2 Supervised Learning

In machine learning, *supervised learning* has come to mean the process of adjusting a system so it produces specified outputs in response to specified inputs. It is often posed as a function approximation problem (figure 2.1). Given training data consisting of pairs of input patterns, $\mathbf{x}$, and corresponding desired outputs or *targets*, t, the goal is to find a function $y(\mathbf{x})$ that matches the desired response for each training input. The functional relationship between the input patterns and target outputs is usually unknown (otherwise different methods would be used) so the idea is to start with a system flexible enough to implement many functions and adjust it to fit the given data.

"Training" refers to the adaptation process by which the system "learns" the relationship between the inputs and targets. This is often a repetitive incremental process guided by an optimization algorithm (figure 2.2). The process is "supervised" in the sense that an external "teacher" must specify the correct output for each and every input pattern. In some cases, the teacher is a person who specifies the correct class for each pattern. In other cases, it may be a physical system whose behavior we want to model.

In this book, the learning system is an artificial neural network. During training, each input pattern is presented and propagated through the network to produce an output. Unless the network is perfectly trained, there will be differences between the actual and desired outputs. The real-world significance of these deviations depend on the application and is measured by an *objective function* whose output rates the quality of the network's response. (The terms "cost function" and "error function" are also used.) The overall goal is then to find a system that minimizes the total error for the given training data.

When defined in this way, training becomes a statistical optimization problem and there are a number of interacting factors to be considered:

- Variable selection and representation. What information should be presented to the network and in what form?
- Selection and preparation of training data.
- Model selection. What structure should the network have?
- Choice of error function. How is network performance graded?
- Choice of optimization method. The network output is a function of its parameters (weights). How should parameters be adjusted to minimize the error?
- Prior knowledge and heuristics. If we know useful rules or heuristics (which may not be learnable from available data) can we somehow insert them into the system? Can we make the system favor particular sorts of solutions?
- Generalization. How well does the network *really* work? Did it learn what we intended or did it simply memorize the training set or find a set of tricks that work on this data but not on others?

These issues are discussed at length in following chapters.

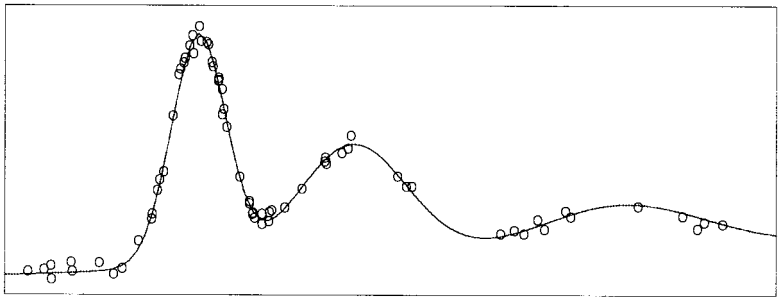

Figure 2.1
Supervised learning is often viewed as function approximation problem. Given a set $\{(\mathbf{x}_i, t_i)\}$, $i = 1 \ldots M$, of training pairs with inputs $\mathbf{x}_i$ and target outputs t_i, the goal is to find a function $f(\mathbf{x})$ that captures the input-output relationships illustrated in the training examples, $f(\mathbf{x}_i) \approx t_i$. If the search is successful, the new function can then be used to estimate the correct output for new points not in the original training set. Ideally, the functional form may also be more compact and faster to evaluate.

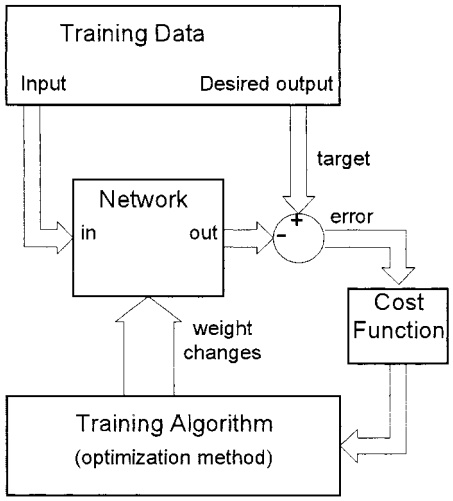

Figure 2.2
Supervised learning model. In supervised learning, a desired output is specified for every pattern encountered during training. Differences between the network output and training target are treated as errors to be minimized by the training algorithm.

2.1 Objective Functions

As noted, the role of the objective function is to measure how well the network performs the intended task. The function defines the difference between good or bad performance and thus guides the search for a solution. It has a fundamental effect on the outcome so it is important to choose a function that accurately reflects our design goals.

A few standard error functions are commonly used. The most common is the sum of squared errors (SSE),

$$E_{SSE} = \sum_p \sum_i (t_{pi} - y_{pi})^2 \tag{2.1}$$

where p indexes the patterns in the training set, i indexes the output nodes, and t_{pi} and y_{pi} are, respectively, the target and actual network output for the ith output node on the pth pattern. This is the sum of the squared errors on each training pattern. The mean-squared-error (MSE)

$$E_{MSE} = \frac{1}{PN} E_{SSE} \tag{2.2}$$

normalizes E_{SSE} for the number of training patterns P and the number of network outputs N. The logarithmic or cross-entropy error function

$$E_{log} = \sum_p \sum_i t_{pi} \ln y_{pi} + (1 - t_{pi}) \ln(1 - y_{pi}) \tag{2.3}$$

is often used for classification problems where the network output is interpreted as the probability that the input pattern belongs to a certain class. Here y_{pi} is the estimated probability that pattern p belongs to class i and $t_{pi} \in \{0, 1\}$ is the target. Other functions have been developed for various applications.

Each of these functions carries assumptions including, among others, assumptions about the distribution of fitting errors that arise given the model and the data. In a statistical setting the mean squared error function, for example, corresponds to a maximum likelihood model with the assumption that errors have a Gaussian distribution (see section 15.2). The logarithmic error function corresponds to a classification model and the assumption of a binomial error distribution. Reasonable performance can be expected if these assumptions match reality but poor performance may result if the assumptions are not met. More details can be found in [328], as well as numerous statistics texts.

These standard functions are convenient to use and are well-understood. Advantages include easy differentiability and independence. (All numerical deviations of equal magni-

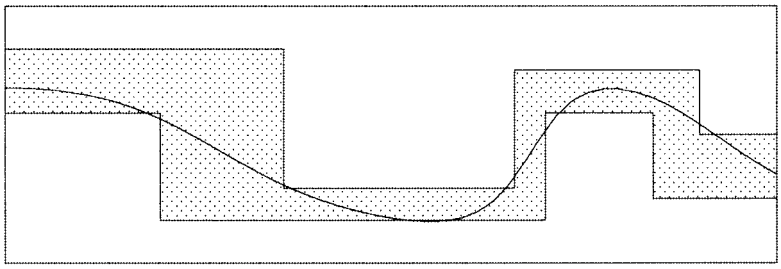

Figure 2.3
Supervised learning can be applied to many different error functions. The figure illustrates a piecewise linear error function with upper and lower tolerance limits; the error is zero when $f(x)$ is within the limits. Functions like this are sometimes useful in engineering applications.

tude have equal costs which do not depend on the input pattern, the sizes of other errors, the trend of previous errors, and so on.) These properties simplify analysis considerably and allow valuable theoretical study that would not be possible otherwise. In spite of this, more idiosyncratic functions may be useful in applications where errors of similar numerical magnitude may have quite different costs depending on the input pattern and other factors. These considerations are completely application dependent, however, so the standard error functions are used for most discussions.

Figure 2.3 illustrates an error function that falls a bit outside the range of standard models but is still included in the supervised learning model. In this case, the target function has piecewise constant upper and lower tolerance limits; the error is zero when $y(x)$ is within the limits and increases quadratically otherwise. Functions like this are sometimes useful in engineering applications. An application-specific error evaluation function is required and the mathematical analysis is not as clean, but the training procedure is basically the same otherwise.

Penalty Terms In addition to the primary terms that measure fitting errors, the cost function is often augmented with terms reflecting goals or preferences, which are not directly measurable in terms of differences between outputs and targets on a set of patterns. "Penalty terms" may be added to steer the solution in preferred directions or enforce constraints. Some common biases include:

- a preference for simple solutions over complex ones (Occam's razor),
- a preference for smooth continuous solutions over wildly varying or discontinuous solutions, and

• beliefs about the relative probabilities of various solutions (corresponding to prior probabilities in Bayesian methods).

Many of the heuristics discussed later can be viewed as modifications of the basic error function which introduce these types of biases. These hints can be especially useful when training data is limited.

2.2 Alternatives and Extensions

An advantage of the supervised learning model is that it is well-defined. It is detailed enough to be useful but simple enough to be analyzed. Details of specific applications are abstracted away. The model has been criticized as an artificial and limited model of learning, however, amounting to nothing more than nonlinear regression—a way to fit a function to a set of data points. Indeed, in many practical applications neural networks are used mainly for function approximation and nothing more is asked. Perhaps the major limitation is the requirement for a teacher to specify in detail the correct output for each and every input. This is not how people learn to walk, for example.

Obviously, there is much more to learning than function approximation so researchers interested in more realistic learning systems must consider additional factors. The model can be extended in many ways, however, and simplified abstract models like this are likely to be useful as core components in a larger system. Some proposals, for example, surround a supervised learning module with key subsystems designed to translate available information into the detailed signals required by the simplified model. Extensions such as this are fascinating, but beyond the scope of this book. Other abstract models at similar levels of complexity include unsupervised learning and reinforcement learning.

Unsupervised Learning A requirement for supervised learning is presence of a teacher to specify the target output for every input. In unsupervised learning, there is no teacher. The training data is unlabeled and there are no targets. Instead, the system adapts to regularities in the data according to rules implicit in its design. The nature of the regularities found by the system depend on details of its design so the teacher is, in a sense, built into the system. Unsupervised learning is useful because unlabeled data is often more readily available than labeled data.

Some systems extract a set of prototype patterns from the training set; given an input, the most similar prototype is recalled. Parameters of the system determine how similarity is defined. In statistics, unsupervised learning often refers to clustering algorithms or probability density approximation. The k-means algorithm and vector quantization are examples.

Unsupervised learning modules are sometimes used as a component of a supervised learning system. To be useful, the unsupervised model must partition the data in a way that preserves the information needed for supervised learning.

Autoassociative networks are on the borderline between supervised and unsupervised learning. Given an input, the network is trained to reproduce the identical pattern at the output. The network acts as autoencoder, mapping an input pattern to itself. This may seem pointless but if the system is constrained by a bottleneck in a small middle layer, the network is forced to find an efficient internal representation of the pattern that preserves as much information as possible. Ideally, it will strip away nonessentials and reproduce only the significant features of the pattern, perhaps making it more useful for other purposes. Alternatively, the output of the bottleneck layer may be useful in itself as a compressed representation of the input pattern. This is related to principal components analysis (see appendix section B.1).

Reinforcement Learning Reinforcement learning (e.g., [25, 364, 24]) is a more realistic model of low-level learning in humans and animals. Reinforcement learning resembles supervised learning in that there is a defined goal, but the objective is defined more abstractly. Instead of a teacher providing detailed targets for each and every output, the only feedback is a sparse reinforcement signal which grades the system response as "good" or "bad" without providing further details. The reinforcement may be sparse in time as well as space. Game playing is a commonly mentioned example: the outcome of a game of chess is a single win-lose signal rather than a detailed list of which moves should have been made at each step in the game. In general, the system produces outputs that act on an external environment and affect the reinforcement eventually received. The training objective is to maximize the amount of positive reinforcement received over time.

In many reinforcement learning models, the key element is a subsystem which learns to predict the future reinforcement expected given the current network inputs and outputs. If this prediction can be learned accurately, then target signals for supervised training of the action selection module can be derived from changes in the predicted reinforcement.

Supervised Learning with a Distal Teacher As noted, the supervised learning model has been criticized because it puts a heavy burden on the teacher to specify detailed, low-level, target signals for every possible input. The model is not completely unrealistic, however, because there may be higher level targets available from which the low level targets needed for network training can be derived. Supervised learning with a distal teacher [200, 201, 199] is intermediate between regular supervised learning and reinforcement learning (figure 2.4). The system output targets are more informative than in reinforce-

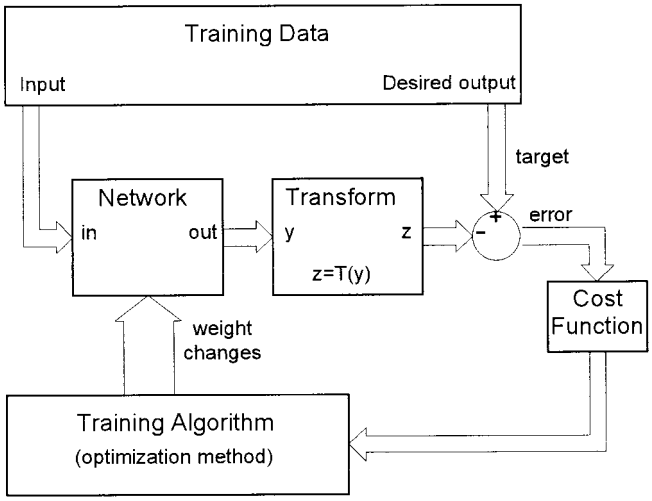

Figure 2.4
In supervised learning with a distal teacher the network output drives another system, T, which produces the final output. This makes the teacher's job easier because targets can be specified at a higher, less detailed, level. When T is a known function, the low-level signals needed for network training can be derived from the high-level errors.

ment learning, but less informative than in regular supervised learning. As in reinforcement learning, the network outputs act as inputs to another system, T, which transforms the network outputs into the final output. When T is well-defined or can be accurately modeled, errors in the overall system-output can be translated backwards to the low level network-output error signals needed for network training. When the overall targets can be specified simply, the teacher's job is simpler.

Using the task of throwing a ball as an analogy, the network outputs are the numerous coordinated muscular actions needed to toss the ball and T is the physics that transform these actions into a result. If the overall goal is to land a basketball in a hoop, the sight of it bouncing off the rim may be a high-level error signal. No coach can tell you exactly when and how to move each individual muscle, but they can provide high-level suggestions in terms you already know how to implement, for example, "put more spin on it." Knowledge of the situation then allows you to translate the high-level suggestion back to individual low-level actions.

Simulation results for a simple robot arm controller are described in [200, 201, 199]. Given inputs representing a position (x, y), the desired network outputs are the joint angles that put the manipulator in this position (figure 2.5). Physical properties of the arm determine the relationship T between the network outputs (joint angle commands) and the

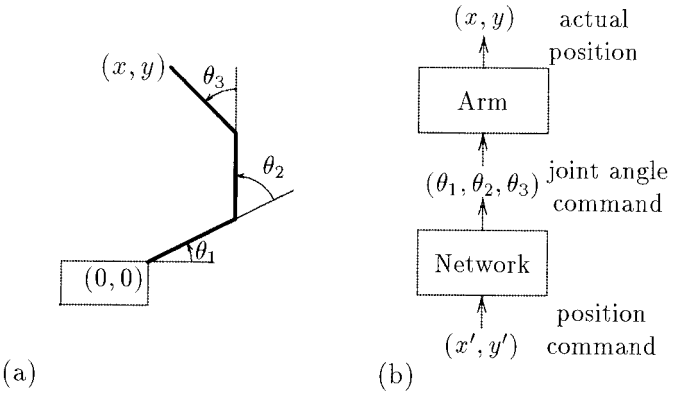

Figure 2.5
Learning with a distal teacher, robot arm example: (a) a robot arm, (b) a neural network translates input commands to joint angles that put the manipulator in the desired position after transformation by the physics of the robot arm.

system output where position errors are measured. The physics of the arm are well-known so the position errors can be translated back to joint-angle error signals needed for training.

Referring ahead a few chapters, it is interesting to note that this model covers the problem of training internal layers of a multilayer network if the first layer is viewed as the network and following layers are viewed as the transform T.