1 数据

1.1 数据来源

MNIST 数据库的手写数字，数字已进行大小标准化，并以固定大小的图像为中心。

1.2 数据描述

有训练集 60000 例，一个测试集 10000 个例子。训练集包含 60000 个示例，并且测试集包含 10000 个示例。

数据集有 4 个文件：

训练图像数据文件：train-images-idx3-ubyte
训练数据的类标文件：train-labels-idx1-ubyte
测试图像数据文件：t10k-images-idx3-ubyte
测试数据的类标文件：t10k-labels-idx1-ubyte

类标文件的数据存储形式是，第 1 个 32 位是一个魔术数字，第 2 个 32 位表示类标个数，随后每 8 位表示一个类标。类标值是 0 到 9。

图像数据文件的数据存储形式是，第 1 个 32 位是一个魔术数字，第 2 个 32 位表示图像的个数，第 3 个 32 位表示图像像素点的行数，第 4 个 32 位表示图像像素点的列数，随后每 8 位表示图像的一个像素（图像以行的方式存储）。像素值是 0 到 255，其中 0 意味着背景（白色），255 意味着前景（黑色）。

数据的详细信息可参考链接：http://yann.lecun.com/exdb/mnist/

2 基础知识

2.1 神经网络

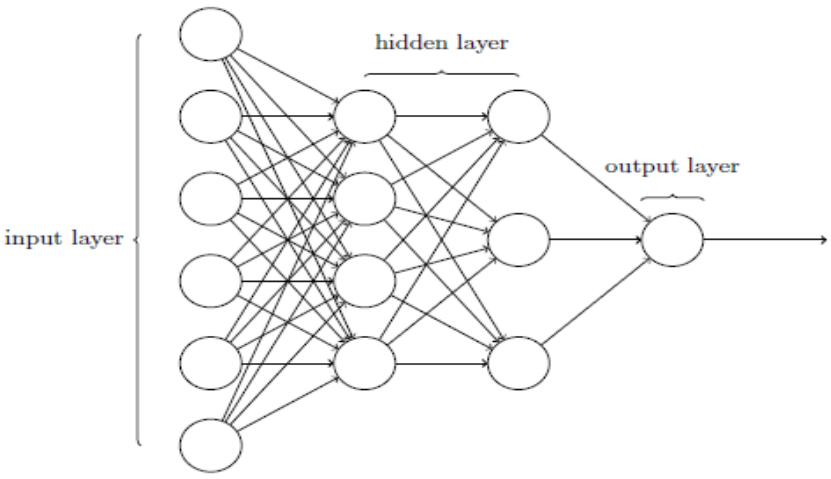

图 2-1　神经网络示例图

神经网络的一般形式如图 2-1 所示，可分为 3 个部分，输入层，隐藏层和输出层，相邻两层之间是全连接的关系,上一层的输出会作为下一层的输入。每一

个隐藏层的作用可以理解为对某一类特征的提取，越靠近输出的层，提取的特征越能代表目标的特征。最后的输出层根据最后一层的特征输出一个类标。神经元之间的连接都有一个权重，第 l-1 层与 l 层神经元的连接可用矩阵W^l表示，其中W^l_{jk}表示第 l-1 层的第 k 个神经元与第 l 层第 k 个神经元的连接。l+1 层神经元从 l 层神经元中得到的输入值的计算如公式（1）所示，其中b表示偏差，σ是 sigmod 函数。

$$a^l = \sigma(W^l a^{l-1} + b^l) \qquad (1)$$

如同其他机器学习算法，神经网络同样有一个目标函数，在书上称为代价函数，下文也这么命名。本文使用的代价函数如公式（2）所示，其中 y 表示实际类标，x 是样本，a 是输出的类标，L 是网络的层数，n 是样本个数。有了这个代价函数，我们便可以通过最小化这个代价函数来达到提升神经网络的分类准确度。

$$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2 \qquad (2)$$

为了最小化这个代价函数，神经网络采用梯度下降法，而梯度的计算方法是反向传播。反向传播的核心是对代价函数 C 关于任何权重 w（或者偏置 b ）的偏导数$\frac{\partial C}{\partial w}$（或$\frac{\partial C}{\partial b}$）的表达式，而$\frac{\partial C}{\partial w}$（或$\frac{\partial C}{\partial b}$）是通过对每个样本求$\frac{\partial C_x}{\partial W_x}$（或$\frac{\partial C_x}{\partial b_x}$）然后求平均得到。这个表达式告诉我们在改变权重和偏置时，代价函数变化的快慢。应用方向传播前，需要做两个假设，首先是代价函数 C 能被改写成一个在每个训练样本 x 上的代价函数$C = \frac{1}{n}C_x$，其中C_x表示每个独立的训练样本其代价是$C_x = \frac{1}{2}||y - a^L||^2$。第二个假设是代价可以写成神经网络输出的函数，代价 C 已经满足了这个假设，因为对于一个样本 x 的输入，C 可以改写成公式（3）。

$$C_x = \frac{1}{2}||y - a^L||^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2 \qquad (3)$$

在表示$\frac{\partial C}{\partial w}$前，先引入一中间变量$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(Z_j^L)$，$\delta_j^l$被称为误差变量，其中$Z_j^L$如公式（4），σ′是 sigmod 函数的导数，$\frac{\partial C}{\partial a} = (a - y)$。公式$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(Z_j^L)$可改写成矩阵的形式如果公式（5）所，其中$\nabla_a C = \frac{\partial C}{\partial a^L}$, ⊙表示矩阵中元素上的乘法。

$$Z_j^l = W_j^l a^{l-1} + b_j^l \qquad (4)$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad (5)$$

使用下一层的δ_j^{l+1}来表示δ_j^l可得到公式（6），这个公式中$\left(W^{l+1}\right)^T$可理解为误差由正向传播改成使误差反向传递的形式，从而求得上一层的误。先用公式（5）求得第一层误差，然后每一层的误差通过公式（6）计算得到。

$$\delta^l = \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l) \qquad (6)$$

代价函数关于网络中任意偏置的改变率$\frac{\partial C}{\partial b_j^l}$的计算如公式（7）所示，改写成矩阵的形式如公式（8）所示。

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad (7)$$

$$\frac{\partial C}{\partial b} = \delta \qquad (8)$$

代价函数关于网络中任意权重的改变率$\frac{\partial C}{\partial w_{jk}^l}$的计算如公式（9）所示，改写成矩阵的形式如公式（10）所示。a_{in}表示神经元输入的上一层输出的激活值，δ_{out}表示输出给下一层的神经元误差。当a_{in}趋于 0 时候，$\frac{\partial C}{\partial w}$趋向很小，就使得 W 的学习变缓慢。

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \qquad (9)$$

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out} \qquad (10)$$

随机梯度下降是加速学习的一种算法，其思想就是通过随机选取小量训练输入样本来计算样本的梯度∇Cx，进而估算实际梯度∇C。估算的公式如公式（11）所示，其中m是一小批样本的数量，n是全部样本的数量。交换公式（11）左右得到公式（12）。

$$\frac{\sum_j^m \nabla C_{x_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C \qquad (11)$$

$$\nabla C \approx \frac{\sum_j^m \nabla C_{x_j}}{m} \qquad (12)$$

假设W_k 和b_l 表示我们神经网络中的权重和偏置。随即梯度下降通过随机地选取并训练输入的小批量数据学习W_k 和b_l的过程如公式（13）和（14）所示。

$$W_k' = W_k - \frac{\eta}{m}\sum_j \frac{\partial C_{x_j}}{\partial C_{W_k}} \qquad (13)$$

$$b_l' = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{x_j}}{\partial C_{b_l}} \qquad (14)$$

神经网络的反向传播过程如表2-1所示。

表2-1　反向传播过程

1. 输入训练样本的集合
2. 设置训练次数epochs，每次训练的样本数mini-batch，学习速率rate
3. 用以0为期望，1为方差的正太分布初始化权值矩阵和偏差矩阵。
4. 设置i=0
5. 当i<epochs执行步骤6到9
6. 将数据随机划分成mini-batch大小的多批样本
7. 对每一小批训练样本，执行步骤8到9：
8.　　对一小批样本x,并执行下面的3个子步骤：

前向传播：对每个l=2,3，...，L计算使用公式（4）计算$z^{x,l}$和使用公式（1）$a^{x,l}$

输出误差：用公式（5）计算输出误差$\delta^{x,L}$

反向传播误差：用公式（6）计算l = L-1,L-2,…,2层的误差$\delta^{x,l}$

9. 根据这一小批样本的误差用公式（13）和（14）更新每一层的W和b

2 SVM 分类

线性 SVM 是对训练数据寻找几何间隔最大的超平面，即以充分大的确信度对训练数据进行分类。如果数据不是线性可分的时候，可以通过非线性变换（核函数技巧）将原空间的数据映射到新空间，使得原空间的超曲面模型对应于新空间超平面模型，然后就可以使用线性 SVM 的方法来求解这个分类问题了。

LibSVM 中可以应用下面 4 种核函数，其中γ（$\gamma > 0$），d 和r是核函数的参数。

●线性核函数$K(x, x^T) = x \cdot x^T$;

●多项式核函数$K(x, x^T) = [\gamma(x \cdot x^T) + r]^d$;

●RBF 核函数$K(x, x^T) = \exp(-\frac{-\gamma|x-x^T|^2}{d^2})$

●sigmod核函数$K(x, x^T) = \tanh(\gamma(x \cdot x^T) + r)$

3 算法实现

3.1 编程环境

系统：Windows10
处理器：赛扬双核 CPU，1.80GHz
GPU：GeForce 610M，显存 1G，显存时钟频率 1.25GHz
内存：4G
编程工具和程序库：pycharm，scikit-learn，tensorflow，cuda

3.2 基于 tensorflow 的神经网络实现

如图 2-1 所示，构建一个神经网络，需要设计神经网络的层数和神经网络每一层的神经元数目，然后设计神经元的连接，以及激活函数。给神经网络设计一个代价函数并通过最小化这个代价函数来优化神经网络的分类效果。

3.2.1 网络的构建

每一层的神经元数目通过 sizes 来设置，然后根据公式（1）来构建网络，激活函数使用 sigmod 函数，边的连接用 W 矩阵表示，传递即是矩阵的乘法。

表 2-2　前递神经网络

```
# sizes[0]输入层神经元个数，sizes[-1]输出层神经元个数，feedforward
def neural_network(data, sizes):
    nextInput = data
    for s1, s2 in zip(sizes[:-1], sizes[1:]):
        layer_w_b = {'w_': tf.Variable(tf.random_normal([s1, s2])),
                     'b_': tf.Variable(tf.random_normal([s2]))}
        # w x+b
        layer = tf.add(tf.matmul(nextInput, layer_w_b['w_']),
layer_w_b['b_'])

        nextInput = tf.nn.sigmoid(layer)    # 激活函数

    return nextInput
```

3.2.2 代价函数和随机梯度下降

使用公式（2）作为代价函数，然后根据公式（13）、（14）实现随机梯度下降法最小化代价函数。

表 2-3　随机梯度下降法最小化代价函数

```
# 使用随机梯度下降法训练神经网络
def train_neural_network(X, Y, sizes, learning_rate, epochs):
    predict = neural_network(X, sizes)
    cost_func = tf.reduce_mean(tf.pow(Y - predict, 2))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=
                    learning_rate).minimize(cost_func)
    epochs = epochs
    with tf.Session() as session:
        session.run(tf.global_variables_initializer())
        epoch_loss = 0
        for epoch in range(epochs):
            for i in range(int(mnist.train.num_examples / batch_size)):
                x, y = mnist.train.next_batch(batch_size)
                _, c = session.run([optimizer, cost_func], feed_dict={X:
x, Y: y})
                epoch_loss += c
            print(epoch, ' : ', epoch_loss)
        correct = tf.equal(tf.argmax(predict, 1), tf.argmax(Y, 1))
        accuracy = tf.reduce_mean(tf.cast(correct, 'float'))
        acc = accuracy.eval({X: mnist.test.images, Y: mnist.test.labels})

        print('准确率: ', acc)

        return acc
```

3.3 基于 scikit-learn 的 svm 实现

Svm使用默认的参数来进行分类，核函数用rbf，d的值设置为3，正则项的值设置为1.0, gamma的值是1/features。该方法是通过使用scikit-learn包的SVC来实现。

表 2-4　基于scikit-learn的svm实现

```
def svm_baseline(C, kernel, degree, gamma):
    training_data, test_data = (mnist.train.images,
np.argmax(mnist.train.labels, axis=1)), (mnist.test.images,
np.argmax(mnist.test.labels, axis=1))
    # train
    clf = svm.SVC(C, kernel, degree, gamma)
    clf.fit(training_data[0], training_data[1])
    # test
    predictions = [int(a) for a in clf.predict(test_data[0])]
    num_correct = sum(int(a == y) for a, y in zip(predictions,
test_data[1]))
    print("Baseline classifier using an SVM.")
    print("Accuracy: {0}".format(num_correct*1.0/len(test_data[1])))
```