has been trained already or which type of

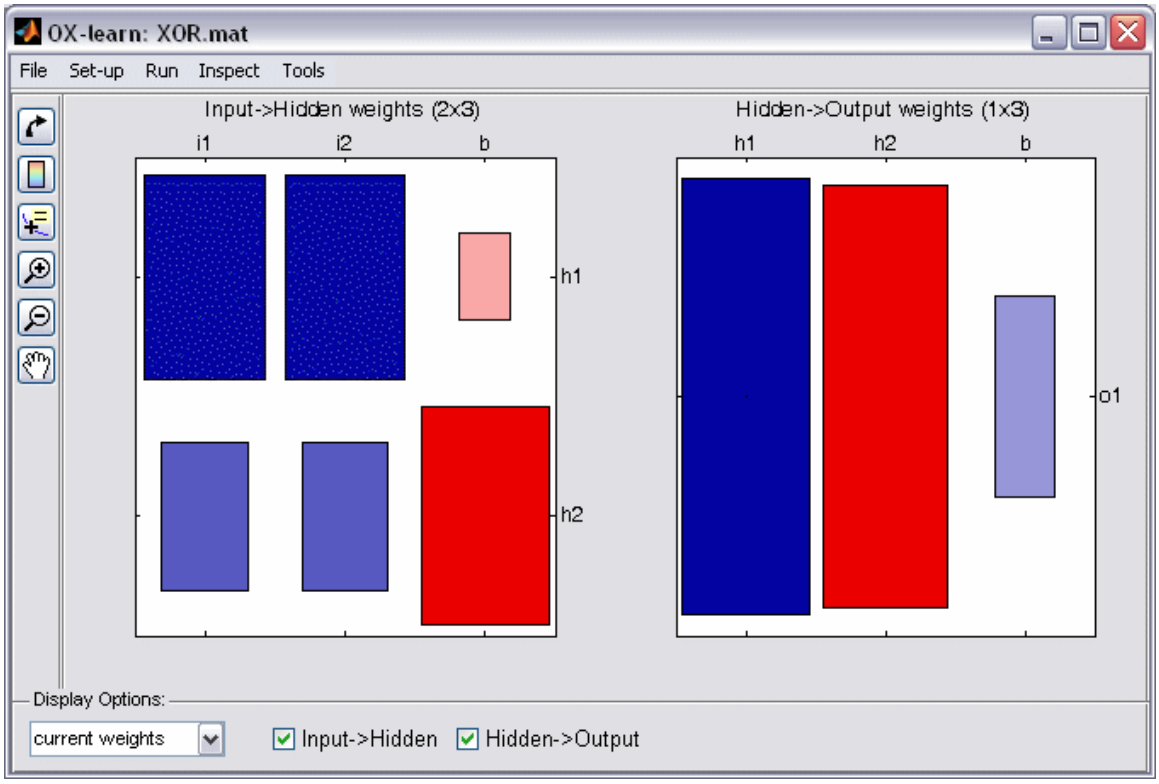
network architecture is currently chosen. The dedicated displays in the 'Inspect' menu provide a more detailed view.

The status panel tells you which parts of the simulation are set-up correctly and which are deficient. The nine labels at the top refer to different components that make up a simulation, and each of these labels summarises over a number of OXparams. The up-most five labels correspond to information that can be manipulated in the 'Set-up' windows (= set-up OXparams), whereas the last three represent the performance OXparams for training, verifying and testing. The sixth element called 'weights' occupies a somewhat intermediate position, because it is dependent on the weight initialisation parameters (which can be manipulated under Set-up -> Training Options, >> more) but will only be assigned values once training is initialized. Those initial values are subsequently adjusted by the training process and thus, once they are given back to the workspace, resemble to other performance parameters in that they are an outcome of the training process.

Green ticks to the left hand side of the labels indicate a satisfactory set-up, a red attention sign means that a problem has been detected with at least one of the OXparams summarized under that label. Clicking on the status indicator (tick or attention sign) will lead the listbox in the lower half of the status panel to show more detailed information regarding the relevant OXparams and the detected problem. The first column indicates problems with exclamation marks ('opt' indicates an optional parameter), the name of the parameter is given in the second column, and more detailed information regarding the parameter or the problem with it can be found in the third column. To address the problems found, the three buttons to the right hand side of each label provide shortcuts to manipulate or display the relevant part of the simulation. Pressing the eye button will switch to the dedicated display (if given for this part) usually found under the 'Inspect' menu. The pen button will raise the corresponding Set-up window (if given) so you can manipulate the erroneous parameters. Note that simply raising the window and closing it again will often solve the problem because all non-existing parameters for this part of the simulation will be created automatically. The red cross button, finally, will clear all OXparams summarised under the label. This is most useful for clearing the performance OXparams (the last four elements). Non-existent or cleared performance OXparams are indicated by a gray tick – the preferable status before training a network. A yellow attention sign at this place indicates that the current set-up might be different to the one that has resulted in those performance OXparams. It is possible to simply train the network again, thus rectifying the inconsistency. To avoid confusion, however, it is recommended to reset the simulation prior to changing the set-up.

A simulation is ready to run when the first five status buttons show a green tick - strictly speaking, the 4th and 5th ('Test Patterns' and 'Testing Options') element is only required for testing, not for training/verifying.

Inspect -> Weights *(shortcut: CTRL-W)*



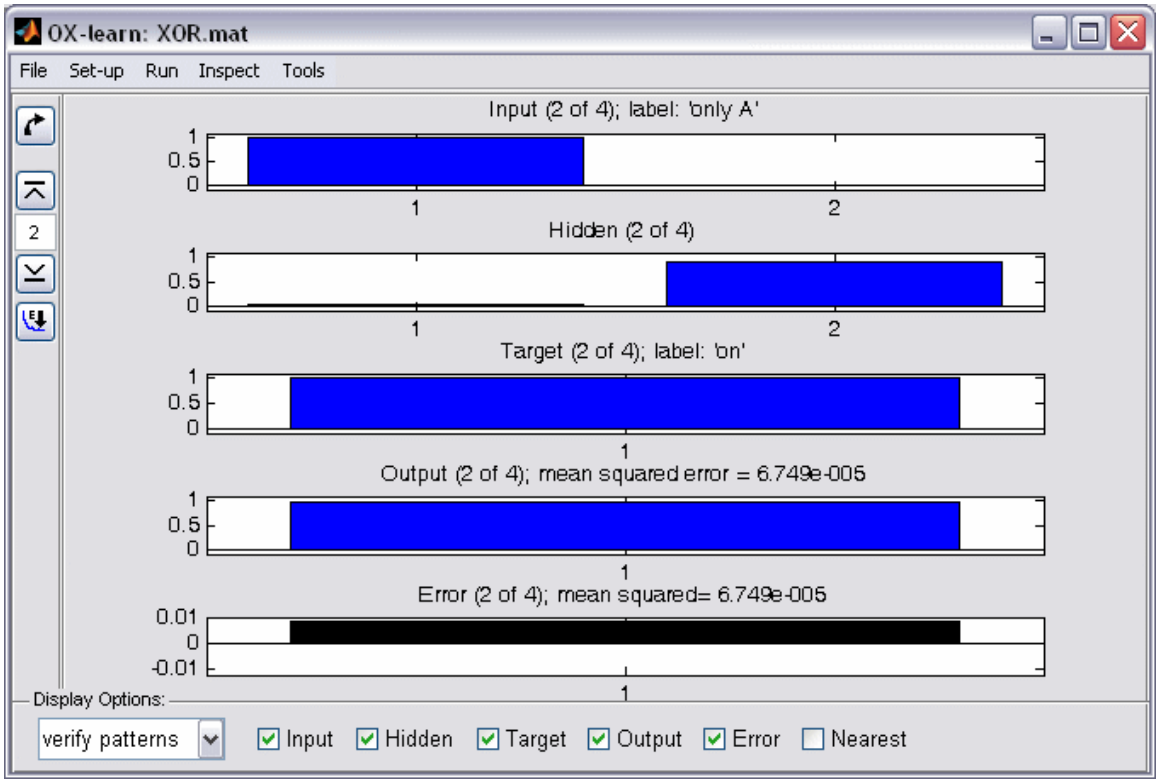
This display shows the current weights configuration of the network in so-called 'Hinton diagrams'. Each connection weight is displayed as a coloured box at a x/y position that indicates from which unit (x-axes at the top) to which unit (y-axes at the right) the corresponding connection leads. Each weight's numerical value is coded in its colour (negative values in red, positive values in blue) and its size (large absolute values -> big box; values close to zero -> small box). Similar to most other displays you can manipulate the subset of data shown in the graphs (zoom, pan, etc.) or gather more detailed information about individual data points with the datatip.

It is not unusual that no weights configuration is displayed prior to having trained a network. The weights matrix will only be initialized at the beginning of training. You can, however, get a visual impression of the weights initialisation by choosing "weights initialisation" from the drop down menu in the options panel. Doing so repeatedly will reinitialize the weights each time. Note, however, that weights will also be reinitialised anew when training starts. Choose a "Seed Nr:" in training options if you want to ensure that the initial weights shown correspond exactly to the ones the network uses when being trained.

Inspect -> Patterns *(shortcut: CTRL-P)*

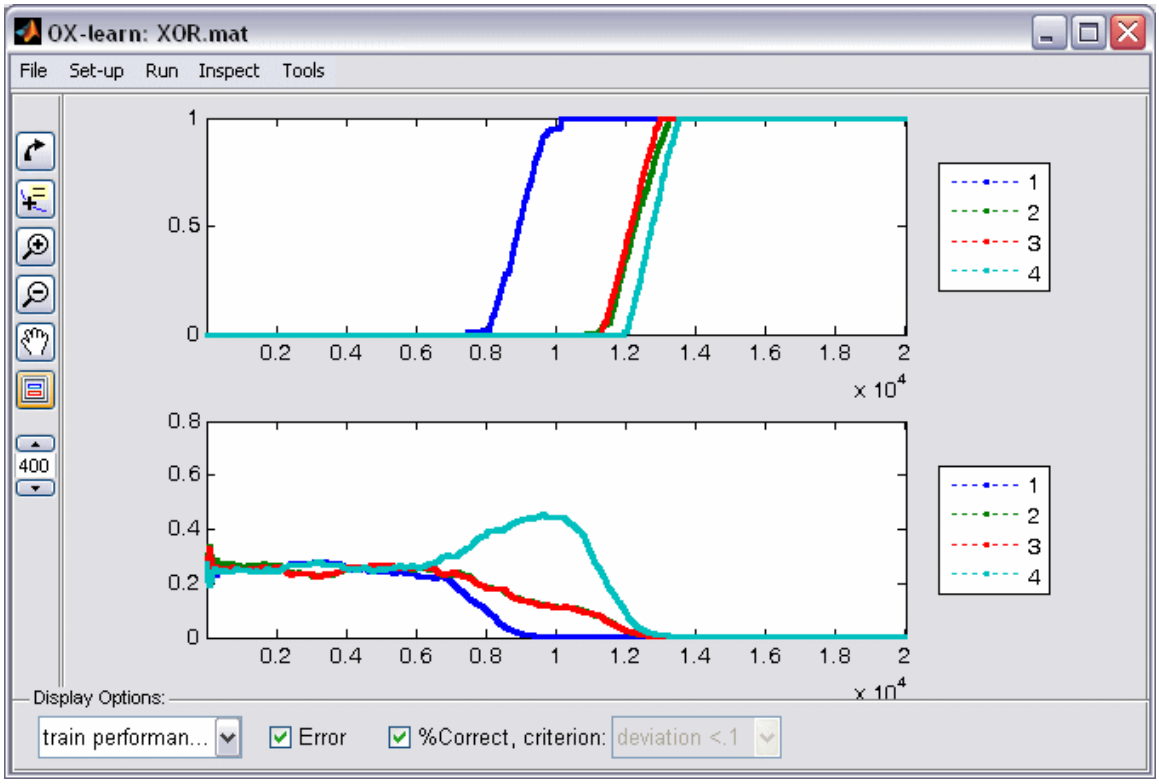
This display shows colour coded images of all patterns (one epoch) used during training, verifying or testing. In addition to the input and target patterns (similar to the visualisation during set up), hidden layer activation, output activation, the resulting error and the pattern nearest in Euclidean space to the obtained output can be displayed when looking at verify or test data. This display enables inspection of the whole epoch (all patterns) at a glance, e.g. in order to identify problematic stimuli. Note that the Inspect -> Activations display provides a similar view on a per pattern basis.

Inspect -> Activations *(shortcut: CTRL-A)*



This display shows network activation and error information for individual patterns/sweeps, thus allowing, e.g., for a direct comparison of the produced output activation of each unit and its intended target activation. Use the arrow buttons on the left to skip through patterns/sweeps, in sequential order, i.e. corresponding to the order in the Inspect -> Patterns display. If the "sort by error" button is toggled, the arrow buttons skip to the pattern with the next higher/lower error instead.

Inspect -> Performance *(shortcut: CTRL-E)*

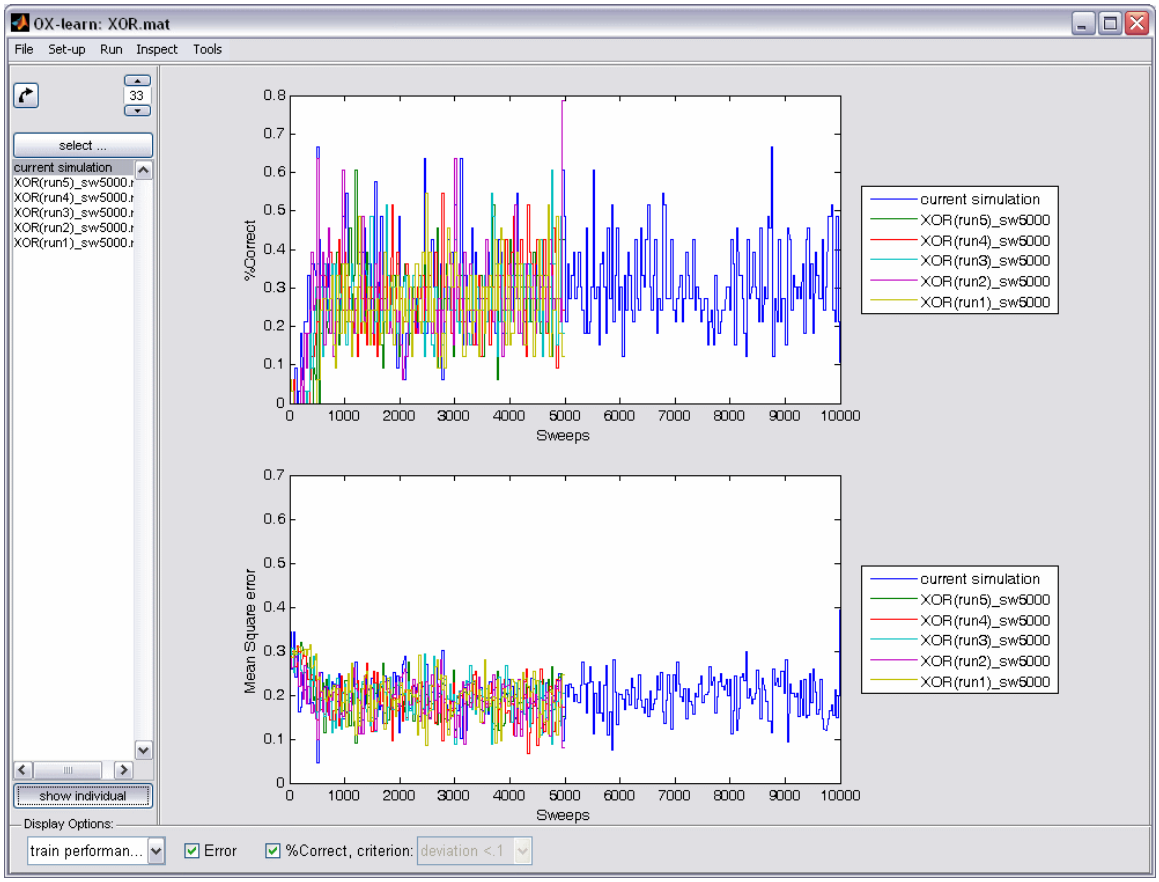


This display shows, for every sweep, the network's performance in terms of mean square error and/or correctness. One datapoint per logged sweep will be displayed if "train performance" is chosen in the options panel, otherwise datapoints will correspond to the patterns in Train or Test Patterns. Recall that the curves can be smoothed and grouped.

The Tools menu

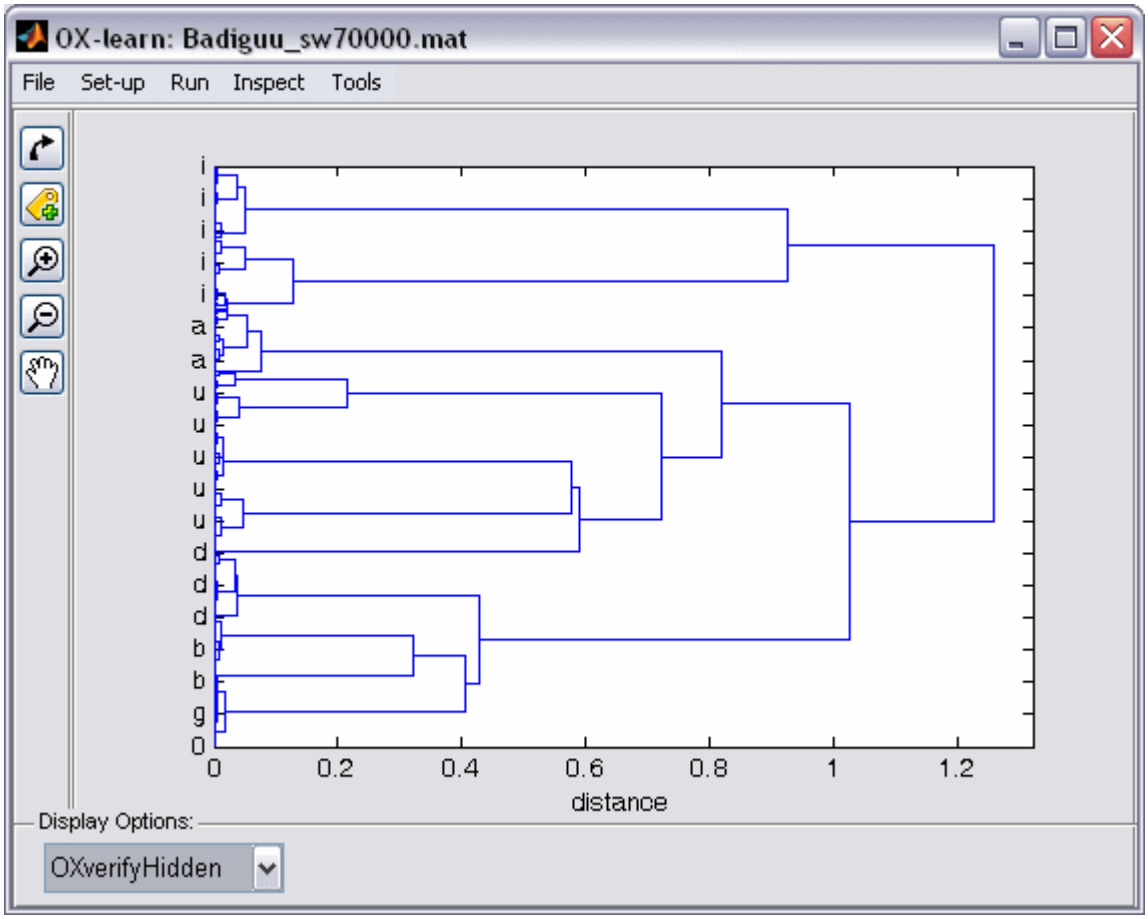
The Tools menu comprises a further set of specialized functions and/or displays.

Tools -> Compare Networks



This tool is meant to enable comparison of the performance of multiple similar networks, or states of one network. For example, if you have dumped network states (including tests) at regular intervals during training you might want to investigate how the networks performance on the test patterns develops over the course of training. Similarly you might want to compare the performance of several networks that differ from one another only in terms of their initial weight values or the learning rate used. To do so, choose several comparable networks from the file selector that is raised by pressing the “select…” button. Similar to testing or verifying multiple networks, the currently loaded simulation (always at the top of the list at the left hand side of the window) will serve as a reference point with which the indicated simulations are compared. Depending on whether you look at train or verify/test performance you can either smooth or group the displayed data, just like in the Inspect -> Performance display. Additionally, the “show individual” button below the list of simulations toggles between displaying data for all the selected simulations and displaying the mean performance plus a measure of the spread of data (+/- one standard deviation, indicated by error bars or dotted lines).

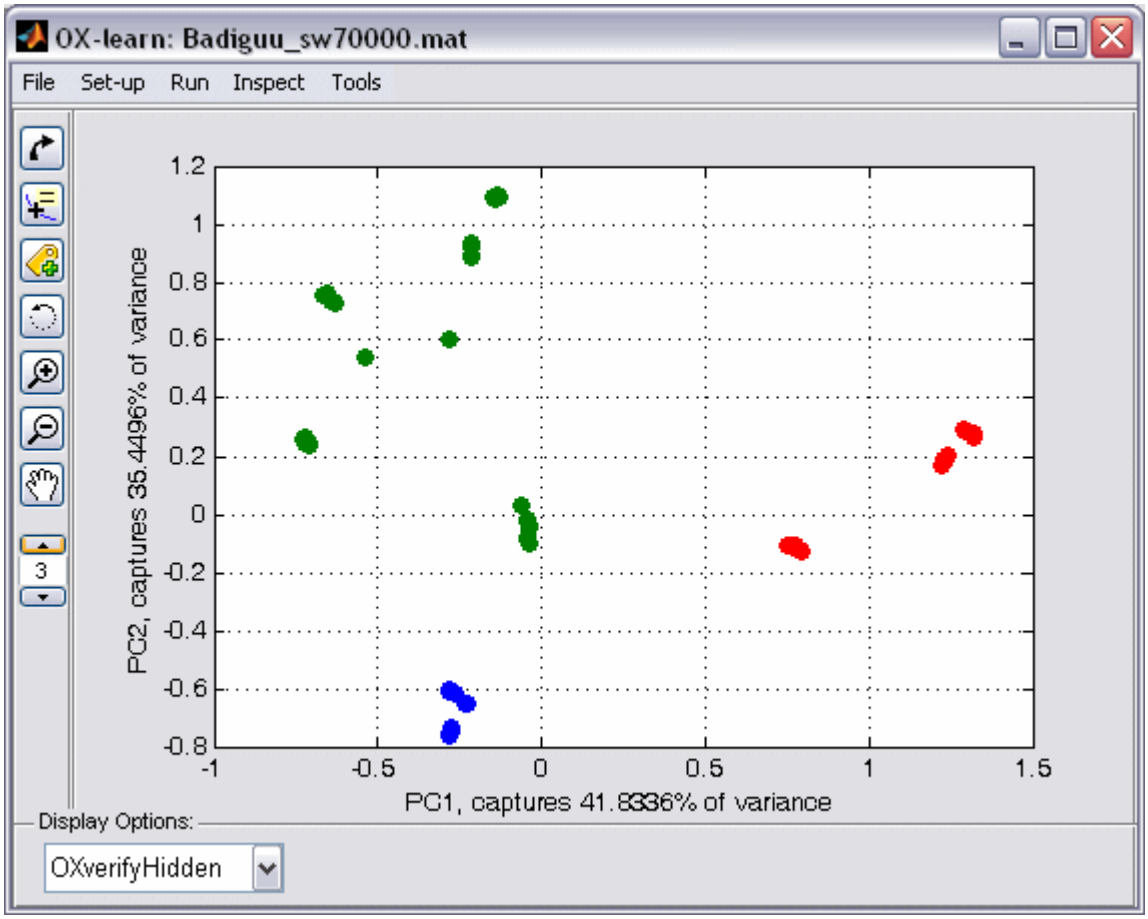
Tools -> Cluster Analysis



Cluster analysis is a technique for visualizing high dimensional data. Essentially the idea is to always group the two elements with the smallest Euclidean distance and subsequently treat them as one item that is located at the midpoint between the two original elements. Iterative application of this rule leads to a cluster tree that conveys an impression on how the items are distributed in some high-dimensional space (e.g. hidden layer activation space). The length of the edges indicates the distance between clusters/items. This technique can be applied to any high dimensional data (i.e. all 2-dimensional matrices of data with columns being interpreted as dimensions and rows as items), but it usually is most useful for analysing the distributed representations that emerge in a trained networks hidden layer. Items/patterns/stimuli that are grouped together are treated as similar by the network.

This tool works only if you have the MATLAB Statistics toolbox installed.

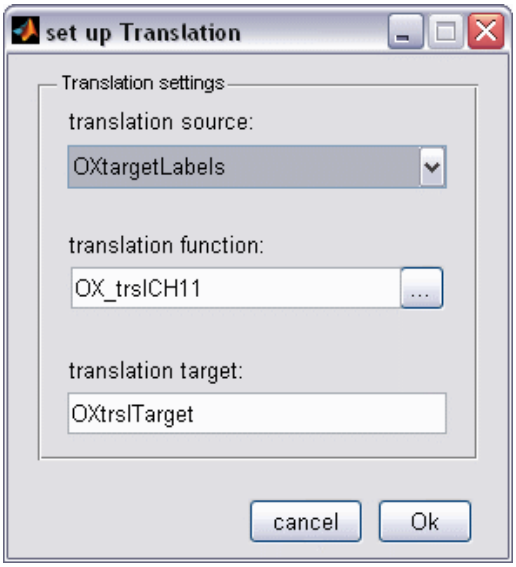
Tools -> PCA



PCA (principle component analysis) is another technique to visualize high dimensional data, this time by dimension reduction. Essentially, a PCA rotates the coordinate system of the underlying high-dimensional such that the first dimension captures the most variance, then the second, and so on. Often it is possible to look only at the first two or three dimensions (or principle components) and still capture the majority of variance or information in the data. The display of this tool shows and possibly labels the underlying data (chosen in the drop down menu at the bottom) in this new coordinate system. You will find the loadings of the first three components in the axes labels. Use the rotation tool to change the viewing angle. The n clusters spinner indicates the n clusters with the highest inter cluster distance by assigning different colours.

This tool works only if you have the MATLAB Statistics toolbox installed.

Tools -> Translation



The translation tool will usually be used during the process of setting up a simulation, for example when you want to transform a couple of words that should serve as the network's input patterns into a numerical representation which can be processed by the network. However, you might also want to translate the network's output back into graphemic representations – to some extent this will happen automatically if you have provided appropriate labels along with the train or test patterns.

Because it is literally impossible to preview the exact translation anyone might want to perform, the translation tool provides a rather high level solution: you can choose which variable within the MATLAB workspace you want the translation to be applied to (translation source), you determine the name that will be given to the outcome of the translation process (translation target) and you choose the translation function that mediates between the content of the source (input) and the content of the target (output). OXlearn provides some inbuilt translation functions, such as mapping letters in the source onto a six-digit binary code that encodes phonological features such as *voiced*, *labial*, *dental*, *etc*[1]. Another, more generic translation function ("OX_trslTlearnStyle.m") will query the user for a left hand side (input) and a right hand side (output) translation table with corresponding rows.

In case none of the provided translation functions suit your needs or you are unclear on how exactly they work it is recommended to have a look at the corresponding .m files. You will find these in the subfolder "TranslationFiles" within the "OXlearn" folder. If you browse to this location using MATLAB's "current directory" editor, a double click will open the file within the MATLAB file editor. The translation functions are heavily commented and even if you have not much experience with programming it should be relatively easy to find out what they are doing or to adjust them to your needs. Please make sure to save any changed files under a different name. As long as you keep your

[1]For details of this specific translation function (called "OX_trslCH11.m") please refer to chapter 11 in "Exercises in Rethinking Innateness" by Plunkett & Elman (1996).

customized translation functions in the same folder ("…\OXlearn\TranslationFiles"), they will be offered as a choice in the drop down menu of the translation tool.

Glossary of neural network terms

activation function: a function that is applied to the sum of a node's incoming activation (= net input). Common choices include '*linear*', which amounts to no change at all (i.e. the node's activation value is set to its net input), *'threshold'*, which means that the node will be 'on' (activation value of 1) if the incoming activation surpasses a specific value (usually 0) and off (activation value of 0) otherwise, and '*sigmoid'*, which can be described as a soft threshold function because the transition from 'off' to 'on' is more gradual, allowing for intermediate values as well. The sigmoid is the typical function of choice for multilayer networks because of these model's requirement for an activation function (at least in the hidden layer(s)) that is (a) non-linear and (b) differentiable.

architecture: the way in which the network is set up, e.g., the number of layers, the number and type of units in each layer, the way the units/layers are connected with each other, if there is a bias, etc.

bias: a node that has no inputs and is always active (activation value of 1). The effect of connecting a bias to a normal node within the network is to provide a constant (independent of the pattern actually processed) bias on the receiving node's propensity to respond or, from a mathematical perspective, a bias induces a lateral shift to the receiving node's activation function. The direction and amount of said shift is directly proportional to the weight that connects the bias with the receiving node, and this weight is adjusted (learned) in the exact same manner as all other weights in the network. In biological terms, the bias can be linked to a neuron's resting activation, in technical terms a bias can help especially in situations in which an output is required although the network has no or little input activation to work with. It is common to connect a bias to all nodes in a network.

catastrophic interference: the most common learning algorithms work on the basis of the individual patterns that are processed in a given sweep. The resulting adjustment of the weights matrix is guaranteed to improve the network's performance for this particular pattern/stimulus. It is possible, however, that the adjustments are detrimental to the processing of *another* pattern. Thus, if the magnitude of adjustments to the current pattern is too large (e.g. due to a large learning rate or because always the same pattern is presented) the network might loose its ability to deal with other patterns. This phenomenon is often termed 'catastrophic interference' or 'catastrophic forgetting', ways to counteract this tendency include: reducing the learning rate, using a momentum, and/or making sure that the different patterns are presented in an interleaved manner rather than in blocks.

epoch: a pass through the entire set of training patterns. If, for example, there are 4 different input/target patterns, an epoch would equal 4 sweeps. OXlearn does not use the concept of epochs, everything is defined in sweeps.

error: the difference between an output unit's activation value and the corresponding target value.

layer: a set of nodes in a network, usually defined by a shared pattern of connectivity. The most basic network architecture has only two layers (input and output), but most models employ a three layer architecture which includes an additional hidden layer.

learning algorithm: the mathematical process in which the network determines how to adjust its weights. In supervised networks, the learning algorithm is aimed at minimizing the error, usually by implementing some form of gradient descend. The result is that the weights are adjusted constantly (usually after each sweep during training) in a direction that would improve the network's response should the same pattern be processed again. Over time and through repeated exposure to the different train patterns, the weights matrix usually settles into a configuration that suits all patterns.

learning rate: the magnitude of weights adjustment. Too small a learning rate can prevent the network from learning at all (at least within a reasonable amount of exposure/sweeps), too large a learning rate can induce catastrophic interference. Determining an appropriate learning rate is not an analytical process, as it depends on many aspects of a network's task and architecture and their interaction. As general rules of thumb, however, smaller networks and networks with little overlap between patterns can afford larger learning rates (> 0.1), while large networks with much redundancy and overlap often use learning rates a small as < 0.001.

momentum: a value that determines the proportion of last sweep's weights adjustment that is added to the current adjustment of weights. As a result, instead of trying to maximally accommodate the current pattern, the weight adjustment reflects a mixture of the current patterns 'demands' and those of other recently processed patterns. In terms of gradient descent, a momentum could be said to add some inertia to the downward path along the error surface, thus preventing, e.g., the current step to go in an entirely different direction than the last. In some circumstances using a momentum can make learning more efficient or stable (see gradient descend). Momentum values > 1 do not usually make sense, because this would mean that the current step has less influence than past ones.

gradient descent: the most common learning algorithms (e.g., backpropagation) determine the necessary weights adjustments by looking at the error that results from the processing of a specific pattern as a function of the weights in the network. Calculating a partial derivative with respect to all the weights in the network thus gives the direction in which the weights must be adjusted in order to reduce the error, i.e., going down the slope of the error function. Depending on the exact set-up of the network, the task processed and the initial value of the weights, however, there is a possibility that this process of incrementally reducing the error for individual patterns will not result in the minimization of the overall error (for all patterns). Analogically speaking, the error surface can be likened to a hilly landscape and if the only directive is going downwards it may be that one ends up in a high valley; if this happens, the network is said to be trapped in a local minimum. Common solutions to prevent this from happening include: retraining the network (because a different set of initial weights corresponds to a different place in the error landscape to start from, which might prevent the network from visiting the problematic region in the error surface), using a momentum (the added inertia can smooth over the local valley), increasing the learning rate (might step over small local valleys), and/or using a different presentation order (can induce a different way down the mountain).

hidden layer: a group of units that mediate between the (externally determined) input units and the output units, which correspond to the network's response. The hidden layer is especially interesting for researchers, because of the emergent, distributed

representations that develop here during the process of training a network. Inspecting/visualizing the hidden layer activations can give valuable insights as to how the network solves (or learns to solve) a task. Determining an appropriate number of hidden units is difficult, as it depends on the complexity of the task. More hidden units grant the network more processing power and enable more potential solutions (in terms of different weights matrixes that solve the task), but that does not necessarily mean that those solutions are easier to find (by the learning algorithm). Apart from runtime considerations, an overabundance of hidden units also has the disadvantage of hampering generalization, because the network can, potentially, solve the task on an exemplar-by-exemplar basis rather than being forced to extract the regularities within the train set. Conversely, however, an insufficient number of hidden units will prevent the network from learning anything but the coarsest regularities in the task.

node: an individual processing unit within a neural network model, usually taken to roughly mimic the functionality of a biological neuron or group of neurons. A note performs two functions: (a) summing up all incoming activation (=> netinput) and (b) passing this sum through an activation function (=> activation value). The incoming activation for a node is determined by multiplying the activation values of all upstream nodes with the respective connection weights. Input units are not usually counted as nodes because their activation value is determined directly by the current input pattern an they thus don not perform any computations. A similar argument applies to bias nodes.

patterns: in supervised networks, input and target patterns form pairs that represent a stimulus processed by the network and the expected response. A single input pattern is a vector of numbers (often binary, i.e. zeros and ones) that determines which activation values the input units will be set to when this pattern is processed. Similarly, a target pattern defines the activation values that the units in the output layer should assume when the corresponding input pattern is passed through the network (The numerical deviation from the target pattern is called the error). From this it follows that the number of elements (numbers) in an input pattern should correspond to the number of input units in the network, whereas the length of the target pattern should correspond to the size of the output layer. The number of different training or test patterns, conversely, corresponds to the number of different stimuli that the network processes. During training, the network is usually presented with the train patterns repeatedly (in a specified order), for verification the network processes each of the train patterns once at the end of training, and testing means that the different set of test patterns (often novel patterns that the network was not exposed to during training) are processed once. Weights are adapted according to the learning algorithm during training, but not during verification or testing. In OXlearn, patterns are represented in a matrix where rows correspond to individual patterns and columns give the respective elements/values.

presentation order: the order in which the individual input/target pattern pairs are presented (repeatedly) to the network during training. *Sequential* presentation follows the order in which the patterns are organized in OXlearn, starting over once the last pattern (= row of the matrix OXinput) has been presented. Presenting the patterns *randomly without replacement* means that, similarly, all patterns are presented once before starting over, but the order is randomly determined for each pass (or epoch). When presenting *randomly with replacement*, one of the patterns is randomly chosen for each individual sweep.

mean squared error: value obtained by first raising the deviation of an output unit's activation value from the corresponding target value (see error), and then averaging over all output units.

run: training a network for a given number of sweeps (or until the mean square error is below a certain value). The weight initialization at the beginning of a run means that all knowledge acquired in from earlier runs is discarded, each run represents a totally new take, similar to training *another* network on a similar task. Successive runs with an unchanged set-up may still produce divergent results due to the weights initialization and/or the presentation order, depending on the settings chosen.

seed: a 'label' for a specific random distribution. A specific seed guarantees that the *same* set of random numbers is drawn whenever this specific seed is used. For example, the weights matrix at the start of training can be kept constant between training runs when a seed is specified, thus ruling out that between run differences are due to the weights initialization.

sweep: forward pass of a single pattern through the network. During training, this is usually followed by a backwards propagation of the observed error and the resulting adjustment of weights through application of the learning algorithm.

task: the task of a network is defined by the entire set of input-target pattern pairs that it is trained on. The network can be said to have solved the task if it performs to a certain criterion (i.e. max error of any output node < 0.1) for all patterns/stimuli. How difficult it is for a network to learn a task depends on several factors, most of which can be linked to the related concepts of interference and redundancy. Learning is easy when many similar (i.e. patterns whose distribution of activation values overlap considerably) input patterns exist that require similar responses (i.e. have similar target patterns, such groups of patterns are often termed 'friends'). Learning is difficult when there is much competition for share processing resources (weights), e.g. when only a small aspect of the input pattern (e.g. the activation value of only one input unit) distinguishes one desired response (target) from another (this is called an 'enemy') or when patterns exist that (a) are infrequent and (b) are not backed up by 'friends'. Maybe surprisingly this means that neural networks are usually more robust (i.e., tolerant with respect to the choice of learning rate, number of hidden units, presentation order, initial weights, etc.) for larger, more naturalistic tasks, because these often involve a large number of highly redundant stimuli.

testing: the presentation of a test set of input-target patterns to a trained network (sometimes also done at intermediate stages during training). The test patterns are usually different from the patterns that the network had processed during training, thus allowing to test the network's ability to generalize, i.e. to transfer the 'knowledge' that was extracted from the training exemplars to novel stimuli. During testing (and verification) weights are frozen, that is, the weights are not adjusted any more.

training: see run.

unit: see node

verification: the presentation of the set of train patterns (once, in sequential order) after training has finished. During verification (and testing) weights are frozen, that is, the weights are not adjusted any more. Verification is necessary to evaluate the network's

performance on the entire task for a given weights matrix. Such an evaluation should not be based on the training performance because the weights change (if only slightly) after each sweep and even adjacent sweeps are thus not directly comparable.

weight: a numerical value (can be positive or negative) that is associated with the connection from one node to another. The activation that the downstream node receives is weighted, that is, the upstream node's activation value gets multiplied with the weight of the connection. Weights represent the strength of the connection or association between two nodes. During the training process, the weights are adjusted according to a learning algorithm.

weights initialization: The values that are assigned to the weights in a network at the very beginning of a training run. The virgin network needs to start from somewhere, and it is usually taken to be the least arbitrary solution to simply assign small random weights (see seed). Because a network's performance is entirely determined by its weights matrix, however, the random values drawn here can in some cases have a profound impact on the network's learning progress. It could be, for example, that the randomly drawn set of initial weights is, by pure chance, very similar to a weights matrix that enables the network to perform correctly on the given task, in which case there would not be much learning (i.e., adjustment of weights) left to do. Conversely, it is possible that the initial weights matrix is detrimental to learning the task, thus either prolonging the learning process or, in the worst case, preventing the network from finding an optimal solution (see gradient descend).

weights matrix: a set of weights, usually this refers to either all weights between two layers or all weights within the whole network. The weights matrix in the latter sense is what defines the networks functionality and 'knowledge'. The process of learning, in neural networks, thus consists of finding an appropriate weights matrix through repeated processing of train patterns and subsequent application of the learning algorithm.