UNIVERSITÀ DEGLI STUDI DI FOGGIA

HR EXCELLENCE IN RESEARCH

Artificial Neural Networks

Insights

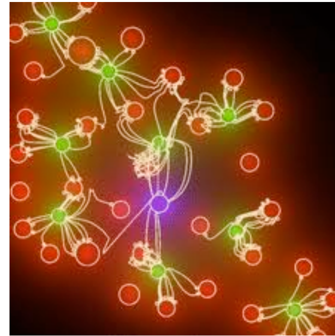
Prof. Crescenzio Gallo

Aggregate Professor of Information Processing Systems

Department of Clinical and Experimental Medicine

crescenzio.gallo@unifg.it

Neural Networks: insights

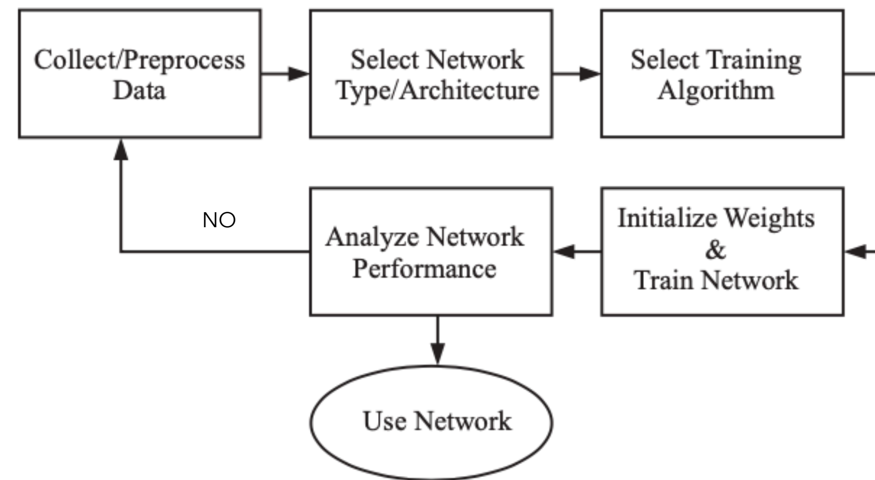


In the first part "Basic Concepts" we have seen the architectures used for neural networks and the *training* techniques from a general point of view.

In this second part we will examine some practical aspects of neural network training, and in particular:

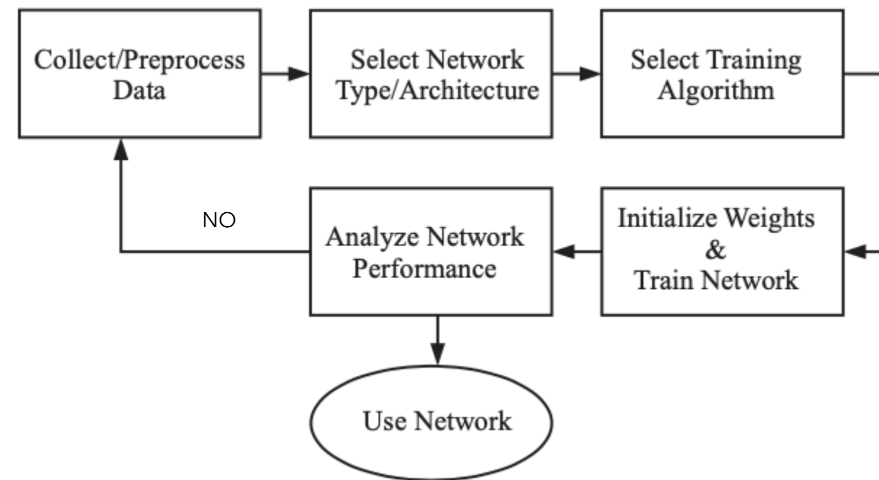
1. **data collection** and **pre-processing**, and **choice of network architecture;**
2. **network training;**
3. **post-training analysis.**

The *training* process



- The training process of a neural network is an iterative procedure that starts with the **collection** of data and their **preprocessing** to make the training process more efficient. At this stage it is also necessary to establish the distribution of the data in the sets of training, validation (optional) and final test.
- After the data have been selected, the **type** of network (MPL, dynamic, etc.) and its **architecture** (e.g., number of layers, number of hidden neurons) must be chosen.
- Finally, the appropriate **training algorithm** for the network and the problem to be solved must be chosen.
- After the network has been trained (i.e., the weights have been defined), **its performance must be analyzed**. This analysis may uncover problems with the data, the network architecture, or the training algorithm. In this case, the entire process must be repeated until the network performance is deemed satisfactory.

The *training* process

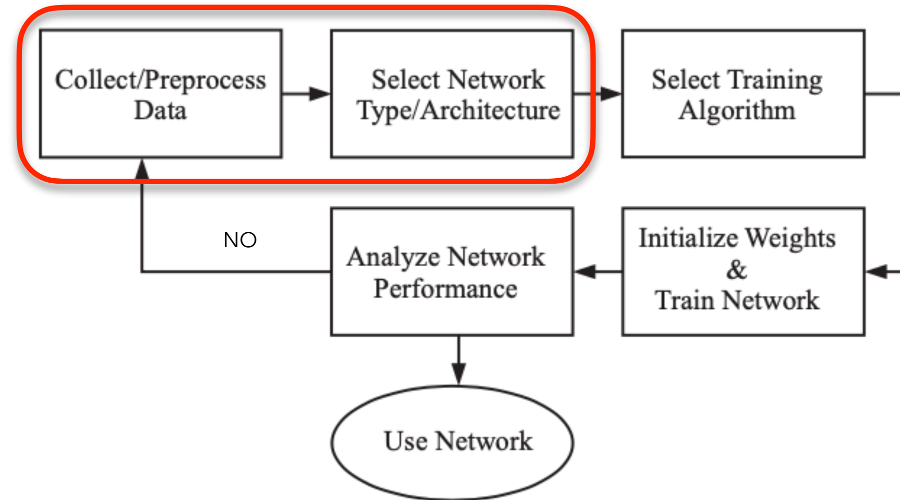


In the following we will examine each part of the training process in detail: pre-training phases (data collection/ preprocessing), real training of the network, post-training analysis.

A preliminary consideration to the use of a neural network deserves to be made at this point: **Is the neural network really the right solution to the problem to solve?** Often a simple linear solution is sufficient to solve a non-complex problem.

For example, there is no need for a neural network for a fitting problem if a standard linear regression already produces a satisfactory result. Neural networks possess greater modeling capability, but at the expense of much more demanding training (and computational) requirements. If linear methods work, they are always the most appropriate choice....

Pre-training phases



The steps prior to network training can be grouped into three categories:

- data **collection**;
- data **pre-processing**;
- choice of network **type** and **architecture**.

Pre-training phases

Data collection

- The main difficulty of a network lies in the acquisition of knowledge related to the problem domain: the "goodness" of a network will be closely related to the **quality of the data available** for training.
- The training data must fully represent the entire "input space" for which the network will be used. There are training methods that can be used to ensure that the network interpolates (**generalizes**) accurately over the range of data provided.
- However, the performance of the network cannot be guaranteed when its inputs are outside the range of the training set. Neural networks, like other nonlinear methods, do not extrapolate well ...
- It is not always easy to be sure that the input space is **adequately sampled** from the training data. For many problems, the size of the input space is too large and often the input variables are dependent. Therefore, it may not be possible to precisely define the useful region of the input space; however, we can often collect data that is representative of all the conditions for which we intend to use the network and then employ it for proper and comprehensive training.

Pre-training phases

Data collection

How can we be sure that the **input space** has been **adequately sampled** from the training data? This is difficult to do prior to training, and there are many cases where we have no control over the data collection process (especially in the clinical field) and therefore must use whatever data is available.

By analyzing the trained network, we can often tell if the training data was sufficient. In this case, we can use techniques that indicate when a network is being used outside of the range of data it was trained on. This will not prove the performance of the network, but it will prevent us from using a network in situations where it is not reliable.

After collecting them, we generally divide (by random sampling) the data into three groups: *training*, *validation*, and *testing*, usually in the ratio of 70% / 15% / 15%. It is important that each of these groups be **stratified**, i.e., representative of the complete data set.

Pre-training phases

Data collection

A final question about data collection is, "*Do we have enough data?* "This question is difficult to answer, especially before training the network. The amount of data required depends on the complexity of the underlying function we are trying to approximate.

If the function to be approximated is very complex, with many inflection points, then this requires a large amount of data.

If the function is regular, then the data required is significantly small (unless it is very noisy).

The choice of dataset size is closely related to the choice of the number of neurons in the neural network. Of course, we generally do not know how complex the underlying function is before we begin training. For this reason, the process of training the neural network is iterative.

At the end of the training, the performance of the network is analyzed: the results of this analysis can help us decide whether we have enough data or not.

Pre-training phases

Data pre-processing

The main purpose of the data preprocessing phase is to **facilitate network training**.

The pre-processing phase consists of operations such as *normalization*, nonlinear *transformations*, feature *extraction*, discrete input/output *coding*, *missing data* handling, etc. The idea is to perform pre-processing of the data to facilitate the extraction of relevant information for training the neural network.

For example, in multilayer networks, sigmoid transfer functions are often used in the hidden layers. These functions essentially become saturated when the net input is greater than three ($e^{-3} \cong 0{,}05$). If we don't want this to happen at the beginning of the training process, it is common practice to **normalize** the inputs (usually in the range [-1,1]) before applying them to the network.

$$\frac{1}{1+e^{-x}}$$

Pre-training phases

Data pre-processing → **Normalization**

There are two standard methods for *normalization*. The first method **scales** the data so that it falls within a standard range, typically -1 to 1. This can be done by transformation:

$$x_i' = 2 \cdot \frac{x_i - x_{min}}{x_{max} - x_{min}} - 1$$

An alternative procedure (called **standardization**) is to transform the data so that they have a specific mean and variance, typically 0 and 1:

$$x_i' = \frac{x_i - \mu}{\sigma}$$

Generally, the normalization step is applied to both the input and output values.

Pre-training phases

Data pre-processing → **Non-linear transformations**

In addition to normalization, which involves a linear transformation, *nonlinear transformations* are sometimes performed as part of the preprocessing step.

Unlike normalization, which can be applied to any set of data, these nonlinear transformations are **case-specific**.

For example, many economic variables exhibit logarithmic dependence: in this case, it may be appropriate to take the logarithm of the input values before applying them to the neural network.

Another example is the simulation of molecular dynamics, where atomic forces are calculated as functions of the distances between atoms: since it is known that forces are inversely related to distances, we could perform the reciprocal transformation on the inputs, before applying them to the network.

This represents a **way of incorporating prior knowledge** into neural network training. If the nonlinear transformation is chosen intelligently, it can make training the network more efficient. Preprocessing will alleviate some of the work required of the neural network to compute the underlying function between the inputs and outputs.

Pre-training phases

Data pre-processing → **Feature extraction**

Another data pre-processing step is called *feature extraction*. This generally applies to situations where the input size is very large and the input components are redundant.

The idea is to **reduce the size of the input space** by computing a small set of features (characteristic variables) from the dataset and using them as input to the neural network.

For example, neural networks can be used to analyze ECG (electrocardiogram) signals to identify heart problems. The ECG might involve 12 or 15 signals measured over several minutes at a high sampling rate. This is too much data to send directly to the neural network. Alternatively, some characteristics of the ECG signal are extracted, such as the mean time intervals between certain waveforms, the mean amplitudes of certain waves, etc.

Pre-training phases

Data pre-processing → **Principal Components (PCA)**

There are also some general purpose *feature extraction* methods. One of them is the **principal component analysis** (PCA) method.

This method transforms the original input vectors so that the components of the transformed vectors are uncorrelated. In addition, the components of the transformed vector are ordered such that the first component has the largest variance, the second has the next largest variance, etc.

Generally, **only the first components** of the transformed vector, which account for most of the variance of the original vector, are used. This results in a large reduction in the size of the input vector if the original components are highly correlated.

The **disadvantage** of using PCA is that it only considers the linear relationships between the components of the input vector. When reducing the dimension using a linear transformation, some nonlinear information may be lost. Since the main purpose of using neural networks lies in their nonlinear mapping power, one must be careful when using PCA before applying inputs to the neural network.

There are also nonlinear versions of PCA, such as "Kernel PCA" (Schölkopf *et al.*, 1999) and "Feature Extraction with Nonlinear PCA" (Gallo & Capozzi, 2019).

Pre-training phases

Data pre-processing → **Discrete data encoding**

Another important preprocessing step is needed when the inputs or outputs take on only **discrete values**. For example, if we have a pattern recognition problem in which there are four classes, there are at least three common ways to encode the outputs.

First, we can have **scalar outputs** that take on four possible values (e.g., 1, 2, 3, 4). Second, we can have **two-dimensional targets**, representing a binary code of the four classes (e.g., (0,0), (0,1), (1,0), (1,1)). Third, we can have **four-dimensional outputs**, where only one neuron is active at a time (e.g., (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,1)): the latter method tends to give the best results in general. Note that discrete inputs can be encoded in the same way as outputs.

When encoding outputs, we must also consider the **transfer function** used in the output layer of the network. For pattern recognition problems, we typically use sigmoid functions: *logsig* or *tansig*. If we use *tansig* in the last layer (the most common case) we might consider assigning output values to -1 or 1, which represent the asymptotes of the function. However, this tends to cause difficulties for the training algorithm, which tries to saturate the sigmoid function to reach the output value.

It is best to assign target values at the point where the second derivative of the sigmoid function is maximum. For the *tansig* function this is when the input is -1 and 1, which corresponds to output values -0.76 and +0.76.

Pre-training phases

Data pre-processing → **Discrete data encoding**

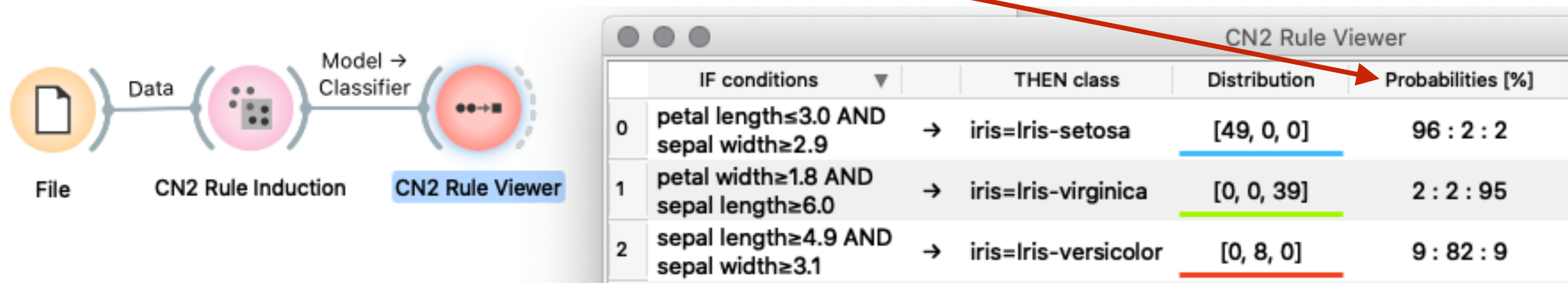
Another transfer function used in the output layer of multilayer networks for pattern recognition problems is the **softmax** function (or *normalized exponential function*), a generalization of the logistic function. This transfer function has the form:

$$f : \mathbf{y} \in \mathbb{R}^N \to \left\{ \mathbf{z} \in \mathbb{R}^N | z_i > 0, \sum_{i=1}^{N} z_i = 1 \right\}$$

$$f(y_j) = \frac{e^{y_j}}{\sum_{i=1}^{N} e^{y_i}} \quad \text{per } j = 1, \ldots, N$$

The outputs of the *softmax* transfer function can be interpreted as the probabilities associated with each class. Each output is between 0 and 1, and the sum of the outputs is equal to 1.

An example of application can be found in the Orange classification widget "CN2 Rule Induction" and its related widget "CN2 Rule Viewer" of classification results visualization.



Pre-training phases

Data pre-processing → **Missing data**

Another practical issue to consider is **missing data**, which can occur for a variety of reasons.

For example, we may have clinical data collected at monthly intervals, with some months in which some tests were not performed for some patients. The simplest solution to this problem would be to eliminate incomplete records: however, the amount of data available may be very limited and it may be very expensive to collect additional data. In this case, full use should be made of all the data we have, even if they are incomplete.

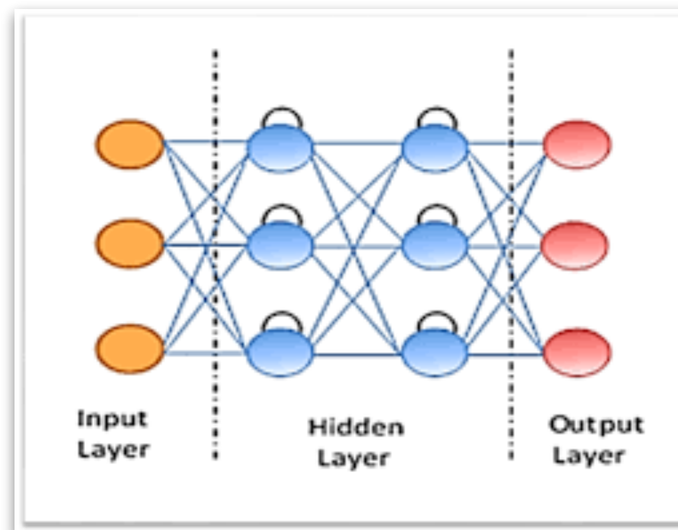
There are **several strategies** for handling missing data. If there is missing data for an input variable, one possibility is to replace the missing values with the ***mean value*** for that particular variable, with the possible addition of a binary input variable indicating the completion of missing data for that variable. This would provide the classifier with information about the missing variables for proper processing of those variables.

If missing data occurs in the output, the network performance index can be modified to not include the errors associated with the missing output values.

Pre-training phases

Choice of network architecture

- The next step in the network training process is to **choose the network architecture**.
- The basic type of network architecture is determined by the type of problem we want to solve.
- Once the basic architecture is chosen, specific details such as the *number of layers* and *hidden neurons*, how many outputs the network should have, and what type of *performance index* to use for training must be decided.



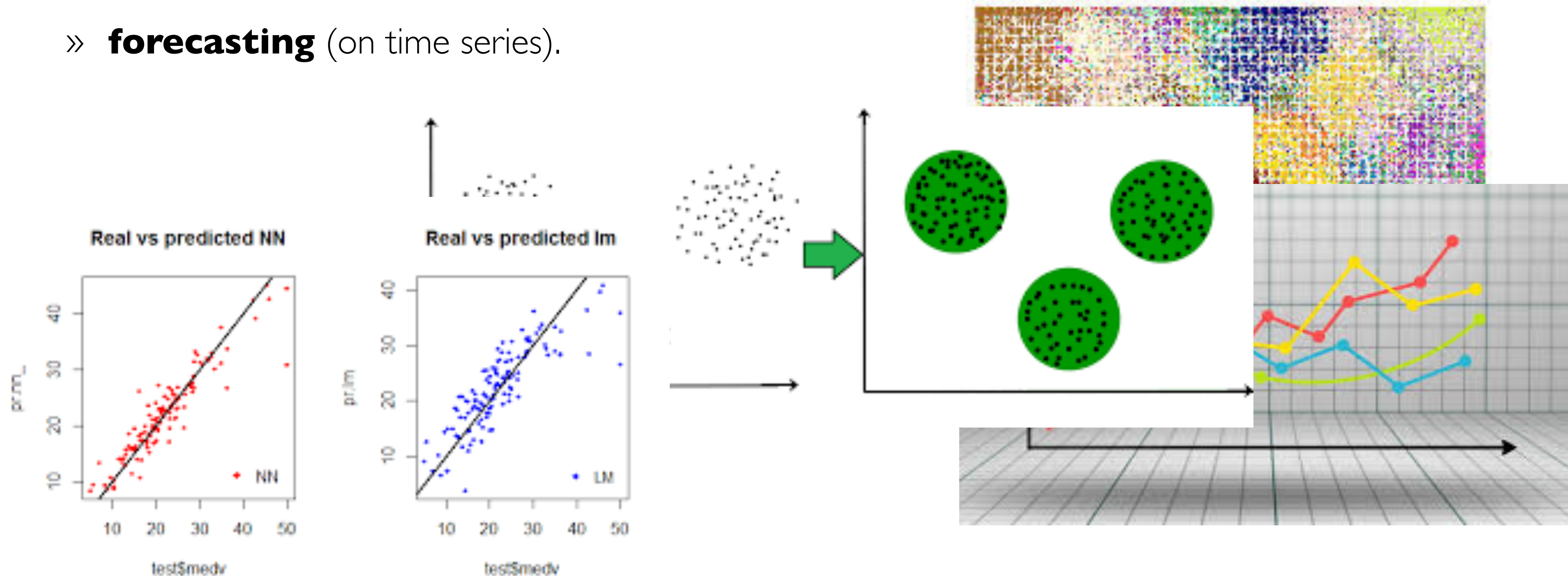
Pre-training phases

Choice of network architecture → **Basic architecture**

The first step in choosing an architecture is to **define the problem** you are trying to solve.

For our purposes, we will limit the discussion to four types of problems:

- » **fitting** (*regression*);
- » **pattern recognition** (*classification*);
- » **clustering**;
- » **forecasting** (on time series).



Pre-training phases

Choice of network architecture → Basic architecture → **Fitting**

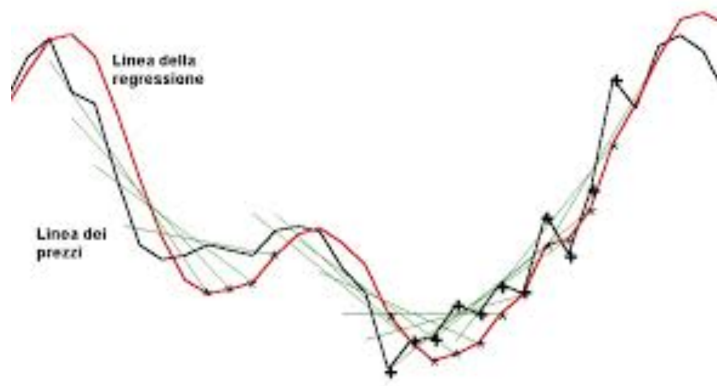
In fitting (*regression*, also referred to as *function approximation*), you want a neural network to "map" a set of inputs to corresponding targets.

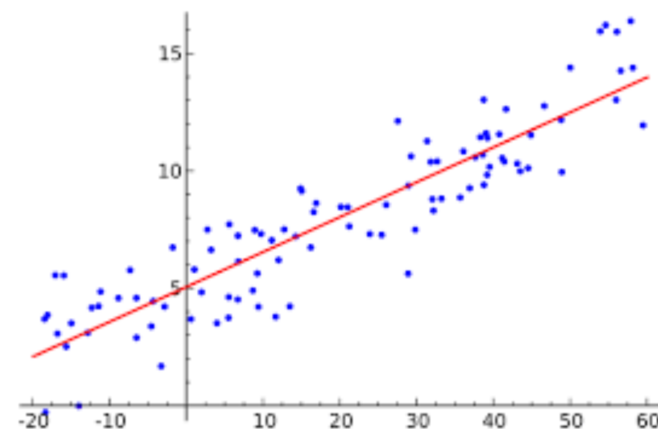
For example, a real estate agent might want to *estimate the prices of a house* based on variables such as the *tax rate*, the *pupil/teacher ratio* at neighborhood schools, and the neighborhood *crime rate*.

An automotive engineer may want to estimate engine emission levels based on *fuel consumption* and *speed* measurements.

A physician may want to predict a patient's level of body fat based on measurements (variables) such as *weight* and *height*.

For fitting problems, the target variable takes on **continuous numerical values**.

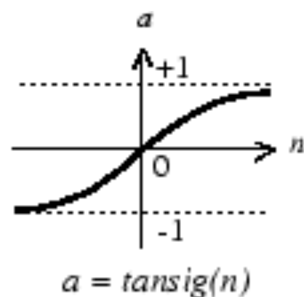




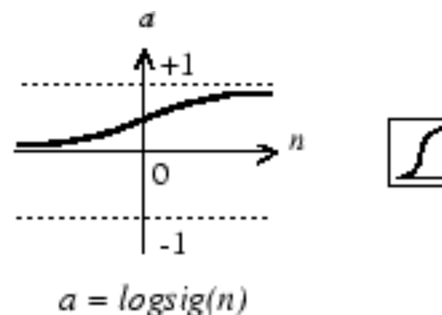
Pre-training phases

Choice of network architecture → Basic architecture → **Fitting**

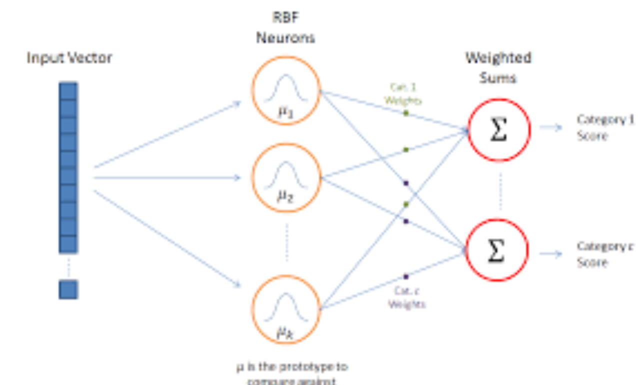
- The **standard neural network architecture** for regression problems is the **Multi Layer Perceptron** (MLP), with *tansig* neurons in the hidden layers, and linear neurons in the output layer.
- The *tansig* transfer function is generally preferred to the *logsig* transfer function in the hidden layers for the same reason that the inputs are normalized: it in fact produces outputs (which are inputs to the next layer) centered around zero, while the *logsig* transfer function always produces positive outputs.
- For most fitting problems, a single hidden layer is sufficient. If the results with one hidden layer are not satisfactory, sometimes two layers are used; rarely more than two hidden layers are used. For very difficult problems, *deep networks* with many layers have been used.
- Linear transfer functions are used in the *output layer* for regression problems because the **output is a continuous variable**.
- Radial Basis Function (RBF) networks can also be used for regression problems, with the Gaussian transfer function used in the hidden layer and the linear transfer function in the output layer.



Tan-Sigmoid Transfer Function



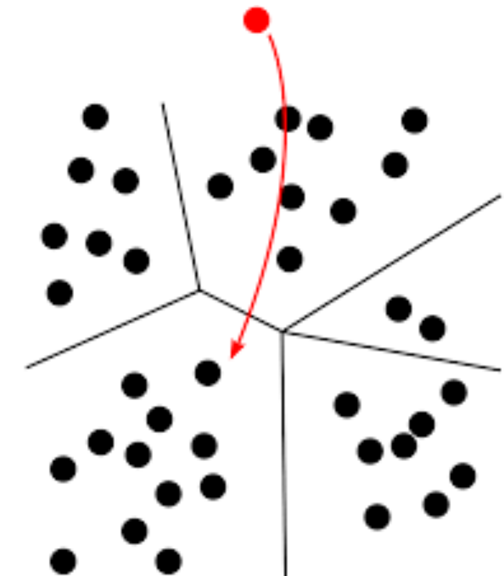
Log-Sigmoid Transfer Function



Pre-training phases

Choice of network architecture → Basic architecture → **Classification**

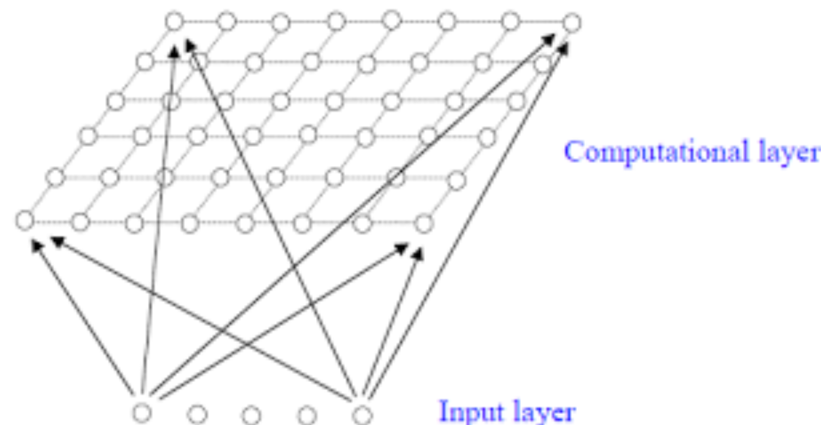
- In **classification** (or *pattern recognition*) problems, inputs are matched to a set of predefined categories (classes).
- *For example, a wine merchant might want to recognize the vineyard from which a particular bottle of wine comes, based on a chemical analysis of the wine. A physician might want to classify a tumor as benign or malignant, based on uniformity of cell size, specimen thickness, and cell mitosis.*
- In addition to fitting problems, **MLP networks** can also be used for *pattern recognition*.
- The main difference between a network for a fitting problem and one for *pattern recognition* lies in the transfer function used in the output layer.
- For *pattern recognition* problems generally a **sigmoid function in the output layer** is used.
- The RBF network can also be used for *pattern recognition*.



Pre-training phases

Choice of network architecture → Basic architecture → **Clustering**

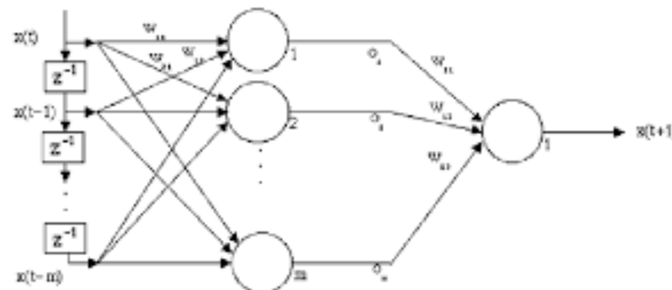
- In **clustering** problems, one wants to implement a neural network to group data by similarity.
- *For example, companies may want to perform market segmentation, which is done by grouping people based on their buying patterns. Computer scientists may want to perform data mining by breaking down data into related subsets. Biologists may want to perform bioinformatics analysis, such as grouping genes with their expression profiles.*
- Any of the **competitive networks** could be used for clustering. The **SOM network** (Self Organizing Map, or Kohonen network) is the most popular network for clustering. The main advantage of this architecture is that it allows the visualization of input spaces with many dimensions.



Pre-training phases

Choice of network architecture → Basic architecture → **Forecasting**

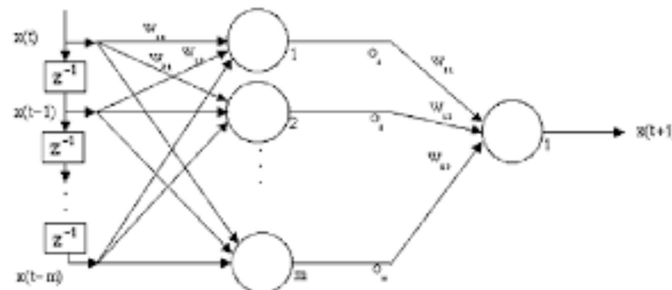
- **Forecasting** also falls into the categories of time series analysis, systems identification, filtering, or dynamic modeling. The idea is that you want to predict the future value of some time series.
- *A stock trader might want to predict the future value of a stock. A chemist might want to predict the future value of the concentration of a certain substance in output from a processing plant. An engineer might want to pre-determine power grid outages.*
- Prediction requires the use of **dynamic neural networks**. The specific form of the network will depend on the particular application.
- The simplest network for nonlinear prediction is the (temporal) delay neural network, which is part of a general class of networks in which the dynamics appear only at the input layer of a static multilayer feedforward network.
- This network has the advantage that it can be trained using static *backpropagation* algorithms, since the delay line at the input of the network can be replaced by an extended vector of delayed input values.



Pre-training phases

Choice of network architecture → Basic architecture → **Forecasting**

- For modeling and dynamic control problems, the **NARX** (Non-linear AutoRegressive model with eXogenous input) network is often used.
- *In this network the input signal might represent, for example, the voltage applied to a motor and the output might represent the angular position of a robot arm.*
- As with the delayed neural network, the NARX network can be trained with static backpropagation. The two delayed lines can be replaced with extended vectors of delayed inputs and outputs.
- *Outputs can be used instead of feeding them back into the network (which would require training with dynamic backpropagation) because the outputs of the network should match the required outputs at the end of training.*



Pre-training phases

Choice of network architecture → **Details**

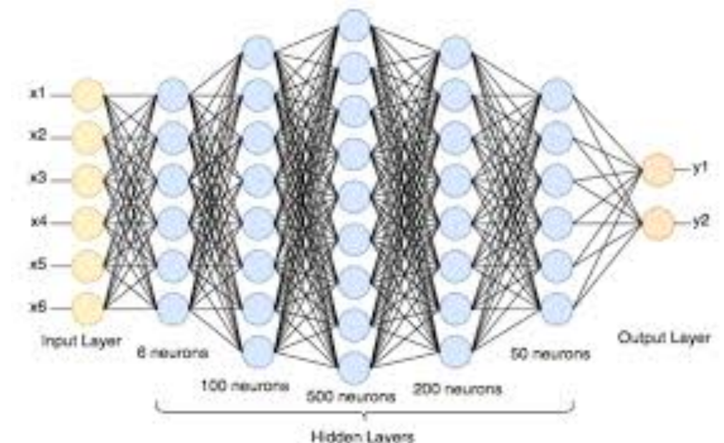
After the basic structure of the network, we need to select the **details of the architecture** (e.g., the *number of hidden layers*, the *number of neurons*, etc.).

In some cases, the choice of base architecture will automatically determine the number of layers. For example, if you chose the SOM architecture for clustering, then the network will have only one layer (the feature map).

In the case of the multilayer network for regression or classification problems, the number of hidden layers is not determined by the problem. The standard procedure is to start with only one hidden layer. If the performance of the network is not satisfactory, an additional hidden layer can be added.

More than two hidden layers are rarely used, as training the network becomes more difficult. This is because each layer performs a "flattening" operation due to the sigmoid functions used, resulting in slower convergence.

For very difficult problems, however, you can use ***deep*** (multilayer) ***neural networks*** with several hidden layers, for which parallel computation is required for convergence in a reasonable time.



Pre-training phases

Choice of network architecture → **Details**

The **number of neurons in each layer** also needs to be selected:

- the number of neurons in the input layer is constrained to the size of the input vector;
- the number of neurons in the output layer is equal to the size of the output vector;
- the number of neurons in the hidden layers is determined by the complexity of the function being approximated, which is not known until we try to train the network.

The standard procedure is to start with more neurons than necessary, and then use the **early stopping** technique to prevent *overfitting*.

However, there may be situations where you are **interested in limiting the computation time or memory space** required by the network (e.g., for real-time implementation on resource-constrained microcontrollers). In these cases, one wants to find the simplest network that "reasonably" fits the data.

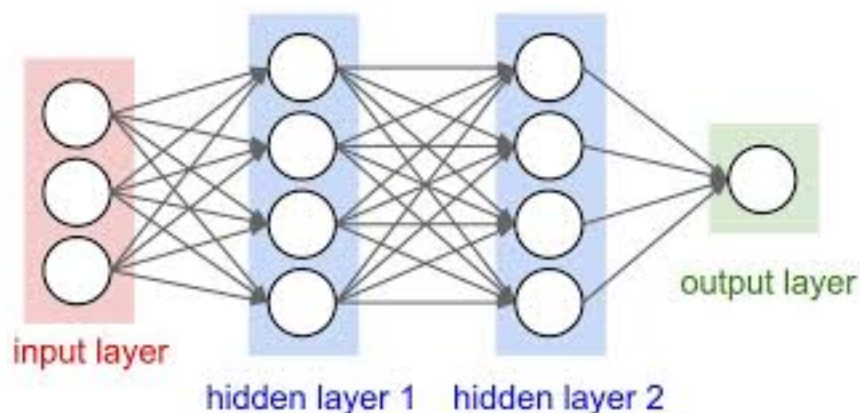
Pre-training phases

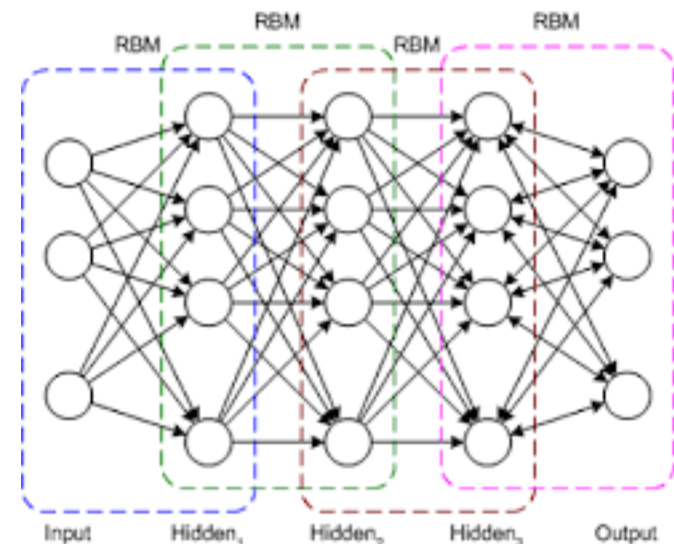
Choice of network architecture → **Details**

When there are **multiple outputs**, a decision must be made whether to have a network with multiple output neurons (one for each value to be computed) or to implement multiple networks, each with a single output.

*For example, neural networks are used to **estimate** LDL, VLDL, and HDL **cholesterol levels** based on a spectral analysis of blood. It is possible to have a neural network with three neurons in the output layer to estimate all three cholesterol levels, or we could have three neural networks, each estimating only one of the three components.*

Theoretically, both methods should work, but in practice one method may work better than the other. We generally start with a multi-output network, and then switch to multiple single-output networks if the original results are not satisfactory.





Pre-training phases

Choice of network architecture → **Details**

Another architectural choice is the **size of the input vector**, which is usually a simple choice determined by the input data.

However, there are situations where the input data has redundant or irrelevant components (*features*), which it may be advantageous to eliminate. This can reduce the computational time required and can help prevent *overfitting* during training.

The process of **choosing the inputs** (*ranking*) for nonlinear networks can be quite difficult and there is no perfect solution: The most widely used techniques are e.g. information indices (*Information Gain*, *Gain Ratio*), *Gini* (inequality between the values of a frequency distribution), *ANOVA* (difference between the averages of feature values in different classes), χ^2 (dependence between feature and class as a measure of the chi-square statistic), *ReliefF* (ability of an attribute to distinguish between classes on similar data instances), *FCBF* (Fast Correlation Based Filter, an entropy-based measure that identifies redundancy due to pairwise correlations between features).

Another technique that can aid in reducing the input vector is trained network **sensitivity analysis**, which will be discussed below.

Training

After the collection and pre-processing of the data and the choice of the network architecture, we move on to the ***training*** phase.

In this section we will look at some of the decisions that need to be made as part of the training process.

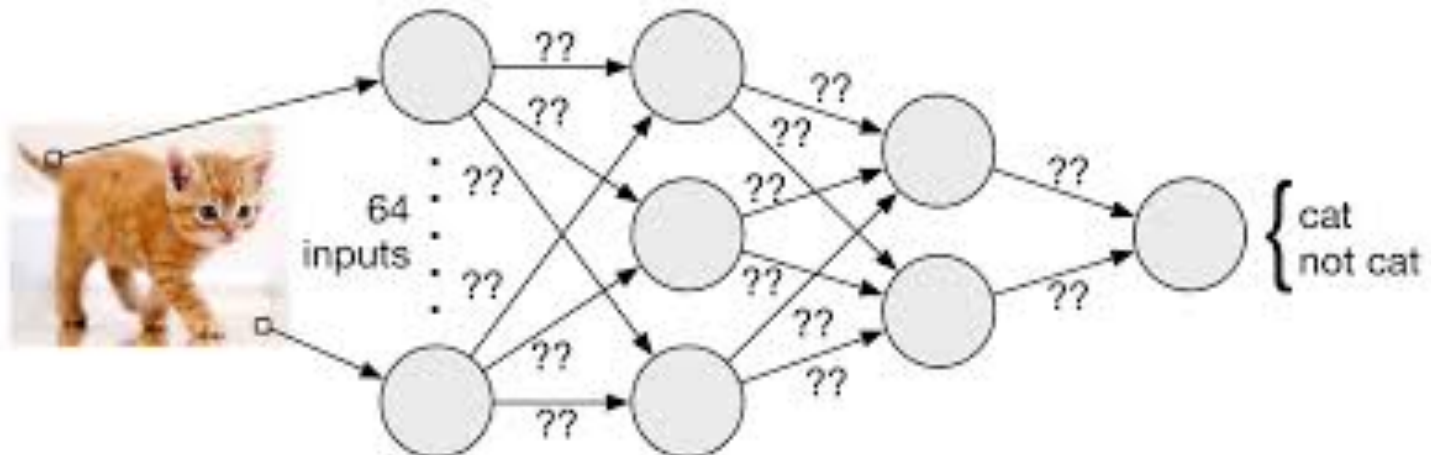
This includes:

» the ***weights initialization method***

» the ***training algorithm***

» the network ***performance index***

» the training **stop criterion**



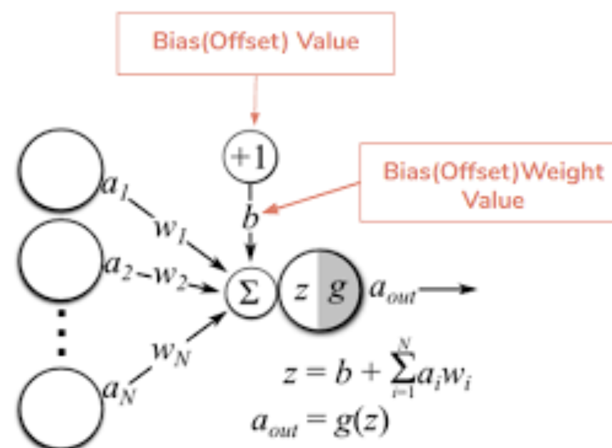
Training

Weights initialization

Before starting network training, the **weights and biases must be initialized**. The method used depends on the type of network.

For multilayer networks, the weights and biases are generally set to *small random values* (e.g., uniformly distributed between -0.5 and 0.5; if the inputs are normalized between -1 and 1) to avoid falling on a saddle point of the error surface.

For competitive networks, the weights can also be set as small random numbers. The initial size of neighboring neurons (*neighborhood*) is large enough so that all neurons have the opportunity to "learn" during the initial stages of training. This will move all weight vectors to the appropriate region of the input space.



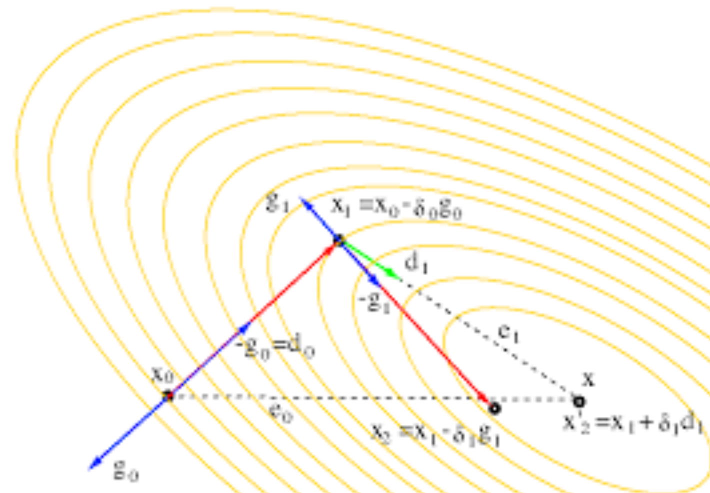
Training

Choice of the training algorithm

For multilayer networks, ***gradient***-based algorithms are generally used. These algorithms can be implemented in two **ways**:

- **sequential**: weights are updated after each input is presented to the network;
- **batch**: all inputs are presented to the network, and the total gradient is calculated by summing the gradients for each input before the weights are updated.

In some situations, the sequential form is preferable (e.g., when an on-line operation is required). However, many of the most efficient optimization algorithms (e.g., the conjugate gradient and Newton's methods) are inherently batch algorithms.

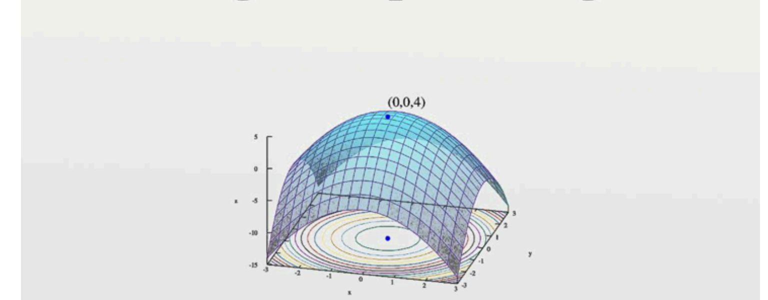


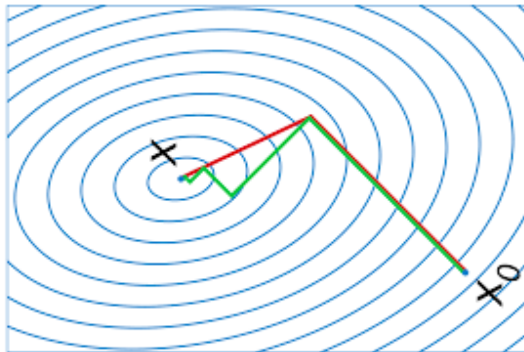
Training

Choice of the training algorithm

For multilayer networks with a few hundred weights and biases used for function approximation, the **Levenberg-Marquardt** algorithm is usually the fastest training method.

When the number of weights reaches a thousand or more, however, the Levenberg-Marquardt algorithm is not efficient, mainly because the computation of the inverse of the matrix scales geometrically with the number of weights.



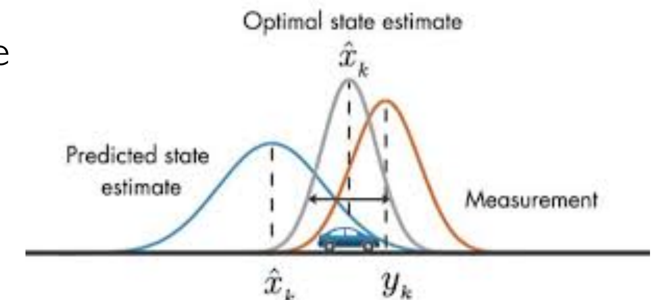


For large networks, the **Scaled Conjugate Gradient** algorithm is very efficient.

This method is also applicable for *pattern recognition* problems.

Of the algorithms that can be implemented in sequential mode, the fastest are the extended **Kalman filter** algorithms closely related to sequential implementations of the Gauss-Newton algorithm.

They do not, however, unlike the batch version of Gauss-Newton, require an inversion of the approximate Hessiana matrix.



Training

Stop criteria

The **training error** almost never converges identically to zero, especially for multilayer networks. Therefore, we need to have other **criteria** for deciding when to stop training the network.

We can stop training when the **error reaches some predefined limit** (although it is difficult to establish an acceptable error level a priori).

The simplest criterion is to stop the training after a fixed number of iterations, usually set at a reasonably high level. If the weights do not converge after that the maximum number of iterations has been caught up, **the training can be repeated** using the final weights of the first epoch like conditions begins them.

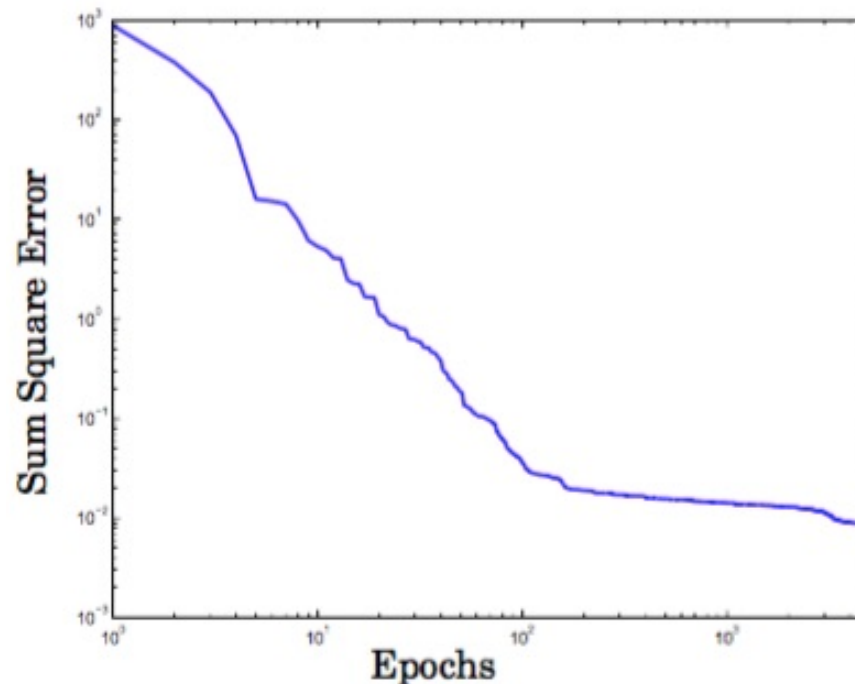
Another stopping criterion is the **norm of the error index gradient**. If this norm reaches a sufficiently small threshold, then the training can be stopped. Unfortunately, the error surface for multilayer networks may have many "flat" regions, where the gradient norm will be small. For this reason, the threshold for the minimum norm should be set to a very small value (e.g., 10^{-6} for the mean square error, with normalized targets), so that the training does not end prematurely.

Training

Stop criteria

We can also stop training when the reduction in error index per iteration becomes small. As with the gradient norm, however, this criterion may stop training too soon.

For multilayer networks, the error may remain nearly constant for a number of iterations before suddenly falling off. When the training is complete, it is useful to visualize the **error curve** on a log-log scale to check for convergence.



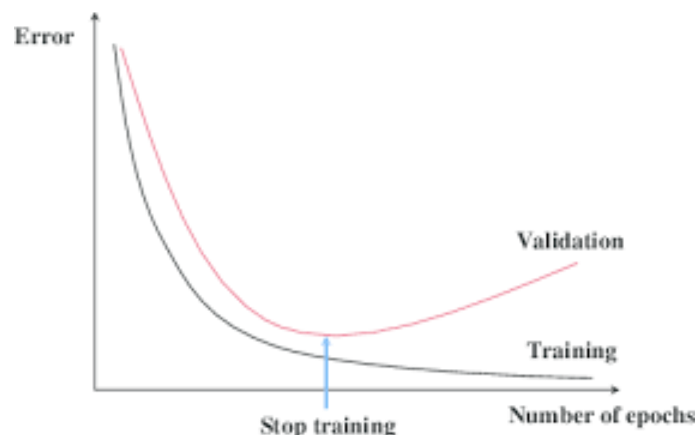
Training

Stop criteria

If ***early stopping*** is used to prevent overfitting, then training is stopped when the error on the validation set increases for a number of iterations. In addition to preventing overfitting, this stopping procedure also provides a significant reduction in computation; for most practical problems, the validation error will increase before any of the other stopping criteria are reached.

Training a neural network is an iterative process. Even after the convergence of the training algorithm, ***post-training analysis*** may suggest that the network is modified and re-trained.

In addition, several training sessions should be conducted to ensure that an overall minimum for the network has been achieved.



Training

Stop criteria

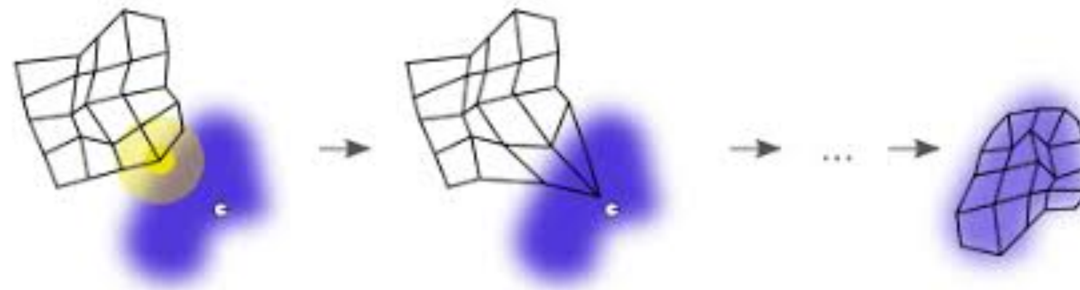
The previous stopping criteria apply primarily to gradient-based training.

When training **competitive networks**, such as SOM, there is no explicit error index or gradient to monitor for convergence. Training stops only when a predetermined maximum number of iterations has been reached.

For the SOMs, the learning rate and the size of the neighbouring neurons (*neighborhood*) decrease with time. Therefore, the maximum number of iterations establishes the end of the training and represents a very important parameter.

Generally this is chosen to be more than ten times the number of neurons in the network. This is an approximate number, and the network must be analyzed at the end of the training to determine if the performance is satisfactory.

The network may need to be trained several times with different training durations to achieve a satisfactory result.



Training

Choice of performance index

For multilayer networks, the standard performance index is the **mean square error**:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Another indicator is the **mean absolute error**, which is generally less sensitive to outliers in the data than the previous one:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

This concept can be extended to any power of the absolute error, as follows:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|^K$$

where K=2 corresponds to the mean square error and K=1 to the mean absolute error. The general error given by the above formula is also known as the **Minkowski error**.

Training

Choice of performance index → **Cross-entropy**

Mean square error works well for function approximation problems where the target values are continuous.

However, in pattern recognition problems, where the outputs take on discrete values, other performance indices are more appropriate.

The **cross-entropy** index is defined as follows:

$$E(x) = -\sum_{i=1}^{N} y_i \ln \frac{\hat{y}_i}{y_i}$$

The target values are assumed to be 0 and 1 at the two classes to which the input vector belongs. The *softmax* transfer function is generally used in the last layer of the neural network, if the cross-enterpy performance index is used.

Finally, recall that the *backpropagation* algorithm for computing the training gradients will work for any differentiable performance index.

Training

Multiple executions and network "committees"

A single training may not produce optimal performance, due to the possibility of reaching a local minimum of the error surface. It is best to **repeat** (five to ten times) the **training** under different initial conditions and select the network that produces the best performance.

There is another way to perform more training and make use of all the networks obtained. This method is called "**network committee**".

For each training session, the *validation set* is randomly selected from the *training set*, and a random set of initial weights and biases is chosen. After the networks have been trained, they are all used together to form a common output.

For function approximation (regression) networks, the joint output may be a simple average of the outputs from each network.

For classification networks, the joint output may be the result of a "vote," in which the class chosen by the majority of the networks is selected as the output.

The **performance** of the "committee" will usually be **better** than even the best of the individual networks. In addition, the variation in the results of the individual networks can be used to provide *confidence levels* for the output obtained.

Post-training analysis

Before using a trained neural network, we need to **analyze** it to determine if the training was successful.

There are many techniques for **post-training analysis**; we will examine the most common ones.

Because these techniques vary by application, we will organize them according to these four application areas:

- **fitting** (*regression*);
- **pattern recognition** (*classification*);
- **clustering**;
- **forecasting** (on time series).

Post-training analysis

Regression (*fitting*)

A useful tool for analyzing trained neural networks for *fitting* problems is the regression between the outputs of the trained network and the corresponding objectives (*targets*).

It involves fitting a linear function of the shape:

$$\hat{y}_i = a \cdot y_i + b + \varepsilon_i$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are respectively the angular coefficient and the ordinate at the origin of the regression line between the outputs $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ obtained by the network and the corresponding targets $\boldsymbol{y}_{\boldsymbol{i}}$ required, with a residual error $\boldsymbol{\varepsilon}_i$.

The $\boldsymbol{a}$ and $\boldsymbol{b}$ terms of the regression can be estimated as follows:

$$\hat{a} = \frac{\sum_{i=1}^{N} (y_i - \overline{y})(\hat{y}_i - \overline{\hat{y}})}{\sum_{i=1}^{N} (y_i - \overline{y})^2}, \ \hat{b} = \overline{\hat{y}} - \hat{a} \cdot \overline{y}$$

where $\overline{y}$ represents the average target value and $\overline{\hat{y}}$ the average value of the network outputs.

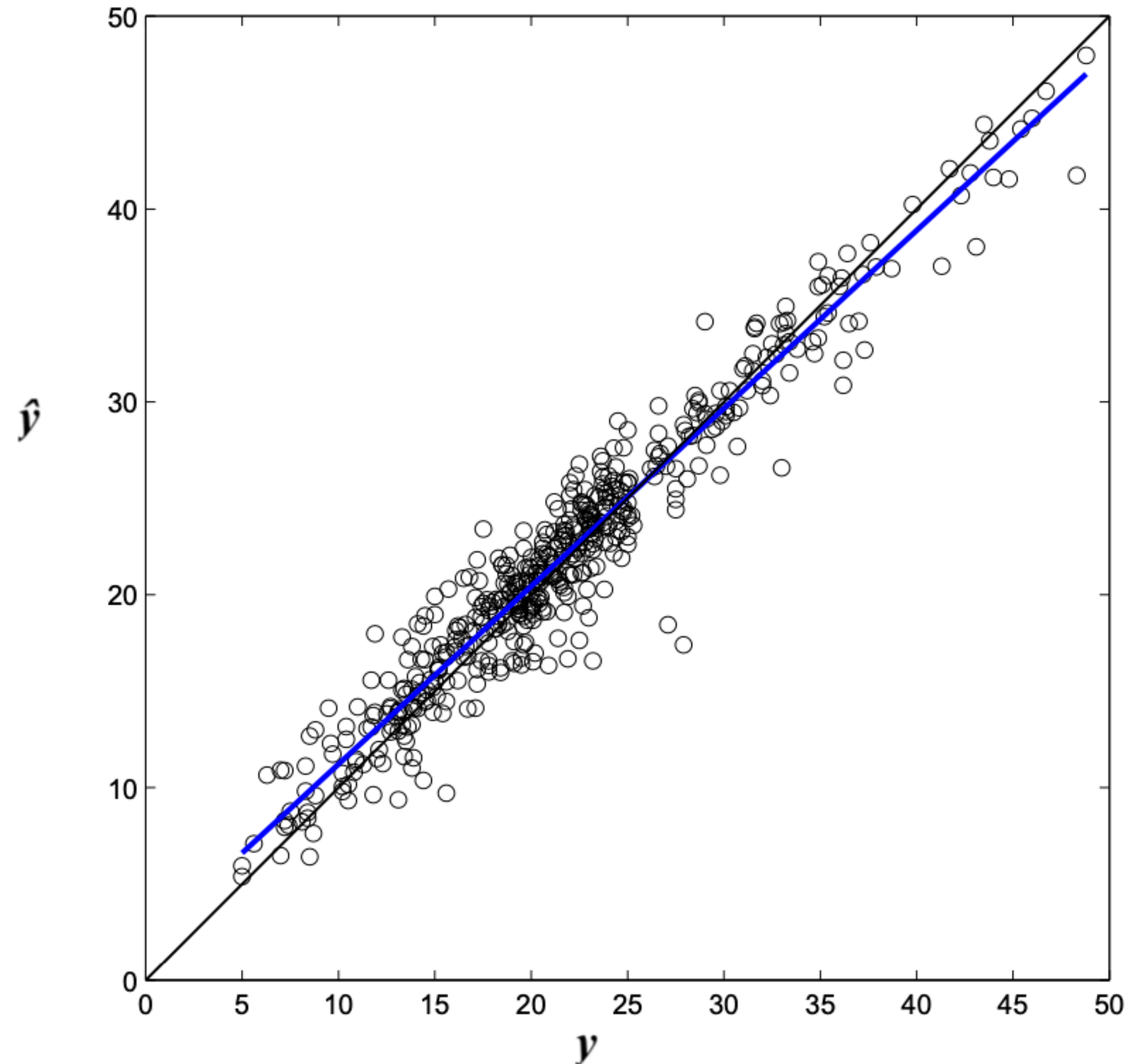
Post-training analysis

Regression (*fitting*) → **Scatterplot (regression analysis)**

The figure shows an example of **regression analysis** using *scatterplot*.

The blue line represents linear regression, the thin black line represents perfect matching $\hat{y} = y$, and the circles represent the data.

In this example, we can see that the match is quite good, though not perfect.



Post-training analysis

Regression (*fitting*) → **Outliers**

The next step is to investigate the points that fall away from the regression line (**outliers**).

For example, there are two points around (27;17) that appear to be outliers: it would be worth examining them in detail to see if there are any issues with data collection and/or preprocessing.

In addition to the regression coefficients, the Pearson (linear) **correlation coefficient** is often calculated $\boldsymbol{R}(\hat{\boldsymbol{y}}_i,\boldsymbol{y}_i)$:

$$R = \frac{\sum_{i=1}^{N}(y_i - \overline{y})(\hat{y}_i - \overline{\hat{y}})}{(N-1)\sigma_y \sigma_{\hat{y}}}$$

where

$$\sigma_y = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})^2}\,, \quad \sigma_{\hat{y}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\hat{y}_i - \overline{\hat{y}})^2}$$

Post-training analysis

Regression (*fitting*) → **Correlation**

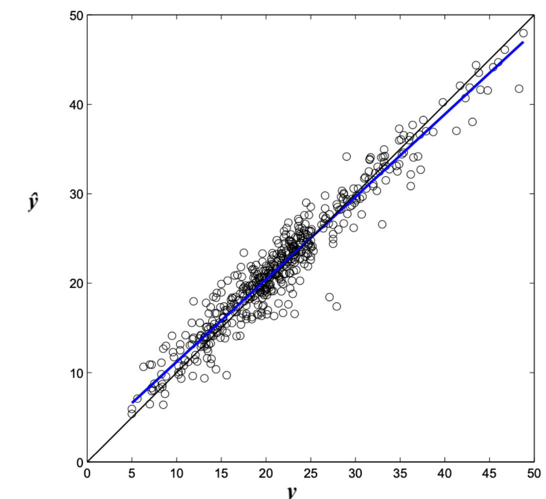
The R coefficient generally varies from -1 (*negative correlation*) to 1 (*positive correlation*).

If $R = 1$, then all the data will fall exactly on the regression line.

If $R = 0$, then the data are randomly dispersed away from the regression line.

For the data in the figure, we have $R = 0{,}965$: we can observe that the data do not fall exactly on the regression line, but the variation is relatively small.

The square of the correlation coefficient, R^2, represents the *proportion of the variability* in a data set that is "captured" by linear regression, and is also called **coefficient of determination**. In practice, it measures the percentage of variability of $\hat{y}$ explained by the variability of y. For the data in the figure, $R^2 = 0{,}931$.



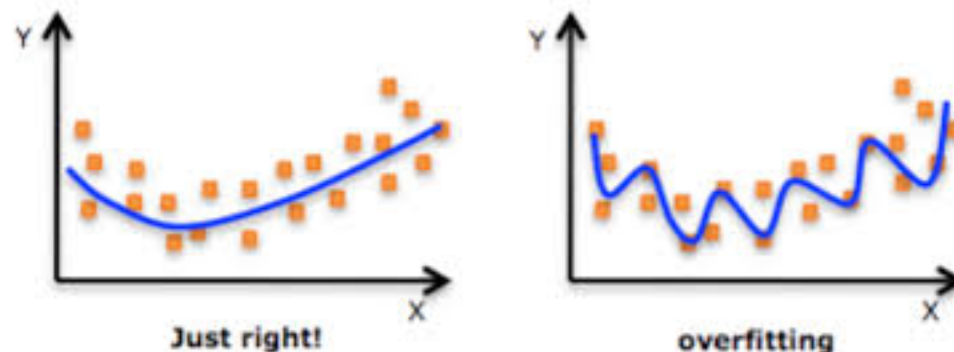
Post-training analysis

Regression (*fitting*) → **Overfitting**

When the values of R or R^2 are significantly **less than 1**, then the neural network has not done a good job of approximating the underlying function. Careful analysis of the scatter plot (*scatterplot*) can be helpful in determining *fitting* problems.

Recall that the original dataset is divided into *training*, *validation* (if using early stopping), and *testing*. Regression analysis should be performed on each subset individually, as well as on the entire data set. Differences between subsets would highlight overfitting or extrapolation.

For example, if the training set shows a very good fit, but the validation and testing results are poor, this indicates ***overfitting****. In this case, we could reduce the size of the neural network (hidden neurons) and repeat the training.*



Post-training analysis

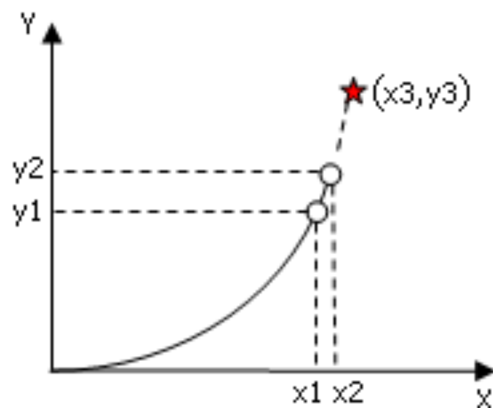
Regression (*fitting*) → **Extrapolation**

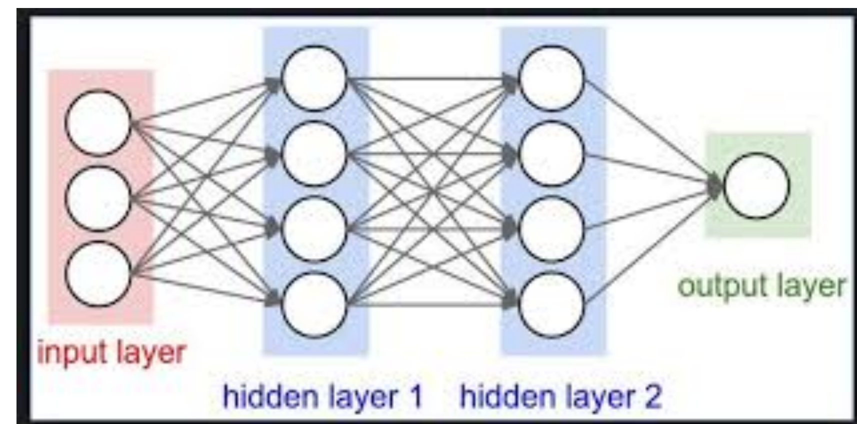
If the training and validatioin results are good, but the test results are poor, then this could indicate **extrapolation** (test data falls outside of the training and validation data). In this case, we need to provide more training and validation data for the network.

If the results for all three datasets are poor, the ***number of neurons*** in the network ***may need to be increased***.

Another choice is to ***increase the number of hidden layers*** in the network.

If you start with a single hidden layer and the results are poor, then a second hidden layer might be useful.. First, try more neurons in the single hidden layer, and then increase the number of layers if the performance is not satisfactory.





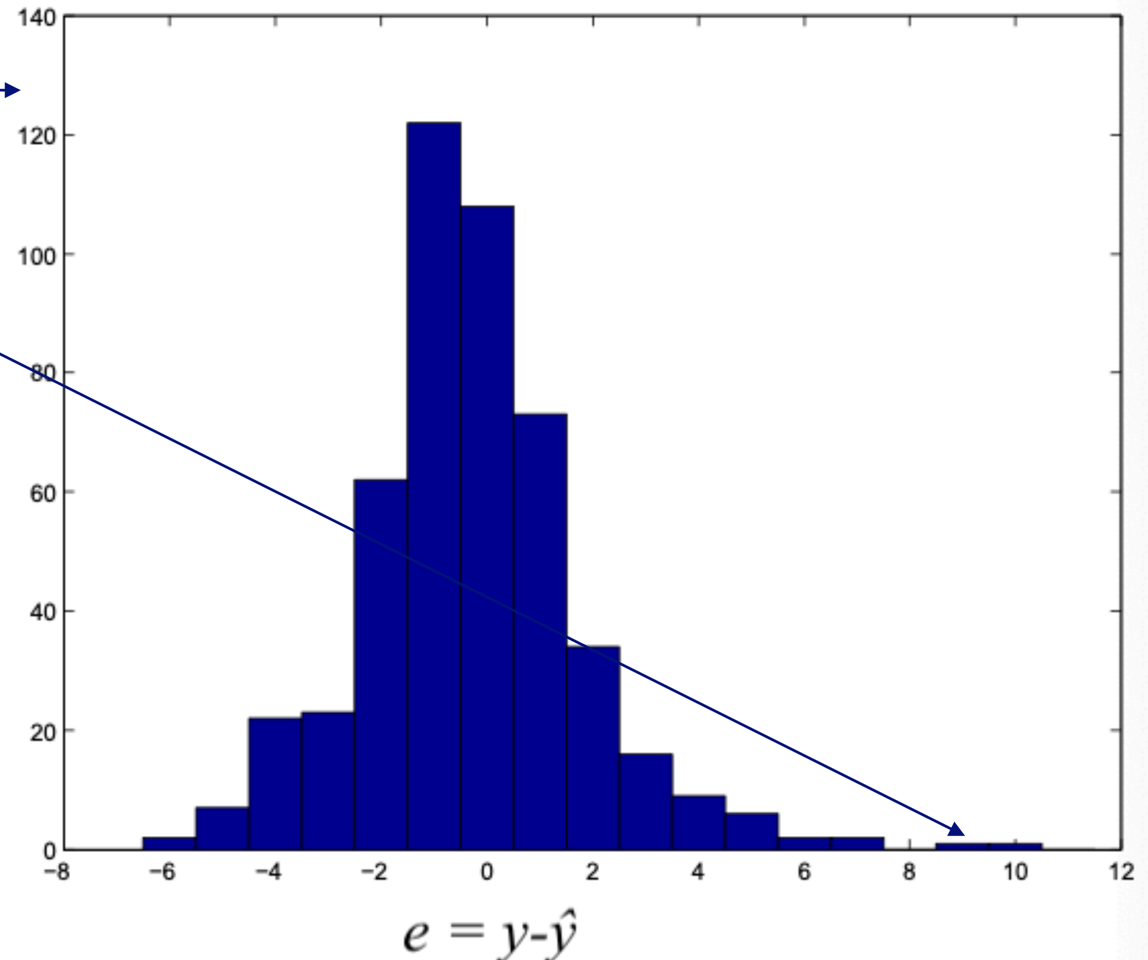
Post-training analysis

Regression (*fitting*) → **Error histogram**

In addition to the regression plot (*scatterplot*), another tool that can identify outliers is the **error histogram**, as shown in Figure.

The vertical axis represents the *number of errors* that fall within each interval of the horizontal axis.

Here we can see that two errors are greater than 8: these represent the same two errors identified as outliers in the previous scatterplot.



Post-training analysis

Classification → **Confusion matrix**

For ***classification*** (*pattern recognition*) problems, regression analysis is not as useful as for fitting problems because the target values are discrete.

However, there is a similar tool: the **confusion matrix**, a table whose columns represent the *target class* and whose rows represent the *output class* of the network.

The figure shows an example of a confusion matrix with 447 observations. There are 54 inputs that belong to class 1 correctly classified as class 1, and 358 inputs from class 2 correctly classified as class 2.

The correctly classified inputs are shown in the main diagonal of the confusion matrix. Cells outside the main diagonal show the incorrectly classified inputs.

*The bottom left cell shows that 10 class 1 inputs were misclassified by the network as class 2. If class 1 is considered a positive result, then the bottom left cell represents the "**false negatives,**" which are also called type II errors.*

*The top right cell shows that 25 class 2 inputs were misclassified as class 1. This error is called a "**false positive**" or type I error.*

predicted class \ actual class	Positive (1)	Negative (2)
Positive (1)	54 *true positive*	25 *false positive*
Negative (2)	10 *false negative*	358 *true negative*

Post-training analysis

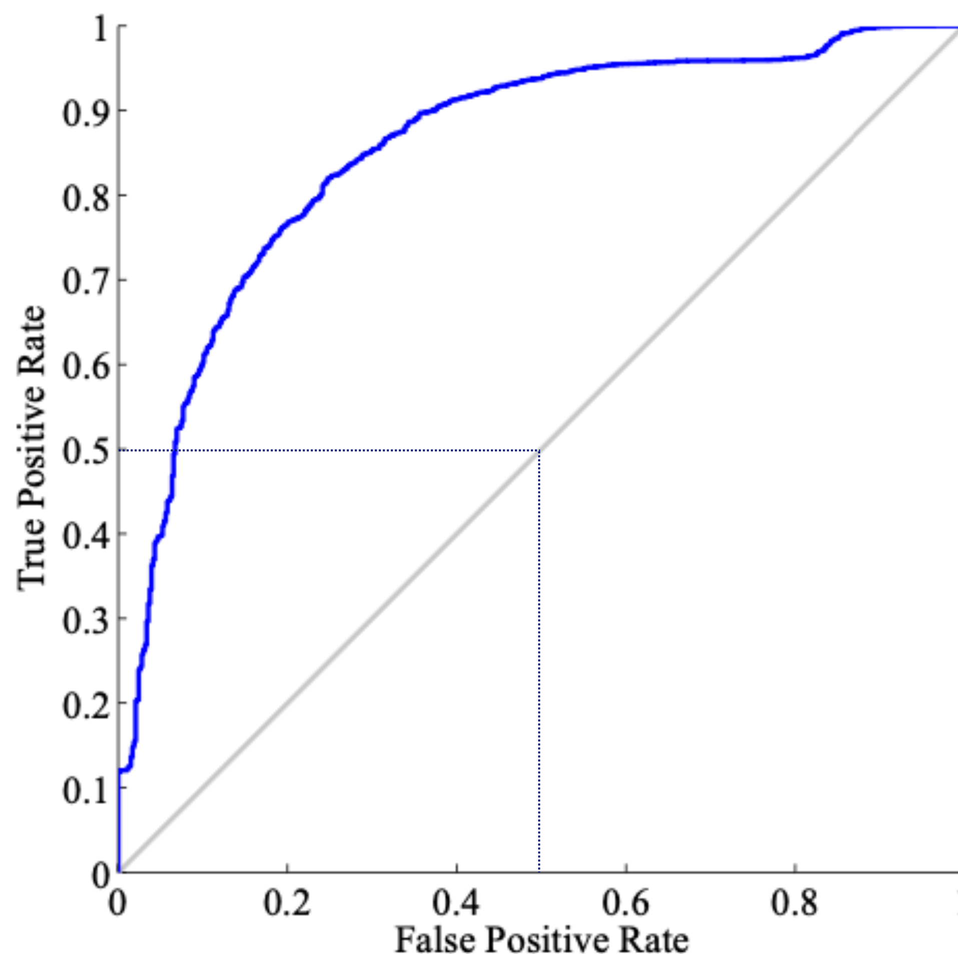
Classification → **ROC (Receiver Operating Characteristic) curve**

Another useful tool for analyzing a network for pattern recognition is the **ROC** (*Receiver Operating Characteristic*) **curve**.

To create this curve, we take the output of the network and compare it to a threshold ranging from -1 to +1 (assuming a tansig *transfer function in the last layer). Inputs that produce values above the threshold are considered to belong to Class 1, while those with values below the threshold are considered to belong to Class 2.*

For each threshold value, we count the fractions of false positives (False Positive Rate) *and true positives* (True Positive Rate) *out of the total data under consideration. This pair of numbers produces a point on the ROC curve. As the threshold changes, we plot the complete curve, as shown in the figure.*

The ideal point for the ROC curve to pass through would be (0,1)*, which would correspond to no false positives and all true positives. A poor ROC curve would represent a random guess, which is represented by the diagonal line in the figure that passes through the point* (0.5,0.5).



Post-training analysis

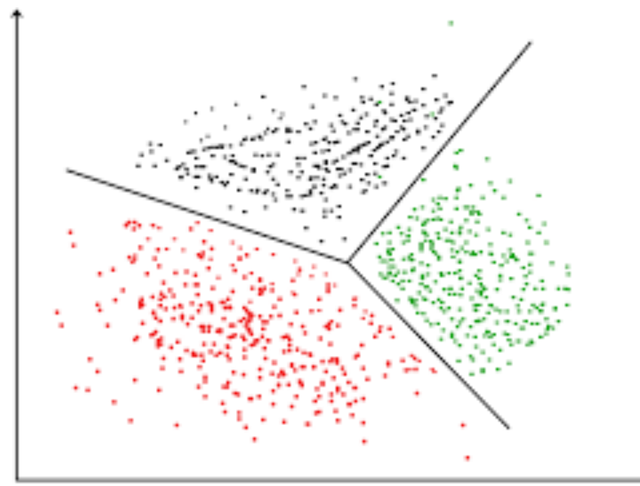
Clustering

The SOM network architecture is the one most commonly used for **clustering**. There are several *measures of SOM network performance*.

One is the *quantization error*, corresponding to the average distance between each input vector and the nearest prototype vector. It measures the "resolution" of the map, and can be made artificially small if a large number of neurons are used. If there are as many neurons as there are input vectors in the dataset, then the quantization error could shrink to zero: this would represent overfitting. If the number of neurons is not significantly less than the number of input vectors, then the quantization error is not significant.

Another measure of SOM network performance is the *topographic error*.

This is the proportion of all input vectors for which the two closest prototype vectors are not close in the feature map topology.



The topographic error measures the conservation of topology.

In a well-trained SOM, prototypes that are close in topology should also be close in input space. In this case, the topographic error should be zero.

Post-training analysis

Forecasting

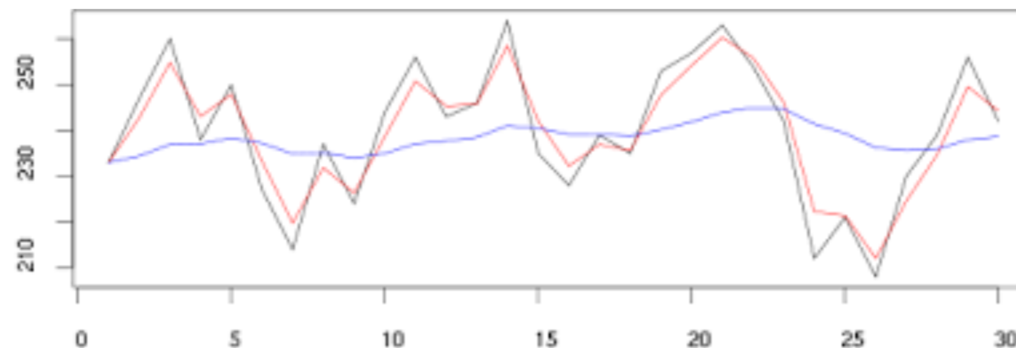
As seen above, one of the applications of neural networks is the ***prediction of future values*** of a time series, for which dynamic networks are used.

There are two important concepts related to the analysis of a trained predictive network:

1. Prediction errors should not be correlated over time;
2. Prediction errors should not be correlated with the input sequence.

In fact, if the prediction errors were correlated over time, then we would be able to predict the prediction errors and, thus, improve our original prediction.

Furthermore, if the prediction errors were correlated with the input sequence, we would also be able to use this correlation to predict the errors.



Post-training analysis

Overfitting and extrapolation

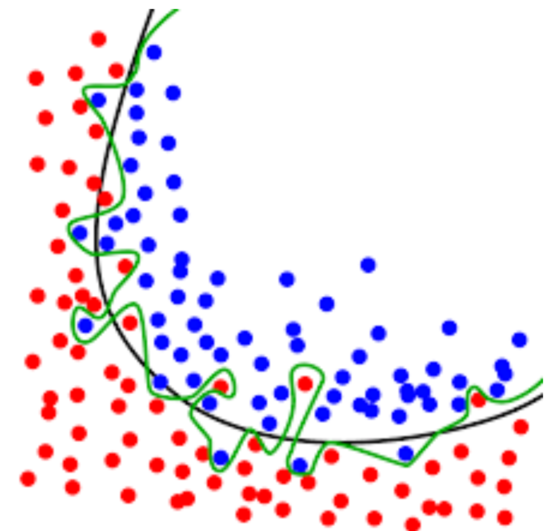
Recall that the total dataset is normally divided into three parts: *training*, *validation* (optional) and *test*.

The *training* set is used to calculate gradients and to determine weight updates.

The *validation* set is used to stop training before overfitting occurs (if regularization is used, it can be merged with the training set).

The *test* set is used to evaluate the future performance of the network and thus measures its quality. If, after a network has been trained, the performance of the test set is not adequate, there are usually four possible causes:

1. the network has reached a **local minimum**;
2. the network does not have enough neurons to "adapt" to the data (**low fitting**);
3. the network is oversized (**overfitting**);
4. the network is "**extrapolating**".



Post-training analysis

Overfitting and extrapolation

The **local minimum** problem can almost always be overcome by re-training the network using random sets of initial weights (usually 5 to 10): the network with the minimum error usually represents a global minimum.

The other three problems can generally be distinguished by analyzing the errors of the *training*, *validation* and *test* sets.

For example, if the validation error is much larger than the training error, it is likely that ***overfitting*** has occurred. Even if we use *early stopping*, it is possible to have some overfitting if training occurs too quickly. In this case, we can use a slower training algorithm to re-train the network.

If the training, validation and testing errors are all of similar size, but the errors are too large, it is likely that the network is not powerful enough to fit the data (**low fitting**). In this case, we should increase the number of neurons in the hidden layer and re-train the network.

Post-training analysis

Overfitting and extrapolation

If the training and validation errors are of similar size, but the testing errors are significantly larger, then the network is probably **extrapolating**.

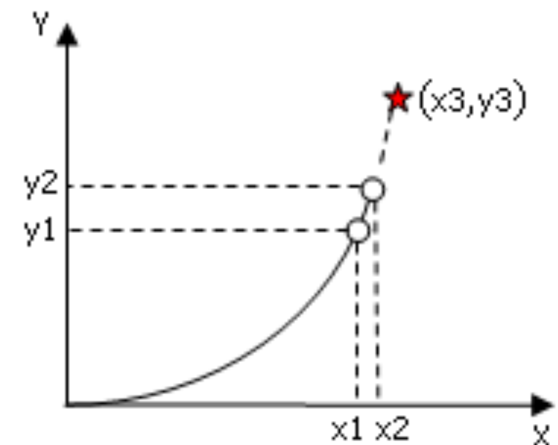
This indicates that the test data is outside the range of the training and validation data.

In this case, **we need more data**. You can merge the test data with the training and validation data and then select new test data. You should continue adding test data until the results on all three data sets are similar.

If the training, validation and testing errors are similar and the errors are small enough, then **we can use the obtained network**.

However, we still need to pay attention to the possibility of extrapolation. Indeed, if an input to the network is provided outside the range of the training data, we will have extrapolation.

It is difficult to guarantee that the training data encompasses all future uses of a neural network ...



Post-training analysis

Sensitivity analysis

After a multilayer network has been trained, it is often useful to evaluate the importance of each element (***feature***) in the input vector. If we are able to determine that a given feature is not important, then we can eliminate it. This can simplify the network, reduce the amount of computation, and help prevent overfitting.

There is no single method that can with certainty determine the importance of each feature, but a sensitivity analysis can be useful in this regard. The analysis calculates the derivatives of the network response with respect to each feature. If a derivative is small, then that feature can be eliminated from the input vector.

Since the multilayer network is not linear, the derivative of the network output with respect to an input feature will not be constant. For each input vector in the training set, the derivatives will be different. For this reason, we cannot use a single derivative to determine sensitivity.

One option would be to take the average of the absolute derivatives, or the RMSs of the derivatives, over the entire training set. If some of these derivatives are much smaller than the maximum derivative, then we can consider removing the corresponding features.

After removing the potentially irrelevant inputs, we re-train the network and compare the performance with the original network. If the performance is similar, we accept the simplified network.