3.1.2 使用交叉熵来对 MNIST 数字进行分类

如果程序使用梯度下降算法和反向传播算法进行学习，那么交叉熵作为其中一部分易于实现。稍后将改进前面对 MNIST 手写数字进行分类的程序 network.py。新的程序命名为 network2.py，不仅使用了交叉熵，还有本章介绍的其他技术。下面看看新程序在 MNIST 数字分类问题上的表现。如第 1 章所示，我们会使用一个包含 30 个隐藏神经元的网络，小批量的大小也设置为 10，将学习率设置为 $\eta=0.5$[①]，训练 30 轮。network2.py 的接口和 network.py 的略有区别，但用法还是很好懂的。可以在 Python shell 中使用 `help(network2.Network.SGD)`这样的命令来查看 network2.py 的接口文档。

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.5, evaluation_data=test_data,
... monitor_evaluation_accuracy=True)
```

注意，`net.large_weight_initializer()`命令使用第 1 章介绍的方式来初始化权重和偏置。这里需要执行该命令，因为后面才会改变默认的权重初始化命令。运行上面的代码，神经网络的准确率可以达到 95.49%，这跟第 1 章中使用二次代价函数得到的结果（95.42%）相当接近了。

对于使用 100 个隐藏神经元，而交叉熵及其他参数保持不变的情况，准确率达到了 96.82%。相比第 1 章使用二次代价函数的结果（96.59%）有一定提升。看起来是很小的变化，但考虑到误差率已经从 3.41%下降到 3.18%了，消除了原误差的 1/14，这其实是可观的改进。

跟二次代价相比，交叉熵代价函数能提供类似的甚至更好的结果，然而这些结果不能证明交叉熵是更好的选择，原因是在选择学习率、小批量大小等超参数上花了一些心思。为了让提升更有说服力，需要对超参数进行深度优化。然而，这些结果仍然是令人鼓舞的，它们巩固了先前关于交叉熵优于二次代价的理论推断。

---

① 第 1 章使用了二次代价和 $\eta=3.0$ 的学习率。前面讨论过，当代价函数改变时用“相同”的学习率效果如何难以预料。对于两种代价函数，基于其他超参数的选择，我都做过试验来找出能显著提高性能的学习率。另外，有一个非常粗略的推导，能将交叉熵和二次代价的学习率关联起来。如前所述，二次代价的梯度项中有一个额外的 $\sigma'=\sigma(1-\sigma)$，假设把它按照 σ 的值平均，$\int_0^1\sigma(1-\sigma)\mathrm{d}\sigma=1/6$。可以看到，按照相同的学习率，二次代价会以平均 1/6 的速度进行学习。这提示我们，一个合理的起点是把二次代价的学习率除以 6。当然，这个理由很不严谨，不应拘泥于此，不过有时可以作为有用的起点。