back-propagation avoids repeating many common subexpressions. This table-filling strategy is sometimes called **dynamic programming**.

6.5.7 Example: Back-Propagation for MLP Training

As an example, we walk through the back-propagation algorithm as it is used to train a multilayer perceptron.

Here we develop a very simple multilayer perception with a single hidden layer. To train this model, we will use minibatch stochastic gradient descent. The back-propagation algorithm is used to compute the gradient of the cost on a single minibatch. Specifically, we use a minibatch of examples from the training set formatted as a design matrix $\boldsymbol{X}$ and a vector of associated class labels $\boldsymbol{y}$. The network computes a layer of hidden features $\boldsymbol{H} = \max\{0, \boldsymbol{X}\boldsymbol{W}^{(1)}\}$. To simplify the presentation we do not use biases in this model. We assume that our graph language includes a `relu` operation that can compute $\max\{0, \boldsymbol{Z}\}$ element-wise. The predictions of the unnormalized log probabilities over classes are then given by $\boldsymbol{H}\boldsymbol{W}^{(2)}$. We assume that our graph language includes a `cross_entropy` operation that computes the cross-entropy between the targets $\boldsymbol{y}$ and the probability distribution defined by these unnormalized log probabilities. The resulting cross-entropy defines the cost J_{MLE}. Minimizing this cross-entropy performs maximum likelihood estimation of the classifier. However, to make this example more realistic, we also include a regularization term. The total cost

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W_{i,j}^{(1)} \right)^2 + \sum_{i,j} \left(W_{i,j}^{(2)} \right)^2 \right) \tag{6.56}$$

consists of the cross-entropy and a weight decay term with coefficient λ. The computational graph is illustrated in figure 6.11.

The computational graph for the gradient of this example is large enough that it would be tedious to draw or to read. This demonstrates one of the benefits of the back-propagation algorithm, which is that it can automatically generate gradients that would be straightforward but tedious for a software engineer to derive manually.

We can roughly trace out the behavior of the back-propagation algorithm by looking at the forward propagation graph in figure 6.11. To train, we wish to compute both $\nabla_{\boldsymbol{W}^{(1)}} J$ and $\nabla_{\boldsymbol{W}^{(2)}} J$. There are two different paths leading backward from J to the weights: one through the cross-entropy cost, and one through the weight decay cost. The weight decay cost is relatively simple; it will always contribute $2\lambda \boldsymbol{W}^{(i)}$ to the gradient on $\boldsymbol{W}^{(i)}$.