实验报告（提纲）

————BP 神经网络

姓名： 何坤宁 学号：16340073 日期：2019.1.12

摘要：

本文构造了一个三层的 BP 神经网络，完成了手写 0-9 数字的识别。利用网上的 MNIST 数据集进行训练、测试，测试正确率在 95%左右。

1．导言

构造一个三层的 BP 神经网络，完成手写 0-9 数字的识别：

1）设计网络的结构，比如层数，每层的神经元数，单个神经元的输入输出函数；

层数为 3，其中输入层包含 784 个神经元，隐藏层包含 30 个神经元，输出层包含 10 个神经元。

输入层的值为用 784 维数组表示的图片。

隐含层和输出层神经元的输入输出函数：

网络的初始化

假设输入层的节点个数为n，隐含层的节点个数为l，输出层的节点个数为m。输入层到隐含层的权重ω_{ij}，隐含层到输出层的权重为ω_{jk}，输入层到隐含层的偏置为a_j，隐含层到输出层的偏置为b_k。学习速率为η，激励函数为$g(x)$。其中激励函数为$g(x)$取 Sigmoid 函数。形式为：

$$g(x) = \frac{1}{1+e^{-x}}$$

隐含层的输出

如上面的三层 BP 网络所示，隐含层的输出H_j为

$$H_j = g\left(\sum_{i=1}^{n} \omega_{ij} x_i + a_j\right)$$

输出层的输出

$$O_k = \sum_{j=1}^{l} H_j \omega_{jk} + b_k$$

2）根据数字识别的任务，设计网络的输入和输出；

网络输入：784 行 1 列的数组表示的图像，表示一张 MNIST 集中 28*28=784 像素的图像

网络输出：十维数组，表示预测的图像的数值

3） 实现 BP 网络的错误反传算法，完成神经网络的训练和测试，最终识别率达到 70%以上

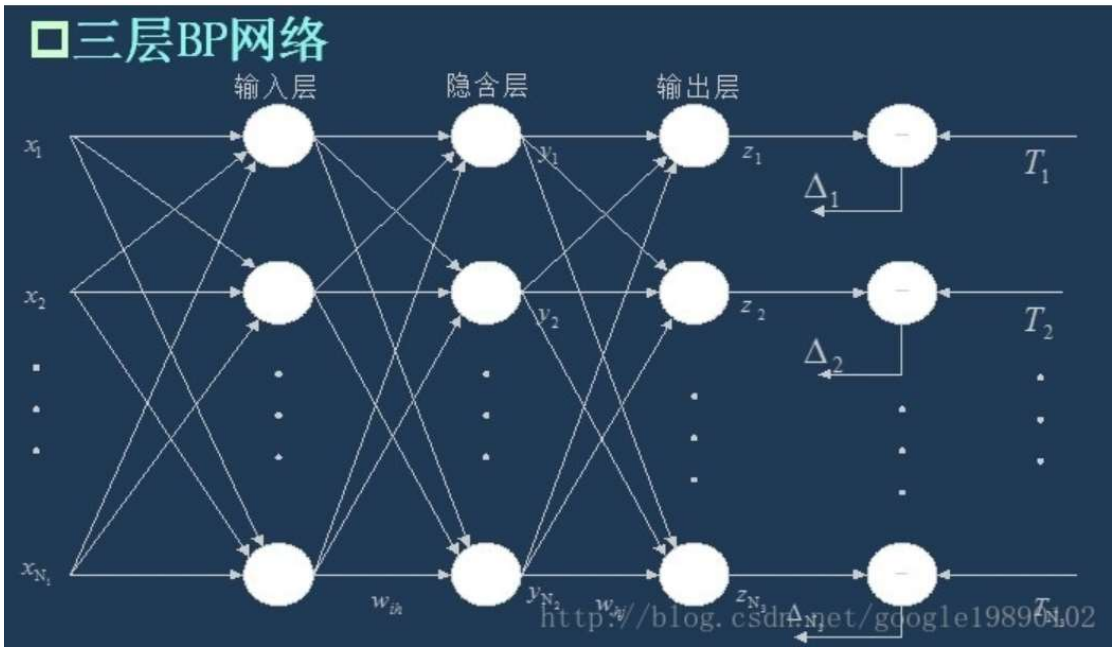
BP 神经网络

BP(Back Propagation)神经网络是一种具有三层或者三层以上的多层神经网络，每一层都由若干个神经元组成，它的左、右各层之间各个神经元实现全连接，即左层的每一个神经元与右层的每个神经元都由连接，而上下各神经元之间无连接。BP 神经网络按有导师学习方式进行训练，当一对学习模式提供给神经网络后，其神经元的激活值将从输入层经各隐含层向输出层传播，在输出层的各神经元输出对应于输入模式的网络响应。然后，按减少希望输出与实际输出误差的原则，从输出层经各隐含层，最后回到输入层（从右到左）逐层修正各

连接权。由于这种修正过程是从输出到输入逐层进行的，所以称它为“误差逆传播算法”。随着这种误差逆传播训练的不断修正，网络对输入模式响应的正确率也将不断提高。

一、BP 神经网络的概念

BP 神经网络是一种多层的前馈神经网络，其主要的特点是：信号是前向传播的，而误差是反向传播的。具体来说，对于如下的只含一个隐层的神经网络模型：



BP 神经网络的过程主要分为两个阶段，第一阶段是信号的前向传播，从输入层经过隐含层，最后到达输出层；第二阶段是误差的反向传播，从输出层到隐含层，最后到输入层，依次调节隐含层到输出层的权重和偏置，输入层到隐含层的权重和偏置。

二、BP 神经网络的流程

在知道了 BP 神经网络的特点后，我们需要依据信号的前向传播和误差的反向传播来构建整个网络。

1、网络的初始化

假设输入层的节点个数为n，隐含层的节点个数为l，输出层的节点个数为m。输入层到隐

含层的权重ω_{ij}，隐含层到输出层的权重为ω_{jk}，输入层到隐含层的偏置为a_j，隐含层到输出层的偏置为b_k。学习速率为η，激励函数为$g(x)$。其中激励函数为$g(x)$取 Sigmoid 函数。形式为：

$$g(x) = \frac{1}{1+e^{-x}}$$

2、隐含层的输出

如上面的三层 BP 网络所示，隐含层的输出H_j为

$$H_j = g\left(\sum_{i=1}^{n} \omega_{ij} x_i + a_j\right)$$

3、输出层的输出

$$O_k = \sum_{j=1}^{l} H_j \omega_{jk} + b_k$$

4、误差的计算

我们取误差公式为：

$$E = \frac{1}{2}\sum_{k=1}^{m} (Y_k - O_k)^2$$

其中Y_k为期望输出。我们记$Y_k - O_k = e_k$，则E可以表示为

$$E = \frac{1}{2}\sum_{k=1}^{m} e_k^2$$

以上公式中，$i = 1 \cdots n$，$j = 1 \cdots l$，$k = 1 \cdots m$。

5、权值的更新

权值的更新公式为：

$$\begin{cases} \omega_{ij} = \omega_{ij} + \eta H_j (1 - H_j) x_i \sum_{k=1}^{m} \omega_{jk} e_k \\ \omega_{jk} = \omega_{jk} + \eta H_j e_k \end{cases}$$

6、偏置的更新

偏置的更新公式为：

$$\begin{cases} a_j = a_j + \eta H_j (1 - H_j) \sum_{k=1}^{m} \omega_{jk} e_k \\ b_k = b_k + \eta e_k \end{cases}$$

隐含层到输出层的偏置更新

$$\frac{\partial E}{\partial b_k} = (Y_k - O_k) \left(-\frac{\partial O_k}{\partial b_k} \right) = -e_k$$

则偏置的更新公式为：

$$b_k = b_k + \eta e_k$$

输入层到隐含层的偏置更新

$$\frac{\partial E}{\partial a_j} = \frac{\partial E}{\partial H_j} \cdot \frac{\partial H_j}{\partial a_j}$$

其中

$$\frac{\partial H_j}{\partial a_j} = \frac{\partial g\left(\sum_{i=1}^{n} \omega_{ij} x_i + a_j\right)}{\partial a_j}$$
$$= g\left(\sum_{i=1}^{n} \omega_{ij} x_i + a_j\right) \cdot \left[1 - g\left(\sum_{i=1}^{n} \omega_{ij} x_i + a_j\right)\right] \cdot \frac{\partial\left(\sum_{i=1}^{n} \omega_{ij} x_i + a_j\right)}{\partial a_j}$$
$$= H_j\left(1 - H_j\right)$$

$$\frac{\partial E}{\partial H_j} = \left(Y_1 - O_1\right)\left(-\frac{\partial O_1}{\partial H_j}\right) + \cdots + \left(Y_m - O_m\right)\left(-\frac{\partial O_m}{\partial H_j}\right)$$
$$= -\left(Y_1 - O_1\right)\omega_{j1} - \cdots - \left(Y_m - O_m\right)\omega_{jm}$$
$$= -\sum_{k=1}^{m}\left(Y_k - O_k\right)\omega_{jk} = -\sum_{k=1}^{m}\omega_{jk} e_k$$

则偏置的更新公式为：

$$a_k = a_k + \eta H_j\left(1 - H_j\right)\sum_{k=1}^{m}\omega_{jk} e_k$$

7、判断算法迭代是否结束

有很多的方法可以判断算法是否已经收敛，常见的有指定迭代的代数，判断相邻的两次误差之间的差别是否小于指定的值等等。

4） 数字识别训练集可以自己手工制作，也可以网上下载。

数字识别集使用 MNIST。

2．实验过程

1.获取训练数据集和测试数据集：

//test.py

```
9    training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
```

training_data:

```
65      training_inputs = [np.reshape(x, (784, 1)) for x in tr_d[0]]
66      training_results = [vectorized_result(y) for y in tr_d[1]]
67      training_data = zip(training_inputs, training_results)
```

(784 像素的图片，图片代表的数值（10 维向量))

validation_data:

```
69      validation_inputs = [np.reshape(x, (784, 1)) for x in va_d[0]]
70      validation_data = zip(validation_inputs, va_d[1])
```

(784 像素的图片，图片数值（单个数字))

test_data:

```
72      test_inputs = [np.reshape(x, (784, 1)) for x in te_d[0]]
73      test_data = zip(test_inputs, te_d[1])
```

(784 像素的图片，图片数值（单个数字))

2.初始化神经网络：

```
11    net = network.Network([784, 30, 10])
```

784 个输入神经元，一层隐藏层，包含 30 个神经元，输出层包含 10 个神经元

3.随机梯度下降，迭代 30 次，小样本数量为 10，学习率为 3.0：

```
12    net.SGD(training_data, 30, 10, 3.0, test_data = test_data)
```

4.每次迭代，首先搅乱数据集，让其排序发生变化：

```
63                random.shuffle(training_data)
```

5. 按照小样本数量划分训练集：

```
65          mini_batches = [
66              training_data[k:k+mini_batch_size]
67              for k in xrange(0, n, mini_batch_size)]
```

共有 5,000 个 mini_batch,每个 mini_batch 有 10 个 training_data

6. 根据每个小样本来更新权重 w 和偏好值 b:

```
68          for mini_batch in mini_batches:
69              # 根据每个小样本来更新 w 和 b，代码在下一段
70              self.update_mini_batch(mini_batch, eta)
```

7. update_mini_batch:根据样本中的每一个输入 x 的其输出 y，计算 w 和 b 的偏导数

```
87      for x, y in mini_batch:
88          # 根据样本中的每一个输入 x 的其输出 y，计算 w 和 b 的偏导数
89          delta_nabla_b, delta_nabla_w = self.backprop(x, y)
```

Backprop 就是 BP 算法，权重和偏差值已经在前面随机设置了，因此我们可以通过前向传播，获得每个节点的值，然后我们需要通过逆向传播来更新权重和偏差值，根据公式：

$$\boldsymbol{W}_L = \boldsymbol{W}_L - \eta \frac{\partial C}{\partial \boldsymbol{W}_L}$$

可知，更新权重，需要知道 C 对 WL 的偏导，而根据数学推导，偏导可以表示成：

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{W}_L} = \boldsymbol{\xi}_L \boldsymbol{o}_{L-1}^T$$

$$\boldsymbol{\xi}_l = (\boldsymbol{W}_{l+1}^T \boldsymbol{\xi}_{l+1}) \circ \boldsymbol{f}'(\boldsymbol{a}_l)$$

因此，只要知道了最后一层的偏差，就可以一步步往前更新权重和偏差。根据定义，假设最

后一层是 L 层，则：

$$\begin{aligned}\boldsymbol{\xi}_L &= \frac{\partial C}{\partial \boldsymbol{a}_L} \\ &= \frac{\partial(\frac{1}{2}||\boldsymbol{o}_L - \boldsymbol{y}||^2)}{\partial \boldsymbol{a}_L} \\ &= (\boldsymbol{o}_L - \boldsymbol{y}) \circ \boldsymbol{f}'(\boldsymbol{a}_L)\end{aligned}$$

现在可以运用公式更新权重和偏差值了。

为了提高效率，我们把训练集随机均匀地分成 N 份。我们就获得了 N 个整体样本的无偏估计子集。在每个子集上进行训练，然后求出平均梯度值，更新一次权重。这样重复直到用完所有的子集。这样的速度就明显会更加快，因为每次只需要用到一小部分来进行更新。

8.backprop:首先进行前向传播，并保存中间变量：

```
114         for b, w in zip(self.biases, self.weights):
115             z = np.dot(w, activation)+b
116             zs.append(z)
117             activation = sigmoid(z)
118             activations.append(activation)
```

9. backprop: 求误差值(最后一层)

```
120         delta = self.cost_derivative(activations[-1], y) * \
121             sigmoid_prime(zs[-1])
122         nabla_b[-1] = delta
```

$$\boldsymbol{\xi}_L = (\boldsymbol{o}_L - \boldsymbol{y}) \circ \boldsymbol{f}'(\boldsymbol{a}_L)$$

10. backprop: 误差值乘于前一层的输出值，获得权重的变化值

```
124         nabla_w[-1] = np.dot(delta, activations[-2].transpose())
```

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{W}_L} = \boldsymbol{\xi}_L \boldsymbol{o}_{L-1}^T$$

11.backprop:根据后一层的误差值更新前一层的误差值及权重变化值，偏差值就设为误差值

```
125         for l in xrange(2, self.num_layers):
126             # 从倒数第 **l** 层开始更新，**-l** 是 python 中特有的语法表示从倒数第 l 层开始计算
127             # 下面这里利用 **l+1** 层的 δ 值来计算 **l** 的 δ 值
128             # wl=wl-n*el*ol-1, el = (Wl+1 ^ T * el+1) ` f'(al)
129             z = zs[-l]
130             sp = sigmoid_prime(z) #f'(al)
131             delta = np.dot(self.weights[-l+1].transpose(), delta) * sp # el = (Wl+1 ^ T * el+1) ` f'(al)
132             nabla_b[-l] = delta
133             nabla_w[-l] = np.dot(delta, activations[-l-1].transpose()) #el*ol-1
134         return (nabla_b, nabla_w)
```

$$\boldsymbol{\xi}_l = (\boldsymbol{W}_{l+1}^T \boldsymbol{\xi}_{l+1}) \circ \boldsymbol{f}'(\boldsymbol{a}_l)$$

12. update_mini_batch: 累加储存偏导值 delta_nabla_b 和 delta_nabla_w

```
91                 nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
92                 nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
```

13. update_mini_batch: 根据累加的偏导值更新 w 和 b，这里因为用了小样本，所以要除于小样本的长度

```
95          self.weights = [w-(eta/len(mini_batch))*nw
96                          for w, nw in zip(self.weights, nabla_w)]
97          self.biases = [b-(eta/len(mini_batch))*nb
98                         for b, nb in zip(self.biases, nabla_b)]
```

$$\boldsymbol{W}_L = \boldsymbol{W}_L - \eta \frac{\partial C}{\partial \boldsymbol{W}_L}$$

14.第一次迭代结束，用测试集测试一下正确率：

```
73                  print "Epoch {0}: {1} / {2}".format(
74                      j, self.evaluate(test_data), n_test)
```

15.evaluate：通过前向传播获得预测结果，然后与正确结果比较

```
138         test_results = [(np.argmax(self.feedforward(x)), y) #argmax得到数值
139                         for (x, y) in test_data]
140         # 返回正确识别的个数
141         return sum(int(x == y) for (x, y) in test_results)
```

16.feedforward: 前向传播，只需要结果，不需要记录中间值

```
44          for b, w in zip(self.biases, self.weights):
45              # 加权求和以及加上 biase
46              a = sigmoid(np.dot(w, a) + b)
47          return a
```

3．结果分析

实验环境：VScode

如何运行：python test.py

结果：

运行 3 次后，测试集正确率最高为 95.30%，最低为 94.81%，平均值为 95.08%，标准差

为 0.20499

1.

```
0 正确率: 9064 / 10000
1 正确率: 9215 / 10000
2 正确率: 9302 / 10000
3 正确率: 9331 / 10000
4 正确率: 9372 / 10000
5 正确率: 9385 / 10000
6 正确率: 9376 / 10000
7 正确率: 9428 / 10000
8 正确率: 9439 / 10000
9 正确率: 9457 / 10000
10 正确率: 9452 / 10000
11 正确率: 9471 / 10000
12 正确率: 9483 / 10000
13 正确率: 9499 / 10000
14 正确率: 9482 / 10000
15 正确率: 9487 / 10000
16 正确率: 9496 / 10000
17 正确率: 9491 / 10000
18 正确率: 9518 / 10000
19 正确率: 9485 / 10000
20 正确率: 9509 / 10000
21 正确率: 9503 / 10000
22 正确率: 9508 / 10000
23 正确率: 9529 / 10000
24 正确率: 9507 / 10000
25 正确率: 9532 / 10000
26 正确率: 9530 / 10000
27 正确率: 9515 / 10000
28 正确率: 9530 / 10000
29 正确率: 9530 / 10000
```

2.

0 正确率: 8178 / 10000
1 正确率: 8329 / 10000
2 正确率: 8444 / 10000
3 正确率: 9363 / 10000
4 正确率: 9394 / 10000
5 正确率: 9399 / 10000
6 正确率: 9409 / 10000
7 正确率: 9372 / 10000
8 正确率: 9466 / 10000
9 正确率: 9418 / 10000
10 正确率: 9454 / 10000
11 正确率: 9479 / 10000
12 正确率: 9493 / 10000
13 正确率: 9467 / 10000
14 正确率: 9492 / 10000
15 正确率: 9486 / 10000
16 正确率: 9502 / 10000
17 正确率: 9533 / 10000
18 正确率: 9531 / 10000
19 正确率: 9506 / 10000
20 正确率: 9532 / 10000
21 正确率: 9501 / 10000
22 正确率: 9498 / 10000
23 正确率: 9497 / 10000
24 正确率: 9543 / 10000
25 正确率: 9533 / 10000
26 正确率: 9518 / 10000
27 正确率: 9532 / 10000
28 正确率: 9529 / 10000
29 正确率: 9515 / 10000

3.

```
0 正确率: 8119 / 10000
1 正确率: 9095 / 10000
2 正确率: 9293 / 10000
3 正确率: 9338 / 10000
4 正确率: 9371 / 10000
5 正确率: 9375 / 10000
6 正确率: 9393 / 10000
7 正确率: 9398 / 10000
8 正确率: 9410 / 10000
9 正确率: 9445 / 10000
10 正确率: 9434 / 10000
11 正确率: 9438 / 10000
12 正确率: 9454 / 10000
13 正确率: 9436 / 10000
14 正确率: 9433 / 10000
15 正确率: 9422 / 10000
16 正确率: 9456 / 10000
17 正确率: 9462 / 10000
18 正确率: 9467 / 10000
19 正确率: 9461 / 10000
20 正确率: 9450 / 10000
21 正确率: 9467 / 10000
22 正确率: 9464 / 10000
23 正确率: 9483 / 10000
24 正确率: 9477 / 10000
25 正确率: 9468 / 10000
26 正确率: 9459 / 10000
27 正确率: 9443 / 10000
28 正确率: 9458 / 10000
29 正确率: 9481 / 10000
```

算法性能：解的精度达到了 70%的要求，结果也比较稳定，算法比较耗时，训练 30 次大概需要 7 分钟。

算法优缺点：

优点：

1.网络实质上实现了一个从输入到输出的映射功能，而数学理论已证明它具有实现任何复杂非线性映射的功能。这使得它特别适合于求解内部机制复杂的问题

2.网络能通过学习带正确答案的实例集自动提取“合理的”求解规则，即具有自学习能力

3.网络具有一定的推广、概括能力

缺点：

1) 对初始权重非常敏感，极易收敛于局部极小

2) BP 神经网络算法的收敛速度慢

3) BP 神经网络结构选择不一：BP 神经网络结构的选择至今尚无一种统一而完整的理论指导，一般只能由经验选定。

改进：

1. 神经网络的层数和每层的神经元数量以及学习率的选择还可以再修改

2. 思考如何避免和修正对初始化权重敏感的问题

3. 解决 Sigmoid 激活函数在深度神经网络中会面临梯度消失的问题

4. 可以在该算法的基础上增加更多可视化的功能，比如手写输入，输入图片判断结果等

4．结论

神经网络是一门重要的机器学习技术。它是目前最为火热的研究方向--深度学习的基础。学习神经网络不仅可以掌握一门强大的机器学习方法，同时也可以更好地帮助理解深度学习技术。

BP 算法，实际上是用来求解多层复合函数的所有变量的偏导数的利器，也就是说，我们可

以通过

$$\boldsymbol{W}_L = \boldsymbol{W}_L - \eta \frac{\partial C}{\partial \boldsymbol{W}_L}$$

这个公式来更新权重和偏好值，但是正向的求偏导方式会造成很多冗余计算，对于权值动则数万的深度模型中的神经网络，这样的冗余所导致的计算量是相当大的。

同样是利用链式法则，BP 算法则机智地避开了这种冗余，它对于每一个路径只访问一次就能求顶点对所有下层节点的偏导值。

虽然对于程序来说，只需要会用推导出来的公式进行计算就好了，但我还是花了一些时间去看推导的过程，也看了很多神经网络相关的资料，对神经网络终于有了更深的了解。如果时间充足，其实还应该增加手写输入的功能。

主要参考文献(三五个即可)

神经网络中 BP 算法的原理与 Python 实现源码解析

https://www.jianshu.com/p/679e390f24bb

BP 神经网络简单流程

https://blog.csdn.net/a493823882/article/details/78683445

BP 神经网络的优缺点介绍

https://blog.csdn.net/chengl920828/article/details/69946881