in parameter space. Given an error function that measures the "mismatch" between the network's actual and desired output, Backpropagation calculates the first derivative of the target function and the parameters of the neural network, and then adapts the parameters in opposite direction of the gradient so as to diminish the mismatch. This raises the question as to what error function to use.

A common approach is to train the learning algorithm so as to maximize the log-likelihood of the training data. More specifically we are given a training set of the form

$$\begin{array}{ccc} \text{input}^{[1]} & \longrightarrow & \text{occ}(\mathbf{m}_i)^{[1]} \\ \text{input}^{[2]} & \longrightarrow & \text{occ}(\mathbf{m}_i)^{[2]} \\ \text{input}^{[3]} & \longrightarrow & \text{occ}(\mathbf{m}_i)^{[3]} \\ \vdots & & \vdots \end{array} \tag{9.13}$$

$occ(\mathbf{m}_i)^{[k]}$ is the k-th sample of the desired conditional probability, and $\text{input}^{[k]}$ is the corresponding input to the learning algorithm. Clearly, the exact form of the input may vary as a result of the encoding known invariances, but the exact nature of this vector will play no role in the form of the error function.

Let us denote the parameters of the learning algorithm by W. Assuming that each individual item in the training data has been generated independently, the likelihood of the training data is now

$$\prod_k p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) \tag{9.14}$$

and its negative logarithm is

$$J(W) \quad = \quad -\sum_k \log p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) \tag{9.15}$$

Here J defines the function we seek to minimize during training.

Let us denote the learning algorithm by $f(\text{input}^{[k]}, W)$. The output of this function is a value in the interval $[0; 1]$. After training, we want the learning algorithm to output the probability of occupancy:

$$p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) \quad = \quad \begin{cases} f(\text{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 1 \\ 1 - f(\text{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 0 \end{cases} \tag{9.16}$$

Thus, we seek an error function that adjusts W so as to minimize the deviation of this predicted probability and the one communicated by the training

example. To find such an error function, we re-write (9.16) as follows:

$$p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) = f(\text{input}^{[k]}, W)^{\mathbf{m}_i^{[k]}} \; (1 - f(\text{input}^{[k]}, W))^{1-\mathbf{m}_i^{[k]}} \tag{9.17}$$

It is easy to see that this product and Expression (9.16) are identical. In the product, one of the terms is always 1, since its exponent is zero. Substituting the product into (9.15) and multiplying the result by minus one gives us the following function:

$$\begin{aligned} J(W) &= -\sum_k \log \left[f(\text{input}^{[k]}, W)^{\mathbf{m}_i^{[k]}} \; (1 - f(\text{input}^{[k]}, W))^{1-\mathbf{m}_i^{[k]}} \right] \\ &= -\sum_k \mathbf{m}_i^{[k]} \log f(\text{input}^{[k]}, W) + (1 - \mathbf{m}_i^{[k]}) \log(1 - f(\text{input}^{[k]}, W)) \end{aligned} \tag{9.18}$$

$J(W)$ is the error function to minimize when training the learning algorithm. It is easily folded into any algorithm that uses gradient descent to adjusts its parameters.

9.3.4 Examples and Further Considerations

Figure 9.9 shows the result of an artificial neural network trained to mimic the inverse sensor model. The robot in this example is equipped with a circular array of sonar range sensors mounted at approximate table height. The input to the network are the relative range and bearing of a target cell, along with the set of five adjacent range measurements. The output is a probability of occupancy: the darker a cell, the more likely it is occupied. As this example illustrates, the approach correctly learns to distinguish freespace from occupied space. The uniformly gray area behind obstacles matches the prior probability of occupancy, which leads to no change when used in the occupancy grid mapping algorithm. Figure 9.9b contains a faulty short reading on the bottom left. Here a single reading seems to be insufficient to predict an obstacle with high probability.

We note that there exists a number of ways to train a function approximator using actual data collected by a robot, instead of the simulated data from the forward model. In general, this is the most accurate data one can use for learning, since the measurement model is necessarily just an approximation. One such way involves a robot operating in a known environment with a known map. With Markov localization, we can localize the robot, and then use its actual recorded measurements and the known map occupancy