7.2.4 Mini-batches

A downside of stochastic gradient descent is that the gradient of the error function computed from a single data point provides a very noisy estimate of the gradient of the error function computed on the full data set. We can consider an intermediate approach in which a small subset of data points, called a *mini-batch*, is used to evaluate the gradient at each iteration. In determining the optimum size for the mini-batch, note that the error in computing the mean from N samples is given by $\sigma/\sqrt{N}$ where σ is the standard deviation of the distribution generating the data. This indicates that there are diminishing returns in estimating the true gradient from increasing the batch size. If we increase the size of the mini-batch by a factor of 100 then the error only reduces by a factor of 10. Another consideration in choosing the mini-batch size is the desire to make efficient use of the hardware architecture on which the code is running. For example, on some hardware platforms, mini-batch sizes that are powers of 2 (for example, 64, 128, 256, ...) work well.

Exercise 7.8

One important consideration when using mini-batches is that the constituent data points should be chosen randomly from the data set, since in raw data sets there may be correlations between successive data points arising from the way the data was collected (for example, if the data points have been ordered alphabetically or by date). This is often handled by randomly shuffling the entire data set and then subsequently drawing mini-batches as successive blocks of data. The data set can also be reshuffled between iterations through the data set, so that each mini-batch is unlikely to have been used before, which can help escape local minima. The variant of stochastic gradient descent with mini-batches is summarized in Algorithm 7.2. Note that the learning algorithm is often still called 'stochastic gradient descent' even when mini-batches are used.

7.2.5 Parameter initialization

Iterative algorithms such as gradient descent require that we choose some initial setting for the parameters being learned. The specific initialization can have a significant effect on how long it takes to reach a solution and on the generalization performance of the resulting trained network. Unfortunately, there is relatively little theory to guide the initialization strategy.

One key consideration, however, is *symmetry breaking*. Consider a set of hidden units or output units that take the same inputs. If the parameters were all initialized with the same value, for example if they were all set to zero, the parameters of these units would all be updated in unison and the units would each compute the same function and hence be redundant. This problem can be addressed by initializing parameters randomly from some distribution to break symmetry. If computational resources permit, the network might be trained multiple times starting from different random initializations and the results compared on held-out data.

The distribution used to initialize the weights is typically either a uniform distribution in the range $[-\epsilon, \epsilon]$ or a zero-mean Gaussian of the form $\mathcal{N}(0, \epsilon^2)$. The choice of the value of ϵ is important, and various heuristics to select it have been proposed. One widely used approach is called *He initialization* (He *et al.*, 2015b). Consider a