智能之门

神经网络和深度学习入门

（基于Python的实现）

STEP 5 非线性分类

第 12 章

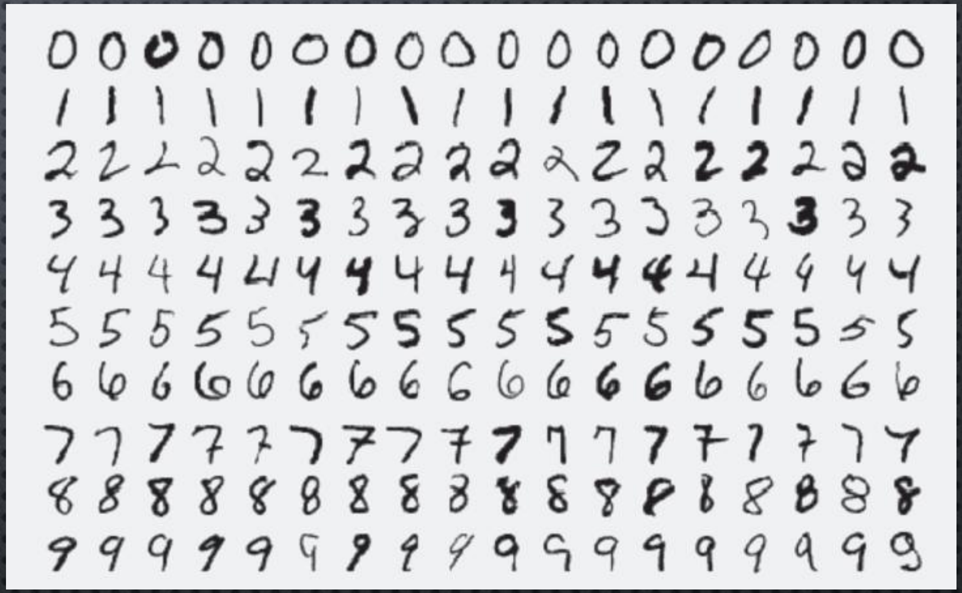
多入多出的三层神经网络
深度非线性多分类

在本部分中，我们将搭建一个三层神经网络，来解决MNIST手写数字识别问题，并学习使用梯度检查来帮助我们测试反向传播代码的正确性。

数据集的使用，是深度学习的一个基本技能，开发集、验证集、测试集，合理地使用才能得到理想的泛化能力强的模型。

12.1 多变量非线性多分类

手写识别是人工智能的重要课题之一。右图是著名的 MNIST 数字手写体识别图片集的部分样例。



由于这是从欧美国家和地区收集的数据，从图中可以看出有几点和中国人的手写习惯不一样：

- ✓ 数字2，下面多一个圈。
- ✓ 数字4，很多横线不出头。
- ✓ 数字6，上面是直的。
- ✓ 数字7，中间有个横杠。

我们可以试试用一个三层的神经网络解决此问题，把每个图片的像素都当作一个向量来看，而不是作为点阵。

本章中，要处理的对象是图片，需要把整张图片看作一个样本，因此先进行数据归一化。

12.2 三层神经网络的实现

➢ 神经网络结构

- 输入层：

$$X \in R^{1\times784}$$

- 隐层1的权重和偏置：

$$W1 \in R^{784\times64}, B1 \in R^{1\times64}$$

- 隐层1：

$$Z1 \in R^{1\times64}, A1 \in R^{1\times64}$$

- 隐层2的权重和偏置：

$$W2 \in R^{64\times16}, B2 \in R^{1\times16}$$

- 隐层2：

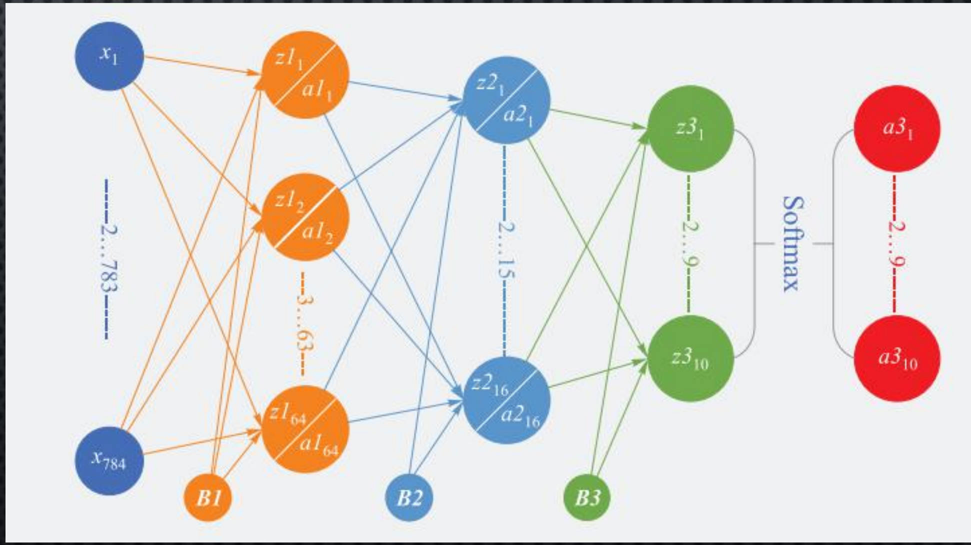
$$Z2 \in R^{1\times16}, A2 \in R^{1\times16}$$

- 输出层的权重和偏置：

$$W3 \in R^{16\times10}, B3 \in R^{1\times10}$$

- 输出层：

$$Z3 \in R^{1\times10}, A3 \in R^{1\times10}$$



12.2 三层神经网络的实现

➢ 前向计算

- 层间计算

$$Z1 = X \cdot W1 + B1, \qquad A1 = Sigmoid(Z1)$$
$$Z2 = A1 \cdot W2 + B2, \qquad A2 = Tanh(Z2)$$
$$Z3 = A2 \cdot W3 + B3, \qquad A3 = Softmax(Z3)$$

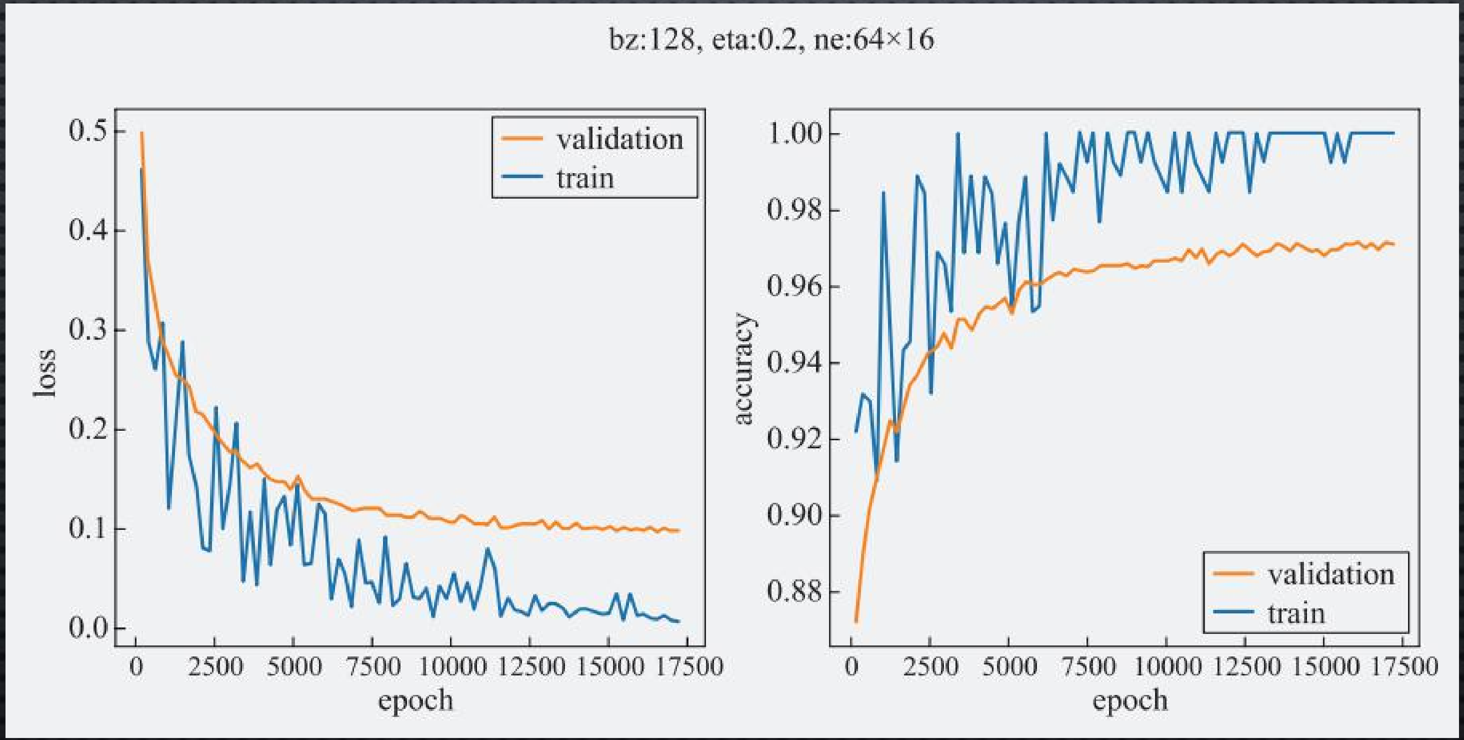
➢ 反向传播

- 链式法则求导，结果类似于此前章节的推导结果：

$$dZ3 = A3 - Y, \qquad dW3 = A2^T \cdot dZ3, \qquad dB3 = dZ3$$
$$dA2 = dZ3 \cdot W3^T, \qquad dZ2 = dA2 \odot (1 - A2 \odot A2), \qquad dW2 = A1^T \cdot dZ2, \qquad dB2 = dZ2$$
$$dA1 = dZ2 \cdot W2^T, \qquad dZ1 = dA1 \odot (1 - A1) \odot A1, \qquad dW1 = X^T \cdot dZ1, \qquad dB1 = dZ1$$

12.2 三层神经网络的实现

➢ 迭代训练结果



12.3 梯度检查

- **梯度检查**
  - 神经网络算法使用反向传播计算目标函数关于每个参数的梯度，可以看做解析梯度。由于计算过程中涉及到的参数很多，用代码实现的反向传播计算的梯度很容易出现误差，导致最后迭代得到效果很差的参数值。
  - 为了确认代码中反向传播计算的梯度是否正确，可以采用梯度检验的方法。通过计算数值梯度，得到梯度的近似值，然后和反向传播得到的梯度进行比较，若两者相差很小的话则证明反向传播的代码是正确无误的。

12.3 梯度检查

➢ 数值微分

- 导数定义式

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- 双边逼近

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

- 泰勒公式

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2!} + \cdots + f^{(n)}(x_0)\frac{(x - x_0)^n}{n!} + R_n$$

- 易知双边逼近的误差量级较单边逼近误差更小。

12.3 梯度检查

➢ 算法内检查

- 初始化神经网络的所有矩阵参数（可以使用随机初始化或其它非0的初始化方法）；
- 把所有层的W,B都转化成向量，按顺序存放在θ中；
- 随机设置X值，最好是归一化之后的值，在[0,1]之间；
- 做一次前向计算，再紧接着做一次反向计算，得到各参数的梯度$d\theta_{real}$；
- 把得到的梯度$d\theta_{real}$变化成向量形式，其尺寸应该和第二步中的θ相同，且一一对应；
- 对第二步中θ向量中的每一个值，做一次双边逼近，得到$d\theta_{approx}$；
- 比较$d\theta_{real}$和$d\theta_{approx}$的值，计算两个向量之间的以下距离：

$$diff = \frac{\left\| d\theta_{real} - d\theta_{approx} \right\|_2}{\left\| d\theta_{real} \right\|_2 + \left\| d\theta_{approx} \right\|_2}$$

12.3 梯度检查

➢ **算法内检查**

- 初始化神经网络的所有矩阵参数（可以使用随机初始化或其它非0的初始化方法）；
- 把所有层的W, B都转化成向量，按顺序存放在θ中；
- 随机设置X值，最好是归一化之后的值，在[0,1]之间；
- 做一次前向计算，再紧接着做一次反向计算，得到各参数的梯度$d\theta_{real}$；
- 把得到的梯度$d\theta_{real}$变化成向量形式，其尺寸应该和第二步中的θ相同，且一一对应；
- 对第二步中θ向量中的每一个值，做一次双边逼近，得到$d\theta_{approx}$；
- 比较$d\theta_{real}$和$d\theta_{approx}$的值，计算两个向量之间的以下距离：

$$diff = \frac{\left\| d\theta_{real} - d\theta_{approx} \right\|_2}{\left\| d\theta_{real} \right\|_2 + \left\| d\theta_{approx} \right\|_2}$$

- 一般而言，当$diff < 1e^{-7}$时，结果非常令人满意；当当$diff > 1e^{-2}$时，需要重新检查过程。此外，网络深度的增加会使得误差积累，因此应视具体情况分析。

12.3 梯度检查

➢ 注意事项

- 不要使用梯度检验去训练，即不要使用梯度检验方法去计算梯度，因为这样做太慢了。
- 其次，如果我们在使用梯度检验过程中发现backprop过程出现了问题，就需要对所有的参数进行计算，以判断造成计算偏差的来源在哪里。
- 别忘了正则化。
- 注意，如果我们使用了drop-out正则化，梯度检验就不可用了。因为我们知道drop-out是按照一定的保留概率随机保留一些节点，因为它的随机性，目标函数J的形式变得非常不明确，这时我们便无法再用梯度检验去检验backprop。
- 最后，介绍一种特别少见的情况。在刚开始初始化W和b时，W和b的值都还很小，这时backprop过程没有问题，但随着迭代过程的进行，W和b的值变得越来越大时，backprop过程可能会出现问题，且可能梯度差距越来越大。要避免这种情况，我们需要多进行几次梯度检验。

12.4 学习率与批大小

- **梯度下降公式**

$$w_{t+1} = w_t - \frac{\eta}{m}\sum_{i=1}^{m}\nabla loss_i(w,b)$$

- 使用梯度下降的各种形式时，我们通常面临以下挑战：
  - ✓ **很难选择出合适的学习率**：太小的学习率会导致网络收敛过于缓慢，而学习率太大可能会影响收敛，并导致损失函数在最小值上波动，甚至出现梯度发散。
  - ✓ **相同的学习率并不适用于所有的参数更新**：如果训练集数据很稀疏，且特征频率非常不同，则不应该将其全部更新到相同的程度，但是对于很少出现的特征，应使用更大的更新率。
  - ✓ **应避免陷于多个局部最小值中**：实际上，问题并非源于局部最小值，而是来自鞍点，即一个维度向上倾斜且另一维度向下倾斜的点。这些鞍点通常被相同误差值的平面所包围，这使得SGD算法很难脱离出来，因为梯度在所有维度上接近于零。

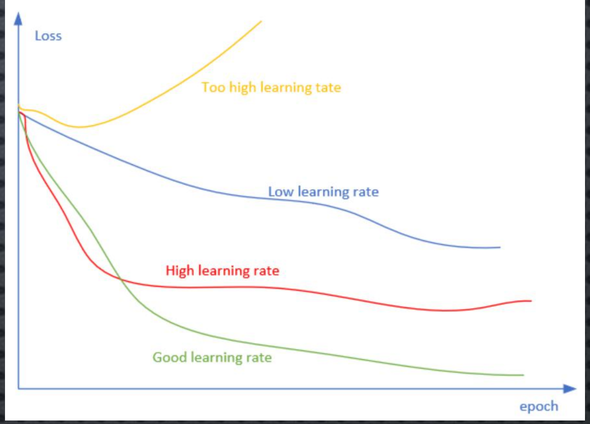
12.4 学习率与批大小

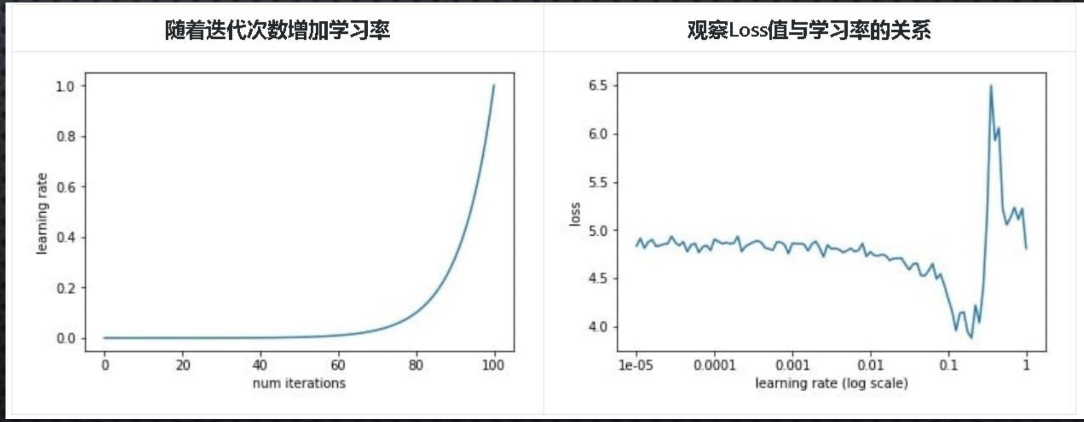
➢ 初始学习率的选择

- 保证SGD收敛的充分条件是：

$$\sum_{k=1}^{\infty} \eta_k = \infty, \qquad \sum_{k=1}^{\infty} \eta_k^2 < \infty$$

- 有一种方式可以帮助我们快速找到合适的初始学习率，方法非常简单：
  - ✓ 首先设置一个非常小的初始学习率；
  - ✓ 然后在每个batch之后都更新网络，计算损失函数值，同时增加学习率；
  - ✓ 最后描绘出学习率的变化曲线和loss的变化曲线，能够发现最好的学习率。





12.4 学习率与批大小

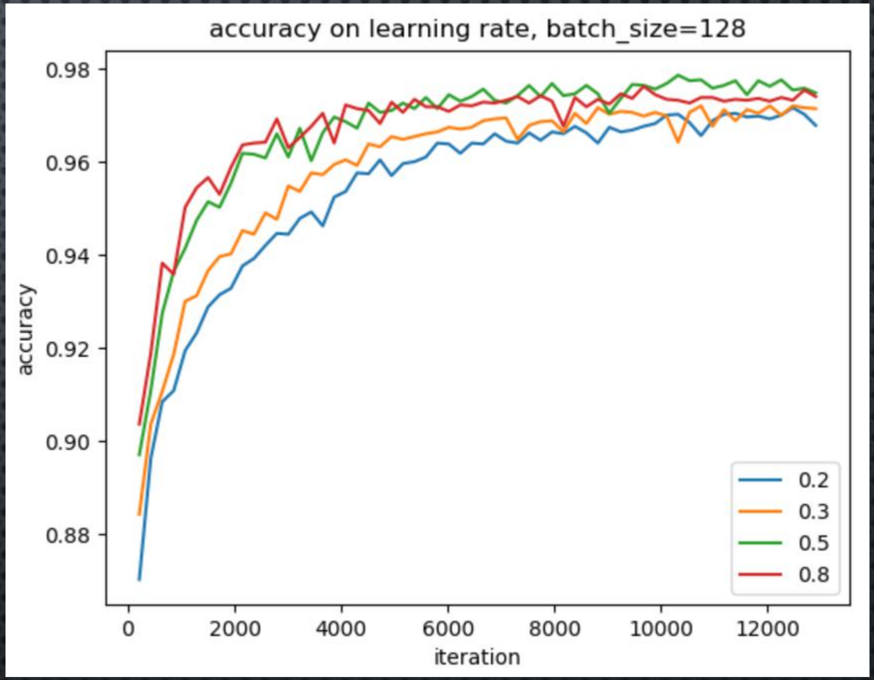
➢ 学习率的后期修正

用MNIST的例子，固定批大小为128时，分别使用学习率为0.2，0.3，0.5，0.8来比较学习曲线。

学习率为0.5时效果最好，虽然0.8的学习率开始时上升得很快，但是到了10个epoch时，0.5的曲线就超上来了，最后稳定在0.8的曲线之上。

这就给了我们一个启示：初期可以把学习率设置大一些，让准确率快速上升，损失值快速下降；到一定阶段后，换用小一些的学习率继续训练。

$$LR_{new} = LR_{current} \times DecayRate^{\frac{GlobalStep}{DecayStep}}$$



12.4 学习率与批大小

➢ 常用学习率下降算法

- **Fixed**：使用固定学习率，比如全程都用0.1。注意这个值不能大，否则在后期接近极值点时不易收敛。
- **Step**：每迭代一个预订的次数后（比如500步），就调低一次学习率。离散型，简单实用。
- **Multistep**：预设几个迭代次数，到达后调低学习率。与Step不同的是，这里的次数可以是不均匀的。
- **Exp**：连续的指数变化的学习率，公式为：

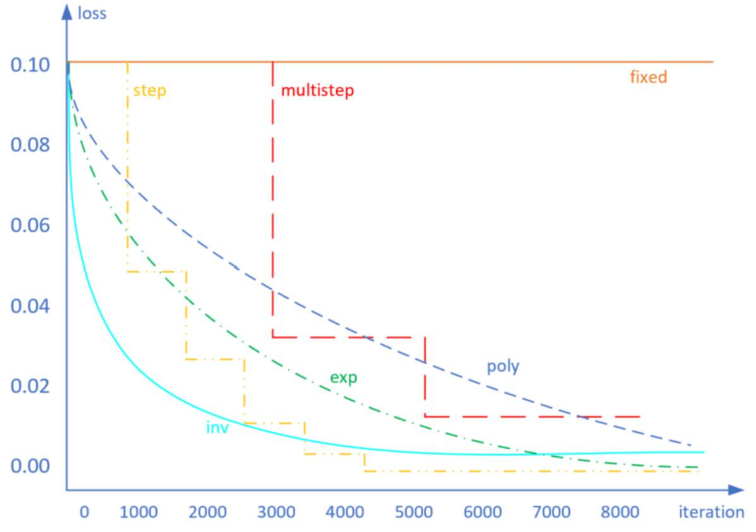
$$lr_{new} = lr_{base} \times \gamma^{iteration}$$

- **Inv**：倒数型变化，公式为：

$$lr_{new} = lr_{base} \times \frac{1}{(1 + \gamma \times iteration)^p}$$

- **Poly**：多项式衰减，公式为：

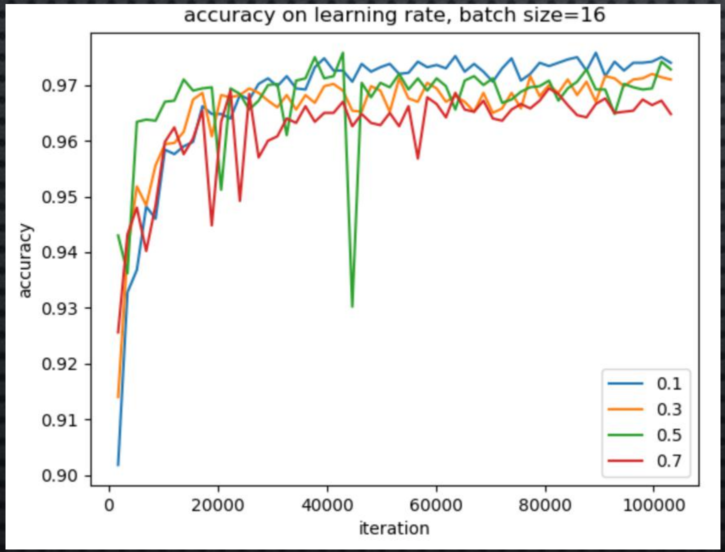
$$lr_{new} = lr_{base} \times \left(1 - \frac{iteration}{iteration_{max}}\right)^p$$



12.4 学习率与批大小

➢ 学习率与批大小的关系

下图是当批大小为16和32时，不同学习率的比较情况。这说明当批大小小到一定数量级后，学习率要和批大小匹配，较大的学习率配和较大的批量，反之亦然。



12.4 学习率与批大小

使用Mini-batch的好处是可以克服单样本的噪音，此时就可以使用稍微大一些的学习率，让收敛速度变快，而不会由于样本噪音问题而偏离方向。从偏差方差的角度理解，单样本的偏差概率较大，多样本的偏差概率较小，而由于独立同分布的假设存在，多样本的方差是不会有太大变化的，即16个样本的方差和32个样本的方差应该差不多，那它们产生的梯度的方差也应该相似。

研究表明，衰减学习率可以通过增加batch size来实现类似的效果，这实际上从SGD的权重更新式子就可以看出来两者确实是等价的。对于一个固定的学习率，存在一个最优的batch size能够最大化测试精度，这个batch size和学习率以及训练集的大小正相关。对此实际上是有两个建议：

- 如果增加了学习率，那么batch size最好也跟着增加，这样收敛更稳定。
- 尽量使用大的学习率，因为很多研究都表明更大的学习率有利于提高泛化能力。如果真的要衰减，可以尝试其他办法，比如增加batch size，学习率对模型的收敛影响真的很大，慎重调整。

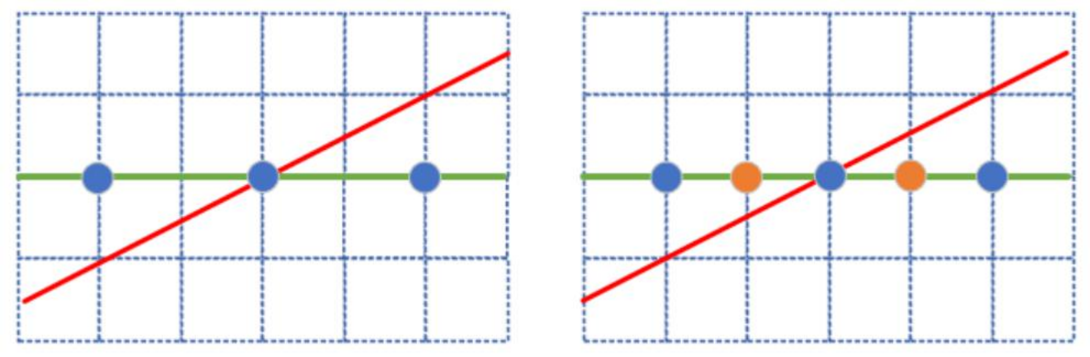
12.4 学习率与批大小

➢ 数值理解

- 左图中有三个蓝色样本点，目前拟合结果是红色直线，斜率为0.5。其损失函数值和目标学习率计算如下：

$$J=\frac{1}{2\times 3}(1^2+0^2+1^2)=0.333$$

$$w-\eta_1\times 0.333=0,\qquad \eta_1=1.5$$



- 右图中多了两个橙色的样本点，相当于批大小从3变为了5。类似地，其损失函数值和目标学习率计算如下：

$$J=\frac{1}{2\times 5}(1^2+0.25^2+0^2+0.25^2+1^2)=0.25$$

$$w-\eta_2\times 0.25=0,\qquad \eta_2=2$$

12.4 学习率与批大小

- 较大的batch size可以减少迭代次数，从而减少训练时间；另一方面，较大batch size的梯度计算更稳定，曲线平滑。在一定范围内，增加batch size有助于收敛的稳定性，但是过大的batch size会使得模型的泛化能力下降，验证或测试的误差增加。
- 批大小的增加可以比较随意，比如从16到32、64、128等等，而学习率是有上限的，不能大于1.0，这一点就如同Sigmoid函数一样，输入值可以变化很大，但很大的输入值会得到接近于1的输出值。因此batch size和学习率的关系可以大致总结如下：
  - ✓ 增加batch size，需要增加学习率来适应，可以用线性缩放的规则，成比例放大
  - ✓ 到一定程度，学习率的增加会缩小，变成batch size的$\sqrt{m}$倍
  - ✓ 到了比较极端的程度，无论batch size再怎么增加，也不能增加学习率了

THE END

谢谢！