# Appendix A

# Definitions and basic concepts

## A.1 Notation

The following mathematical notation conventions will be used:

- All vectors are column vectors unless otherwise stated, so the dot product $\mathbf{x} \bullet \mathbf{y}$ is equal to $\mathbf{x}^T\mathbf{y}$.
- $\mathbf{x}^{(i)}$ is the $i$'th element of vector $\mathbf{x}$.
- Vector derivatives: for vectors $\mathbf{x}$ (of size $n \times 1$) and $\mathbf{y}$ (of size $m \times 1$), the quantity $\mathrm{d}\mathbf{x}/\mathrm{d}\mathbf{y}$ is a matrix of size $n \times m$:

$$\frac{\mathrm{d}\,\mathbf{x}}{\mathrm{d}\,\mathbf{y}} = \begin{bmatrix} \frac{\mathrm{d}\,\mathbf{x}^{(1)}}{\mathrm{d}\,\mathbf{y}^{(1)}} & \cdots & \frac{\mathrm{d}\,\mathbf{x}^{(1)}}{\mathrm{d}\,\mathbf{y}^{(m)}} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}\,\mathbf{x}^{(n)}}{\mathrm{d}\,\mathbf{y}^{(1)}} & \cdots & \frac{\mathrm{d}\,\mathbf{x}^{(n)}}{\mathrm{d}\,\mathbf{y}^{(m)}} \end{bmatrix} \tag{A.1}$$

  Similarly for $\partial\mathbf{x}/\partial\mathbf{y}$.

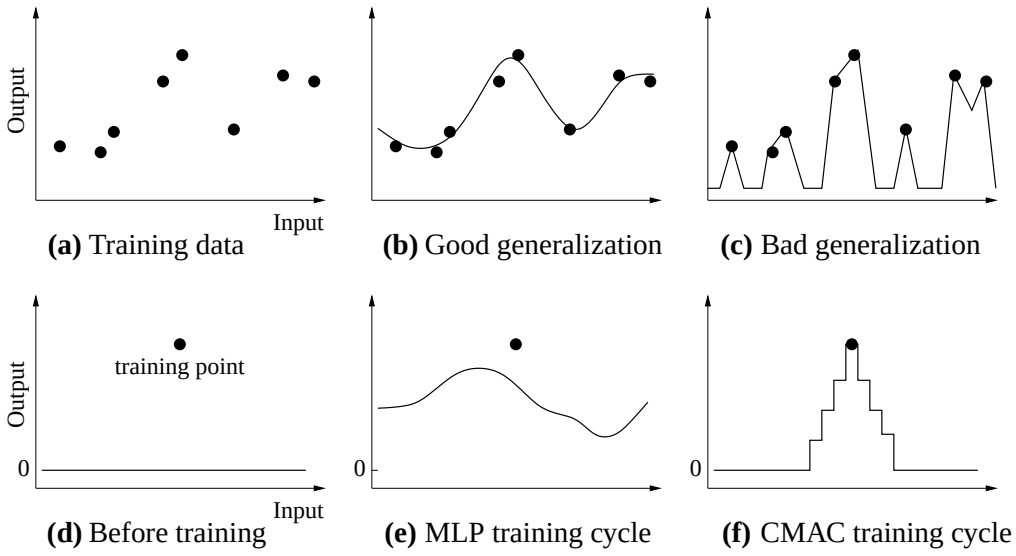
## A.2 Parameterized mappings and generalization

A parameterized mapping is a function of the following form:

$$\mathbf{x} = f(\mathbf{y}, \mathbf{w}) \tag{A.2}$$

where $\mathbf{y}$ is the $n_y \times 1$ input vector, $\mathbf{x}$ is the $n_x \times 1$ output vector, and $\mathbf{w}$ is the $n_w \times 1$ vector of parameters or *weights*. If the function $f$ is a *general approximator* then the weights $\mathbf{w}$ can be selected so that $f$ will approximate any desired $\mathbf{y} \rightarrow \mathbf{x}$ mapping, within some constraints. In other words the mapping is flexible. There are as many different types of general approximator as there are researchers in the field. Many popular approaches are based on neural networks and fuzzy logic, including multi-layer perceptrons, radial basis function networks, and fuzzy neural networks [49].

It is sometimes useful to talk about trajectories in input space and output space. The *input space* is the $n_y$ dimensional space in which a single input $\mathbf{y}$ is represented as a point. An input space trajectory is a continuous one dimensional curve in this space, which represents an input that changes with time. A similar convention applies to the output space.




**(a)** Training data **(b)** Good generalization **(c)** Bad generalization

**(d)** Before training **(e)** MLP training cycle **(f)** CMAC training cycle

**Figure A.1**: *What generalization means.*

## A.3 Some notes on training and generalization

Training is the process of finding a weight vector $\mathbf{w}$ that gives a desired mapping. It is common to train general approximators by presenting *training points* to a training algorithm. Each training point is a pair $(\mathbf{y}_i, \mathbf{x}_i^d)$ where $\mathbf{y}_i$ is an input and $\mathbf{x}_i^d$ is the corresponding desired output ($i$ indexes the training point number). This is called *target training*. A one dimensional example is shown in figure A.1a. An ideal approximator, once trained, will generalize the training points to the surrounding space, producing a mapping which is consistent with the training data and which ignores noise while representing any underlying structural features (figure A.1b). A bad approximator may fit the training data well without correctly interpolating the in-between areas (figure A.1c).

Some approximators such as the multi-layer perceptron (MLP, Appendix C) perform *global generalization*. Every weight has an influence on the mapping over the entire input space. This means that when each data point is presented to the training algorithm, the entire mapping is adjusted. While the mapping at the training point becomes better, the error at other points can be made worse. However the process usually converges to a mapping which accommodates all the data points. Figure A.1d shows one data point and figure A.1e shows a possible result of one MLP training iteration on that point.

Other approximators (such as the CMAC neural network, Chapter 3) perform *local generalization*. Every weight only influences a small region of the entire input space. This means that when each data point is presented to the training algorithm, only a small region of the mapping is adjusted (figure A.1f). Approximators that exhibit local generalization are often quicker to train because the correct value for each weight depends on fewer training points. However, they often give a poorer approximation to the function from which the training data is drawn. One reason for this is that they can easily over-fit the training data. Another is that they may be unable to correctly interpolate the training data in the regions of the input space where there are few data points.

Another common training method is called *error training*. The training data consists of pairs $(\mathbf{y}_i, \Delta\mathbf{x}_i)$ where $\mathbf{y}_i$ is an input and $\Delta\mathbf{x}_i$ is the desired change in output, called the output error. The training algorithm adjusts the mapping so that the output $\mathbf{x}_i$ for the given $\mathbf{y}_i$ becomes $\mathbf{x}_i + \alpha\,\Delta\mathbf{x}_i$, i.e. so it moves in

the *direction* $\Delta\mathbf{x}_i$ ($\alpha$ is an arbitrary number). Note that target training is a special case of error training where $\Delta\mathbf{x}_i = \mathbf{x}_i^d - \mathbf{x}_i$ and $\alpha = 1$.

A common method for target training is gradient descent. A scalar error quantity $E$ is defined like this:

$$E = \sum_i \Big(\mathbf{x}_i^d - \mathbf{x}_i\Big)^2 \tag{A.3}$$

$$= \sum_i \Big(\mathbf{x}_i^d - f(\mathbf{y}_i, \mathbf{w})\Big)^2 \tag{A.4}$$

When all the outputs match the *desired* outputs, the error $E$ is minimized. To minimize $E$ the weight vector $\mathbf{w}$ is moved along the negative gradient $\mathrm{d}E/\mathrm{d}\mathbf{w}$ in small increments. In other words, the following is done at each step:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \left[\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{w}}\right]^T \tag{A.5}$$

$$\leftarrow \mathbf{w} - \alpha \left[\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{x}} \cdot \frac{\mathrm{d}\,\mathbf{x}}{\mathrm{d}\,\mathbf{w}}\right]^T \tag{A.6}$$

$$\leftarrow \mathbf{w} + 2\alpha \left[\frac{\mathrm{d}\,\mathbf{x}}{\mathrm{d}\,\mathbf{w}}\right]^T \cdot \left(\sum_i \mathbf{x}_i^d - \mathbf{x}_i\right) \tag{A.7}$$

The parameter $\alpha$ is the learning rate. Gradient descent can also be applied to *error training* if we define

$$E = \sum_i (\Delta\mathbf{x}_i)^2 \tag{A.8}$$

In which case the training algorithm becomes

$$\mathbf{w} \leftarrow \mathbf{w} + 2\alpha \left[\frac{\mathrm{d}\,\mathbf{x}}{\mathrm{d}\,\mathbf{w}}\right]^T \cdot \left(\sum_i \Delta\mathbf{x}_i\right) \tag{A.9}$$

Often it is computationally useful to perform training using an approximation of the gradient vector $\mathrm{d}E/\mathrm{d}\mathbf{w}$. In such cases it is useful to know under what circumstances the approximation is valid. That obviously depends on the specifics of the approximation, but there are some general observations to be made. Consider figure A.2, which shows how the training algorithm operates in weight-error space. The best error reduction is obviously obtained by traveling along the vector $\mathrm{d}E/\mathrm{d}\mathbf{w}$. However, the error will still be reduced (at least in the short term) along any path within $90°$ of this vector. Such paths make up exactly half of all the possible directions that the weight vector could travel in. So, in an extremely loose sense, it could be said that half of all approximations will work (i.e. will reduce the error). Of course the complicated geometry of the weight-error space means that some directions (i.e. some approximation) will be more useful than others.

One final point: general approximators are often used as components of dynamic systems, that is systems that operate over time. To analyze such systems the concept of *unfolding* is extremely useful (figure A.3). Each time step is represented separately and the system's operation over all time is treated as a whole, where signals propagate from start to finish in a single instant.

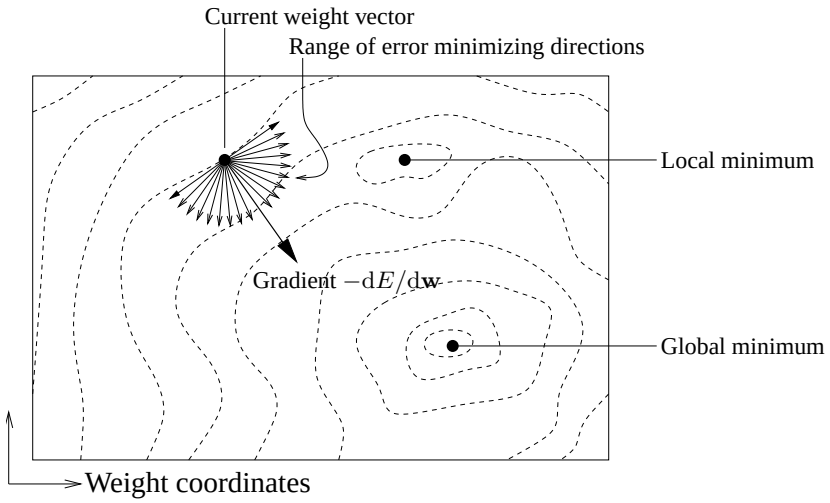



**Figure A.2**: *Gradient descent in weight-error space.*

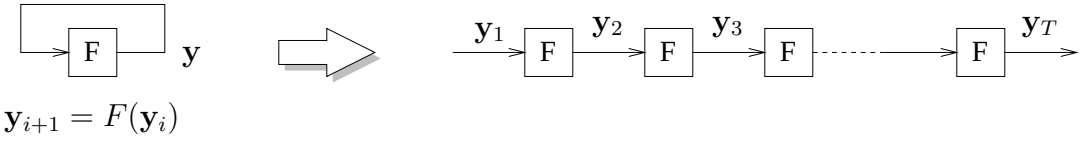



**Figure A.3**: *Unfolding a dynamic system.*

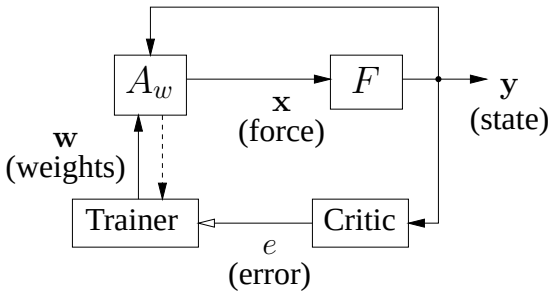



**Figure A.4**: *A basic adaptive control system.*

## A.4 Adaptive control

Figure A.4 shows the form of a simple adaptive control system. $F$ is the process being controlled, which obeys the dynamical equation:

$$\frac{\mathrm{d}\,\mathbf{y}}{\mathrm{d}\,t} = F(\mathbf{y}, \mathbf{x}) \tag{A.10}$$

The vector $\mathbf{y}$ is the system state (for a mechanical system this is typically positions and velocities). It is fed into a stateless controller $A_w$ to produce the vector $\mathbf{x}$, which is the process control force. The function $A_w$ is some kind of "black box" (like a general approximator) that makes decisions about how to control the system. It is parameterized by the weight vector $\mathbf{w}$.

If there is some detailed knowledge of how the controller should work then it may be possible to choose a specific form for the function $A$ and the parameters $w$ that will allow optimal control and make the adaptation problem easier. This can result in efficient control and adaptation when the system $F$ is simple or linear, and its dynamics are well known.

The alternative is to choose $A_w$ to be a general approximator. This means that $A_w$ should be flexible enough that the weights $w$ can be adjusted to achieve *whatever* mapping will eventually be required. In practice it is not possible to specify every possible function with a finite set of parameters, so $A_w$ is chosen such that it can merely *approximate* a wide set of useful functions. This is a useful approach if the required $A_w$ is unknown, for example because the system is nonlinear or has unknown dynamics, or perhaps because the control engineer just can't be bothered to work it out.

The goal of adaptive control is to adjust the parameters $\mathbf{w}$ so that $A_w$ provides the best control (the definition for "best" depends on exactly what you are trying to do). In *optimal* control an error quantity $E$ is defined and the weights $\mathbf{w}$ are adjusted to minimize it. The training process usually involves a *critic* which determines an error signal $e$, and a *trainer* which uses the error information to adjust the weights (using gradient descent for example).

In figure A.4 it is assumed that the entire system state vector can be measured by the controller. In reality only part of the state may be measurable. However, measurability issues will be mostly ignored in this thesis: it will be assumed that enough of the state vector can be determined to provide adequate control.