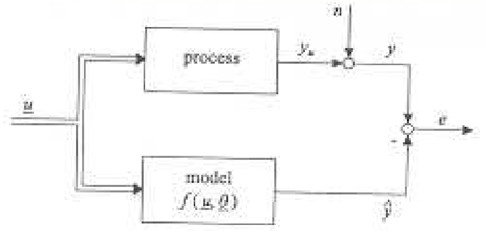

Fig. 2.1. Process and model

the deviation (error) of the pole from the desired upright position is utilized. On the other hand, if only the information on whether the run was successful or a failure is exploited, a reinforcement problem arises. Clearly, it is always advantageous to exploit all available information. Therefore, reinforcement learning techniques are not addressed in this book. They mainly become interesting for strategy learning where no desired output and consequently no error signal is available for each step.

For unsupervised learning methods only input data is utilized. The objective of unsupervised methods is grouping or clustering of data. The exploited information is the input data distribution. Unsupervised learning techniques are primarily applied for data preprocessing. They are discussed in Chap. 6.

This chapter gives an introduction, and a brief overview of the most important optimization techniques. The focus is on the application of these techniques to modeling and identification. Most of the methods described are parameter optimization techniques. In Fig. 2.1 the basic concept is depicted from a modeling point of view. A model $f(\cdot)$ maps the inputs gathered in the input vector $\underline{u}$ to the scalar output y. The model is parameterized by a set of n parameters gathered in the parameter vector $\underline{\theta}$ such that $\hat{y} = f(\underline{u}, \underline{\theta})$. The goal of a parameter optimization technique is to find the "best" approximation $\hat{y}$ of the measured output y, which may be spoiled with noise n, by adapting the parameter vector $\underline{\theta}$. A more precise definition of "best" will be given in Sect. 2.3. It is helpful to look at this problem as a search for the optimal point in an n-dimensional parameter space spanned by the parameter vector $\underline{\theta}$. The Chaps. 3, 4, and 5 address such parameter optimization techniques.

Besides these parameter optimization techniques, so-called structure optimization techniques represent another category of methods. They deal with the problem of searching an optimal model structure, e.g., the optimal kind of function $f(\cdot)$ and the optimal number of parameters. These issues are discussed in Chap. 7 and partly addressed also in Chap. 5.

This chapter is organized as follows. First, a brief overview on the supervised learning methods is given. Section 2.2 gives an illustration of these techniques by means of a humorous analogy. In Sects. 2.3 and 2.4 some definitions of loss functions for supervised and unsupervised learning are given.

2.1 Overview of Optimization Techniques

The supervised learning techniques can be divided into three classes: the linear, the nonlinear local, and the nonlinear global optimization methods. Out of these three, linear optimization techniques are the most mature and most straightforward to apply. They are discussed in Chap. 3. Nonlinear local optimization techniques, summarized in Chap. 4, are a well understood field of mathematics, although active research still takes place, especially in the area of constrained optimization. By contrast, many questions remain unresolved for nonlinear global optimization techniques, and therefore this is quite an active research field at the moment; see Chap. 5. Figure 2.2 illustrates the relationship between the discussed supervised optimization techniques.

2.2 Kangaroos

Plate and Sarle give the following wonderful description of the most common nonlinear optimization algorithms with respect to neural networks (NN) in [307] (comments in brackets by the author are related to Chap. 4):

Training a NN is a form of numerical optimization, which can be linked to a kangaroo searching the top of Mt. Everest. Everest is the global optimum, *the highest mountain in the world, but the top of any other really tall mountain such as K2 (a good* local optimum*) would be satisfactory. On the other hand, the top of a small hill like Chapel Hill, NC, (a bad local optimum) would not be acceptable.*

This analogy is framed in terms of maximization, while neural networks are usually discussed in terms of minimization of an error measure such as the least squares criterion, but if you multiply the error measure by -1, *it works out the same. So in this analogy, the higher the altitude, the smaller the error.*

The compass directions represent the values of synaptic weights [parameters] *in the network. The north/south direction represents one weight, while the east/west direction represents another weight. Most networks have more than two weights, but representing additional weights would require a multi-dimensional landscape, which is difficult to visualize. Keep in mind that when you are training a network with more than two weights, everything gets more complicated.*

Initial weights are usually chosen randomly, which means that the kangaroo is dropped by parachute somewhere over Asia by a pilot who has lost the map. If you know something about the scales of input, you may be able to give the pilot adequate instructions to get the kangaroo to land near the Himalayas. However, if you make a really bad choice of distributions for the initial weights, the kangaroo may plummet into the Indian Ocean and drown.

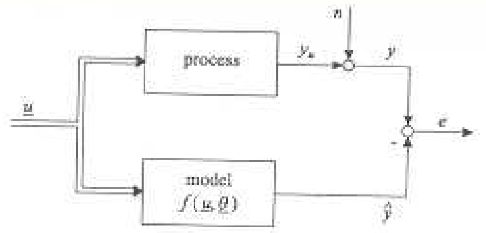

Fig. 2.1. Process and model

the deviation (error) of the pole from the desired upright position is utilized. On the other hand, if only the information on whether the run was successful or a failure is exploited, a reinforcement problem arises. Clearly, it is always advantageous to exploit all available information. Therefore, reinforcement learning techniques are not addressed in this book. They mainly become interesting for strategy learning where no desired output and consequently no error signal is available for each step.

For unsupervised learning methods only input data is utilized. The objective of unsupervised methods is grouping or clustering of data. The exploited information is the input data distribution. Unsupervised learning techniques are primarily applied for data preprocessing. They are discussed in Chap. 6.

This chapter gives an introduction, and a brief overview of the most important optimization techniques. The focus is on the application of these techniques to modeling and identification. Most of the methods described are parameter optimization techniques. In Fig. 2.1 the basic concept is depicted from a modeling point of view. A model $f(\cdot)$ maps the inputs gathered in the input vector $\underline{u}$ to the scalar output y. The model is parameterized by a set of n parameters gathered in the parameter vector $\underline{\theta}$ such that $\hat{y} = f(\underline{u}, \underline{\theta})$. The goal of a parameter optimization technique is to find the "best" approximation $\hat{y}$ of the measured output y, which may be spoiled with noise n, by adapting the parameter vector $\underline{\theta}$. A more precise definition of "best" will be given in Sect. 2.3. It is helpful to look at this problem as a search for the optimal point in an n-dimensional parameter space spanned by the parameter vector $\underline{\theta}$. The Chaps. 3, 4, and 5 address such parameter optimization techniques.

Besides these parameter optimization techniques, so-called structure optimization techniques represent another category of methods. They deal with the problem of searching an optimal model structure, e.g., the optimal kind of function $f(\cdot)$ and the optimal number of parameters. These issues are discussed in Chap. 7 and partly addressed also in Chap. 5.

This chapter is organized as follows. First, a brief overview on the supervised learning methods is given. Section 2.2 gives an illustration of these techniques by means of a humorous analogy. In Sects. 2.3 and 2.4 some definitions of loss functions for supervised and unsupervised learning are given.

2.1 Overview of Optimization Techniques

The supervised learning techniques can be divided into three classes: the linear, the nonlinear local, and the nonlinear global optimization methods. Out of these three, linear optimization techniques are the most mature and most straightforward to apply. They are discussed in Chap. 3. Nonlinear local optimization techniques, summarized in Chap. 4, are a well understood field of mathematics, although active research still takes place, especially in the area of constrained optimization. By contrast, many questions remain unresolved for nonlinear global optimization techniques, and therefore this is quite an active research field at the moment; see Chap. 5. Figure 2.2 illustrates the relationship between the discussed supervised optimization techniques.

2.2 Kangaroos

Plate and Sarle give the following wonderful description of the most common nonlinear optimization algorithms with respect to neural networks (NN) in [307] (comments in brackets by the author are related to Chap. 4):

Training a NN is a form of numerical optimization, which can be linked to a kangaroo searching the top of Mt. Everest. Everest is the global optimum, *the highest mountain in the world, but the top of any other really tall mountain such as K2 (a good* local optimum*) would be satisfactory. On the other hand, the top of a small hill like Chapel Hill, NC, (a bad local optimum) would not be acceptable.*

This analogy is framed in terms of maximization, while neural networks are usually discussed in terms of minimization of an error measure such as the least squares criterion, but if you multiply the error measure by −1, it works out the same. So in this analogy, the higher the altitude, the smaller the error.

The compass directions represent the values of synaptic weights [parameters] *in the network. The north/south direction represents one weight, while the east/west direction represents another weight. Most networks have more than two weights, but representing additional weights would require a multi-dimensional landscape, which is difficult to visualize. Keep in mind that when you are training a network with more than two weights, everything gets more complicated.*

Initial weights are usually chosen randomly, which means that the kangaroo is dropped by parachute somewhere over Asia by a pilot who has lost the map. If you know something about the scales of input, you may be able to give the pilot adequate instructions to get the kangaroo to land near the Himalayas. However, if you make a really bad choice of distributions for the initial weights, the kangaroo may plummet into the Indian Ocean and drown.

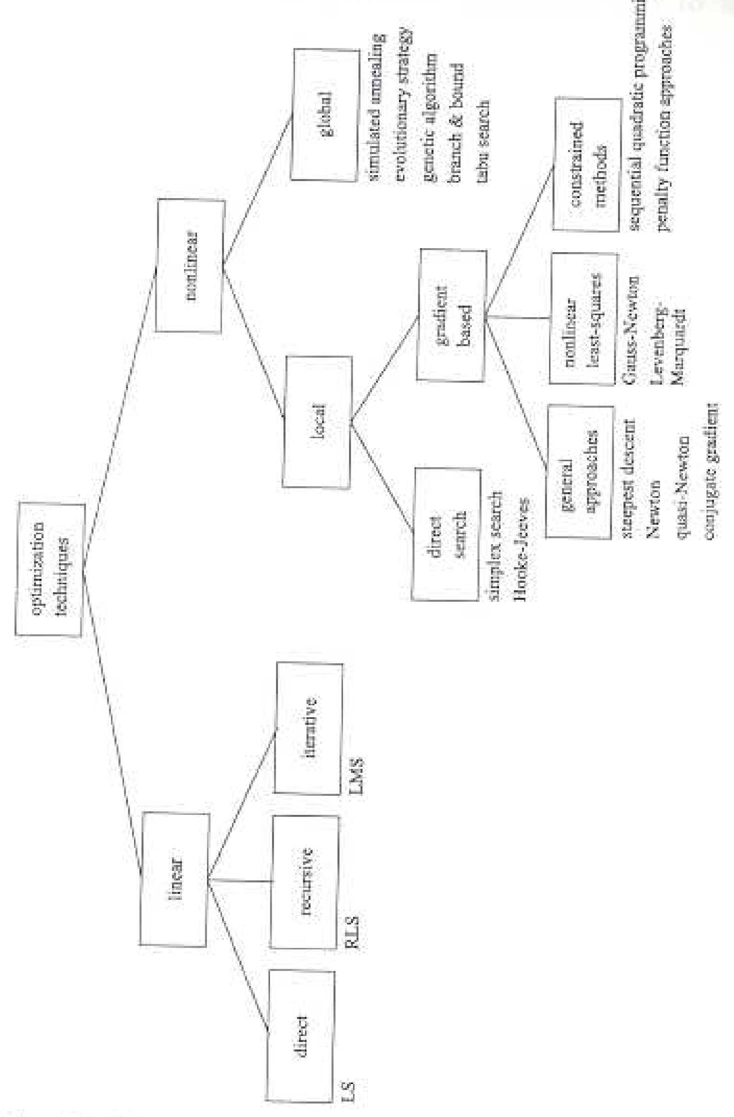

Fig. 2.2. Overview of linear and nonlinear optimization techniques

With Newton-type (second order) algorithm [with fixed step size $\eta = 1$], *the Himalayas are covered with fog, and the kangaroo can only see a little way around her location* [first and second order derivative information]. *Judging from the local terrain, the kangaroo makes a guess about where the top of the mountain is, assuming that the mountain has a nice, smooth, quadratic shape. The kangaroo then tries to leap all the way to the top in one jump.*

Since most mountains do not have a perfect quadratic shape, the kangaroo will rarely reach the top in one jump. Hence, the kangaroo must iterate, *i.e., jump repeatedly as previously described until she finds the top of the mountain. Unfortunately, there is no assurance that this mountain will be Everest.*

In a stabilized Newton algorithm [with variable step size η], *the kangaroo has an altimeter, and if the jump takes her to a lower point, she backs up to where she was and takes a shorter jump. If ridge stabilization* [the Levenberg-Marquardt idea] *is used, the kangaroo also adjusts the direction of her jump to go up a steeper slope. If the algorithm isn't stabilized, the kangaroo may mistakenly jump to Shanghai and get served for dinner in a Chinese restaurant* [divergence].

In steepest ascent with line search, the fog is very dense, and the kangaroo can only tell, which direction leads up most steeply [only first order derivative information]. *The kangaroo hops in this direction until the terrain starts going down. Then the kangaroo looks around again for the new steepest ascent direction and iterates.*

Using an ODE (ordinary differential equation) solver is similar to steepest ascent, except that kangaroo crawls on all fives to the top of the nearest mountain, being sure to crawl in steepest direction at all times.

The following description of conjugate gradient methods was written by Tony Plate (1993):

> *The environment for conjugate gradient search is just like that for steepest ascent with line search – the fog is dense and the kangaroo can only tell, which direction leads up. The difference is that the kangaroo has some memory of the direction it has hopped in before, and the kangaroo assumes that the ridges are straight (i.e., the surface is quadratic). The kangaroo chooses a direction to hop that is upwards, but that does not result in it going downwards in the previous directions it has hopped in. That is, it chooses an upwards direction, moving along which will not undo the work of previous steps. It hops upwards until the terrain starts going down again, then chooses another direction.*

In standard backprop, the most common NN training method, the kangaroo is blind and has to feel around on the grounds to make a guess about, which way is up. If the kangaroo ever gets near the peak, she may jump back and forth across the peak without ever landing on it. If you use a decaying step size, the kangaroo gets tired and makes smaller and smaller hops, so if she ever gets near the peak she has a better chance to actually landing on it *before the Himalayas erode away. In backprop with momentum the kangaroo has poor traction and can't make sharp turns. With online training, there are frequent earthquakes, and mountains constantly appear and disappear. This makes it difficult for the blind kangaroo to tell whether she has ever reached the top of a mountain, and she has to take small hops to avoid falling into the gaping chasms that can open up at any moment.*

Notice that in all the methods discussed so far, the kangaroo can hope at best to find the top of a mountain close to where she starts. In other words these are local ascent *methods. There's no guarantee that this mountain will be Everest, or even a very high mountain. Many methods exist to try to find the global optimum.*

In simulated annealing, the kangaroo is drunk and hops around randomly for a long time. However, she gradually sobers up and the more sober she is, the more likely she is to hop up hill [temperature decreases according to the annealing schedule].

In a random multi-start method, lots of kangaroos are parachuted into the Himalayas at random places. You hope at least one of them will find Everest.

A genetic algorithm begins like random multi-start. However, these kangaroos do not know that they are supposed to be looking for the top of a mountain. Every few years, you shoot the kangaroos at low altitudes and hope that the ones that are left will be fruitful, multiply, and ascend. Current research suggests that fleas may be more effective than kangaroos in genetic algorithms, since their faster rate of reproduction more than compensates for their shorter hops [crossover is more important than mutation].

A tunneling algorithm can be applied in combination with any local ascent method but requires divine intervention and a jet ski. The kangaroo first finds the top of any nearby mountain. Then the kangaroo calls upon her deity to flood the earth to the point that the waters just reach the top of the current mountain. She get on her ski, goes off in search of a higher mountain, and repeats the process until no higher mountains can be found.

2.3 Loss Functions for Supervised Methods

Before starting with any optimization algorithm, a criterion needs to be defined that is the exact mathematical description of what has to be optimized. In supervised learning the error $e(i)$ is usually computed as the difference between the measured process output $y(i)$ and the model output $\hat{y}(i)$ for a given number N of training data samples $i = 1, \dots, N$: the so-called training data set (see Fig. 2.3). Usually the measured process output $y(i)$ is corrupted with noise $n(i)$. So the actually desired output $y_u(i)$ is unknown. The most common choice for a criterion is the sum of squared errors or its square root,

$$I(\underline{\theta}) = \sum_{i=1}^{N} e^2(i) \qquad \text{with} \quad e(i) = y(i) - \hat{y}(i) \,. \tag{2.1}$$

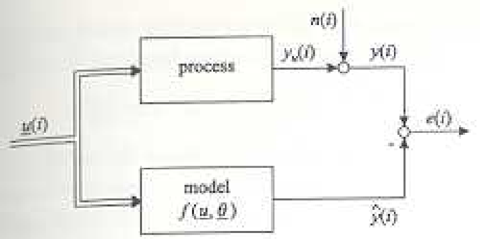

Fig. 2.3. Process and model for data sample i

Since the objective is to find the minimum of this function $I(\underline{\theta})$, it is called a *loss function*. Linear and nonlinear optimization problems applying this special kind of loss function are called *least squares (LS)* and *nonlinear least squares (NLS)* problems. The reasons for the popularity of this loss function are the following. To achieve the minimum of a loss function, its gradient has to be equal to zero. With (2.1) this leads to a linear equation system if the error itself is linear in the unknown parameters. Thus, for linear parameters the error sum of squares leads to an easy-to-solve linear optimization problem; see Chap. 3. Another property of this loss function is the quadratic scaling of the errors, which favors many small errors over a few larger ones. This property can often be seen as an advantage. Note, however, that this property makes the sum of squared errors sensitive to outliers.

The sum of squared errors loss function can be extended by weighting the contribution of each squared error with a factor, say q_i,

$$I(\underline{\theta}) = \sum_{i=1}^{N} q_i \, e^2(i) \,. \tag{2.2}$$

This offers the additional advantage that knowledge about the relevance of or confidence in each data sample i can be incorporated in (2.2) by selecting the q_i appropriately. Problems applying this type of criterion are called *weighted least squares (WLS)* problems.

Even more general is the following loss function definition:

$$I(\underline{\theta}) = \left(\sum_{i=1}^{N} q_i \, \|e(i)\|^p \right)^{1/p} \,. \tag{2.3}$$

Besides $p = 2$ common choices for p are $p = 1$ (that is, the sum of absolute errors) and $p = \infty$ (that is, the maximum error)[1]. Figure 2.4 shows the fit of a second order polynomial through 11 data samples by minimizing three different loss functions of type (2.3) with $p = 1, 2$, and ∞, respectively.

[1] Note that taking the pth root in (2.3) just scales the absolute value of the loss function. It is important, however, to let (2.3) converge to the maximum error for $p \to \infty$.