the difference between the unit's state and some reference value. Here, we present the dropout algorithm in terms of multiplication by zero for simplicity, but it can be trivially modified to work with other operations that remove a unit from the network.

Recall that to learn with bagging, we define k different models, construct k different datasets by sampling from the training set with replacement, and then train model i on dataset i. Dropout aims to approximate this process, but with an exponentially large number of neural networks. Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent. Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network. The mask for each unit is sampled independently from all of the others. The probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins. It is not a function of the current value of the model parameters or the input example. Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5. We then run forward propagation, back-propagation, and the learning update as usual. Figure 7.7 illustrates how to run forward propagation with dropout.

More formally, suppose that a mask vector $\boldsymbol{\mu}$ specifies which units to include, and $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$. Then dropout training consists in minimizing $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$. The expectation contains exponentially many terms but we can obtain an unbiased estimate of its gradient by sampling values of $\boldsymbol{\mu}$.

Dropout training is not quite the same as bagging training. In the case of bagging, the models are all independent. In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory. In the case of bagging, each model is trained to convergence on its respective training set. In the case of dropout, typically most models are not explicitly trained at all—usually, the model is large enough that it would be infeasible to sample all possible sub-networks within the lifetime of the universe. Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters. These are the only differences. Beyond these, dropout follows the bagging algorithm. For example, the training set encountered by each sub-network is indeed a subset of the original training set sampled with replacement.