Examples of learning rate schedules include linear, power law, and exponential decay:

$$\eta^{(\tau)} = (1 - \tau/K)\,\eta^{(0)} + (\tau/K)\eta^{(K)} \tag{7.36}$$

$$\eta^{(\tau)} = \eta^{(0)}\,(1 + \tau/s)^{c} \tag{7.37}$$

$$\eta^{(\tau)} = \eta^{(0)} c^{\tau/s} \tag{7.38}$$

where in (7.36) the value of η reduces linearly over K steps, after which its value is held constant at $\eta^{(K)}$. Good values for the hyperparameters $\eta^{(0)}$, $\eta^{(K)}$, K, S, and c must be found empirically. It can be very helpful in practice to monitor the *learning curve* showing how the error function evolves during the gradient descent iteration to ensure that it is decreasing at a suitable rate.

7.3.3 RMSProp and Adam

Section 7.3

We saw that the optimal learning rate depends on the local curvature of the error surface, and moreover that this curvature can vary according to the direction in parameter space. This motivates several algorithms that use different learning rates for each parameter in the network. The values of these learning rates are adjusted automatically during training. Here we review some of the most widely used examples. Note, however, that this intuition really applies only if the principal curvature directions are aligned with the axes in weight space, corresponding to a locally diagonal Hessian matrix, which is unlikely to be the case in practice. Nevertheless, these types of algorithms can be effective and are widely used.

The key idea behind *AdaGrad*, short for 'adaptive gradient', is to reduce each learning rate parameter over time by using the accumulated sum of squares of all the derivatives calculated for that parameter (Duchi, Hazan, and Singer, 2011). Thus, parameters associated with high curvature are reduced most rapidly. Specifically,

$$r_i^{(\tau)} = r_i^{(\tau-1)} + \left(\frac{\partial E(\mathbf{w})}{\partial w_i}\right)^2 \tag{7.39}$$

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{\tau}} + \delta}\left(\frac{\partial E(\mathbf{w})}{\partial w_i}\right) \tag{7.40}$$

where η is the learning rate parameter, and δ is a small constant, say 10^{-8}, that ensures numerical stability in the event that r_i is close to zero. The algorithm is initialized with $r_i^{(0)} = 0$. Here $E(\mathbf{w})$ is the error function for a particular mini-batch, and the update (7.40) is standard stochastic gradient descent but with a modified learning rate that is specific to each parameter.

One problem with AdaGrad is that it accumulates the squared gradients from the very start of training, and so the associated weight updates can become very small, which can slow down training too much in the later phases. The idea behind the *RMSProp* algorithm, which is short for 'root mean square propagation', is to replace the sum of squared gradients of AdaGrad with an exponentially weighted average