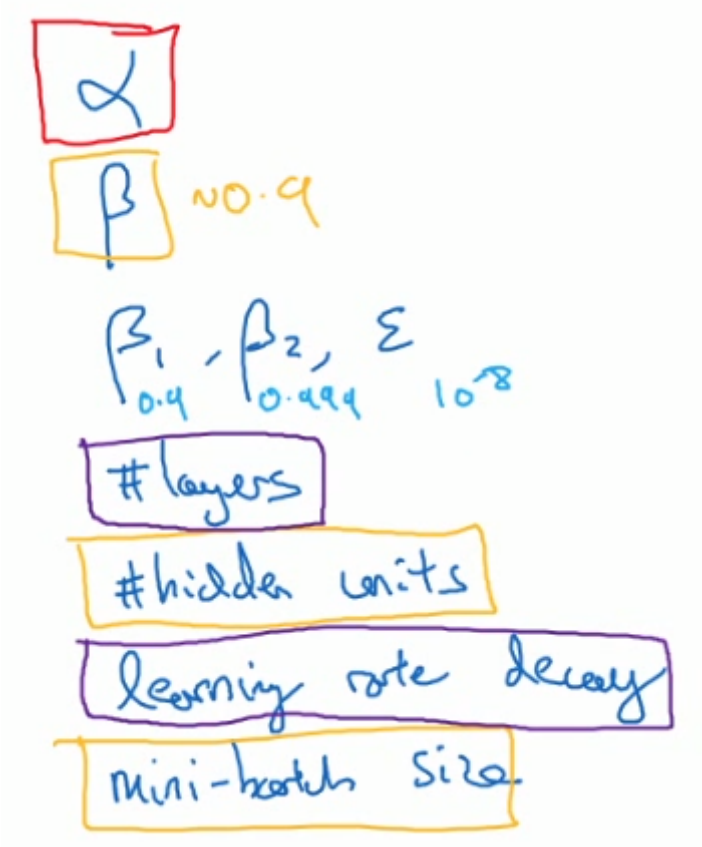
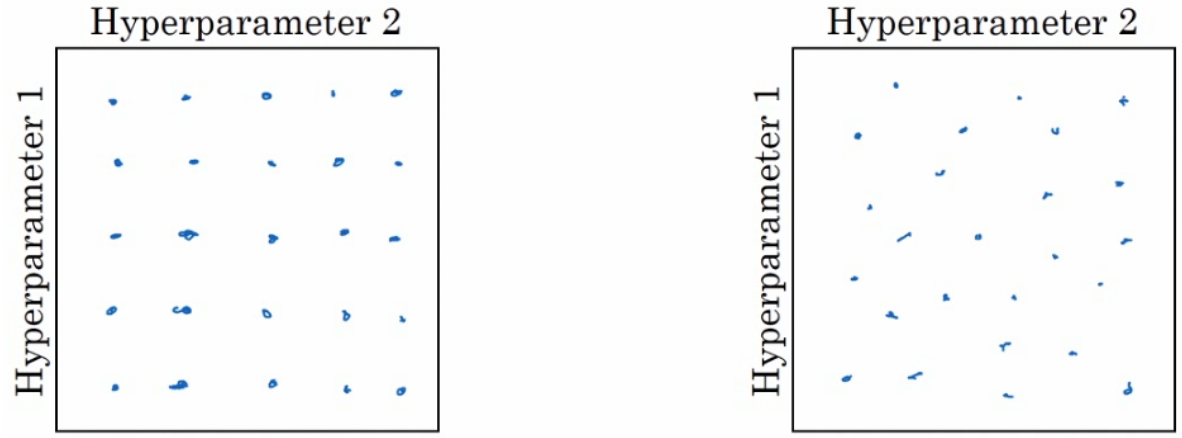
（一）调试处理

超参数优先级



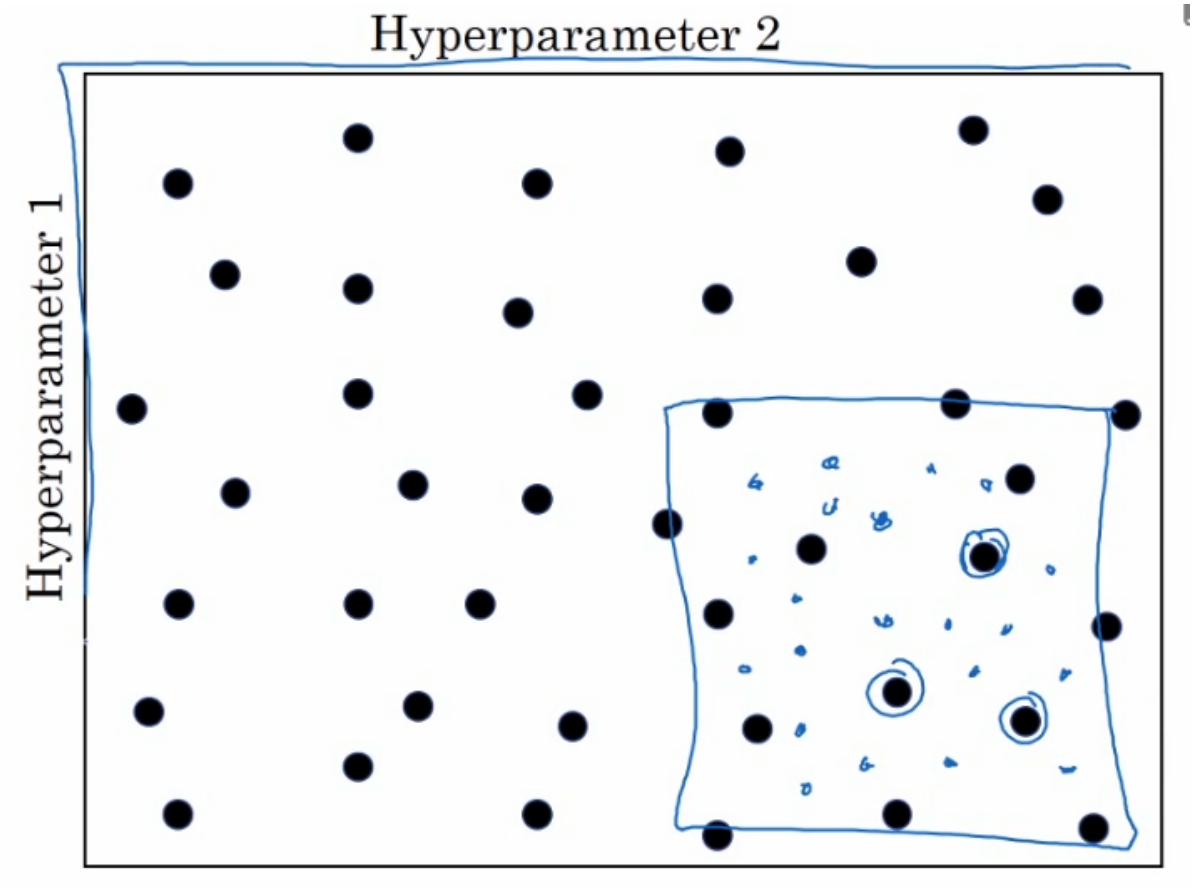
红色第一，黄色第二，紫色第三，没框的基本不调

超参数数值搭配选择方法



超参数少的时候可以像左面一样均匀取点研究效果。

超参数多的时候可以选择随机取点研究效果。如果此时发现某点及其附近的效果比其它部分好，那么就在这附近的区域较为密集地再多取一些点。如此研究直到得出足够满意的超参数搭配。如下图：



（二）为超参数选择合适的范围（标尺）

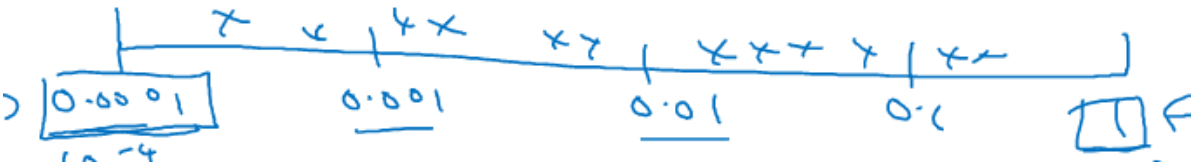
举例说明：

神经网络某层的神经元个数：可以取50-100间均匀随机值

神经网络层数：可以取2-4间均匀随机值

学习率：可以取0.0001-1间不均匀随机值

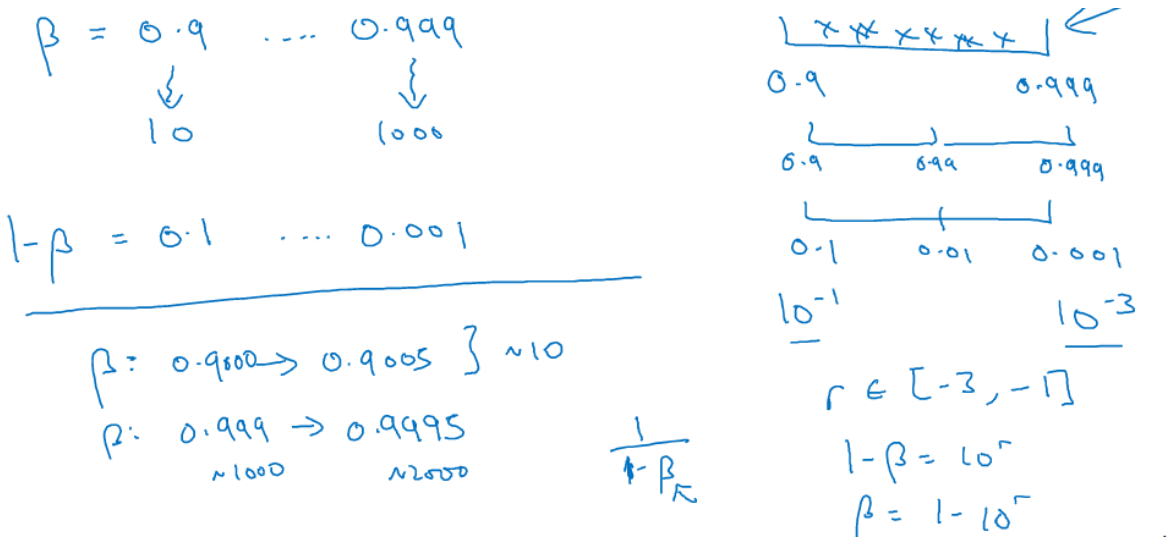
$$\alpha = 10^{-4*np.randon.rand()}$$



标尺长上面这样。这样可以给不同的数量级分配相同的搜索资源。

指数平均的beta：可以取0.9-0.999间不均匀随机值。

因为指数平均计算的是$\frac{1}{1-\beta}$个数的平均值，这个式子在beta接近1的时候对beta更加敏感，因此应该令beta越接近1时，给相应的beta范围分配更多的搜索权重。



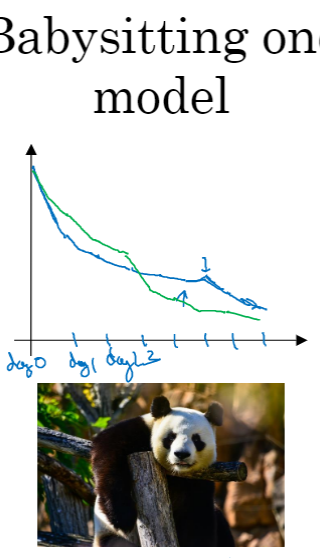
如图所示，对1-beta施以类似上面的学习率的计算，即可达到效果。

实际上，即使不使用这类换标尺的方法，只要有足够数据，或者能恰当的使用（一）中逐渐缩小超参数组合范围的方法，也可以较快的算出超参数的恰当值。

（三）超参数训练方法实践

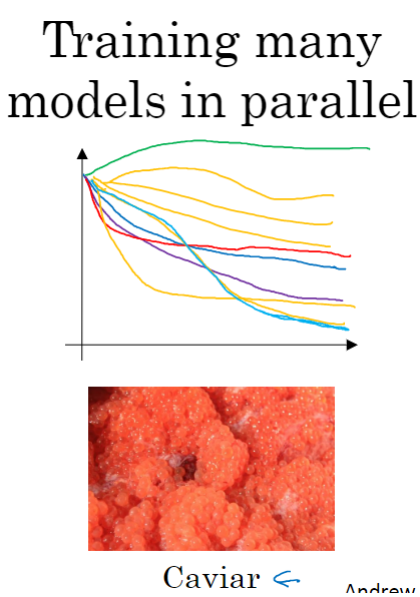
熊猫法

同时运行一个模型，观察其性能随时间变化，手动调整超参数



鱼子酱法

同时运行多个模型，不进行人工干预。全部训练完毕后选出训练结果较好的模型。



（四）对网络的激活函数进行正则化

含义：将z进行正则化。即将z的分布调整到平均值为0，值分布调整到0-1之间（方差为1）。

目的：使得神经网络的参数计算更有效率

注意：在训练隐藏层的时候，有时候为了发挥sigmoid、tanh等的效果，你不希望数据的方差变为1，那么你就没必要对z正则化了

结果：此时公式为

$$z = \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

其中mu是z的平均数，sigma是z的方差，epsilon是防止除零的工具数。

但是有时候我们不希望z分布在0-1、平均值为0，也许分布在别的地方会更有意义。

因此我们有了新公式：

$$z = \gamma z + \beta$$

gamma和beta是两个参数

当

$$\gamma = \frac{1}{\sqrt{\sigma^2 + \epsilon}}$$
$$\beta = \frac{-\mu}{\sqrt{\sigma^2 + \epsilon}}$$

即为平均值为0方差为1的情况

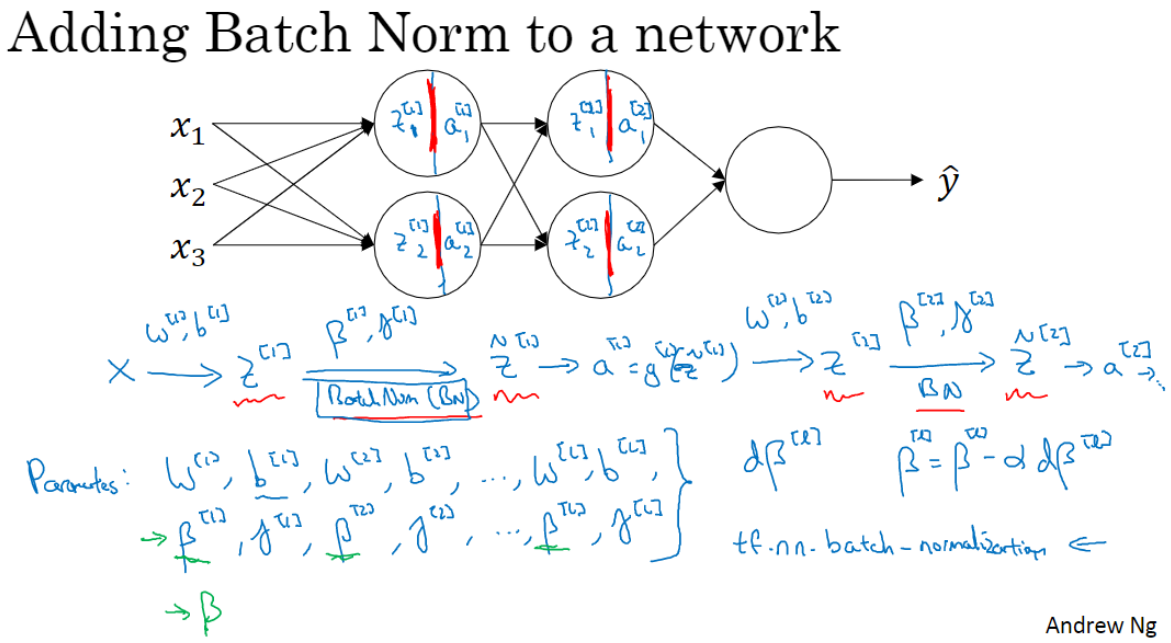
在神经网络中，每一层都有自己的gamma和beta。他们的值的设定视情况而定

图像请看第一周的**正则化输入**一节

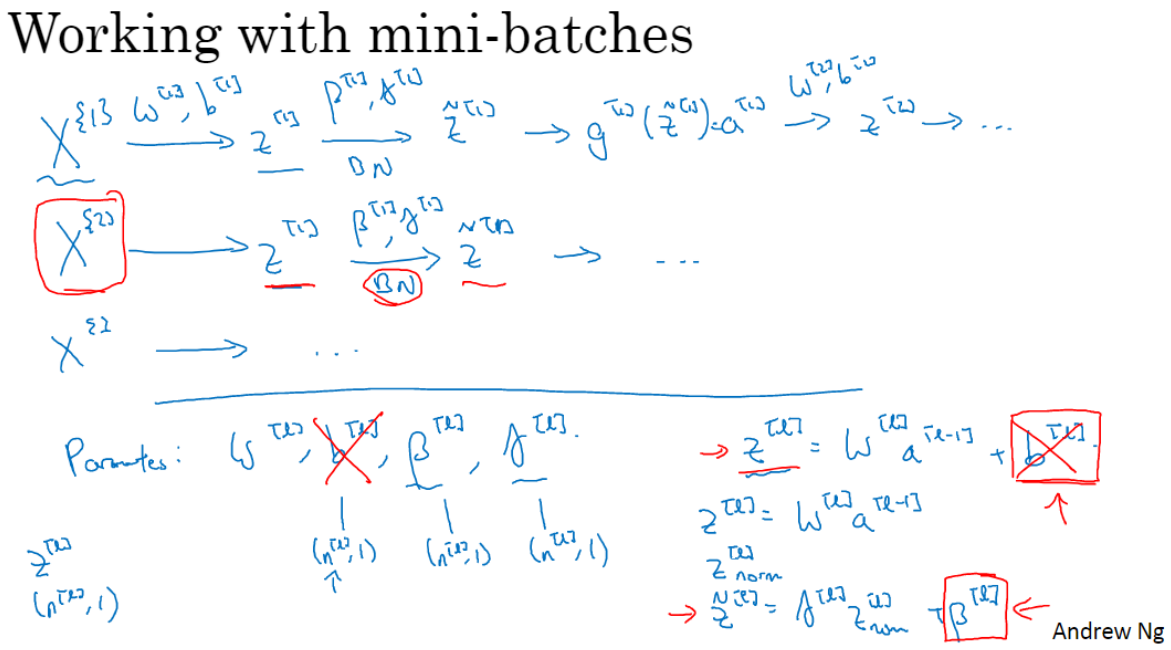
（五）把batch norm拟合进神经网络

使用的位置

前向传播中：在计算出z后，使用激活函数前



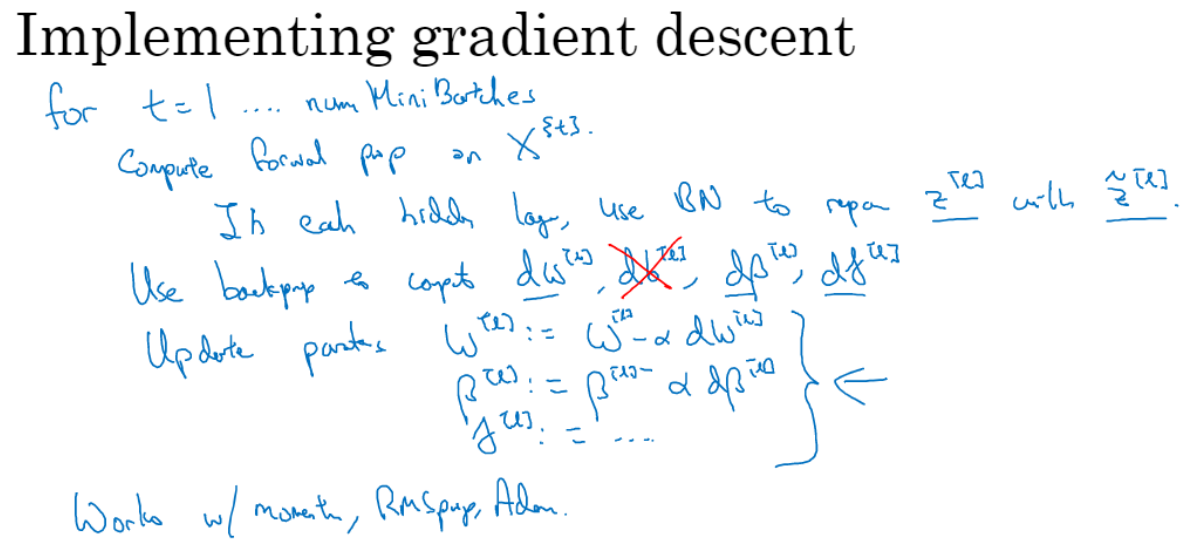
与minibatch一起使用



注意，此时参数b失去了意义，因为b原来的意义是调整z的偏移，现在z所有的偏移最终都由归一化确定了，所以b没用了，可以不算

> 注意到实际计算中b的维数和gamma和beta的维数相同

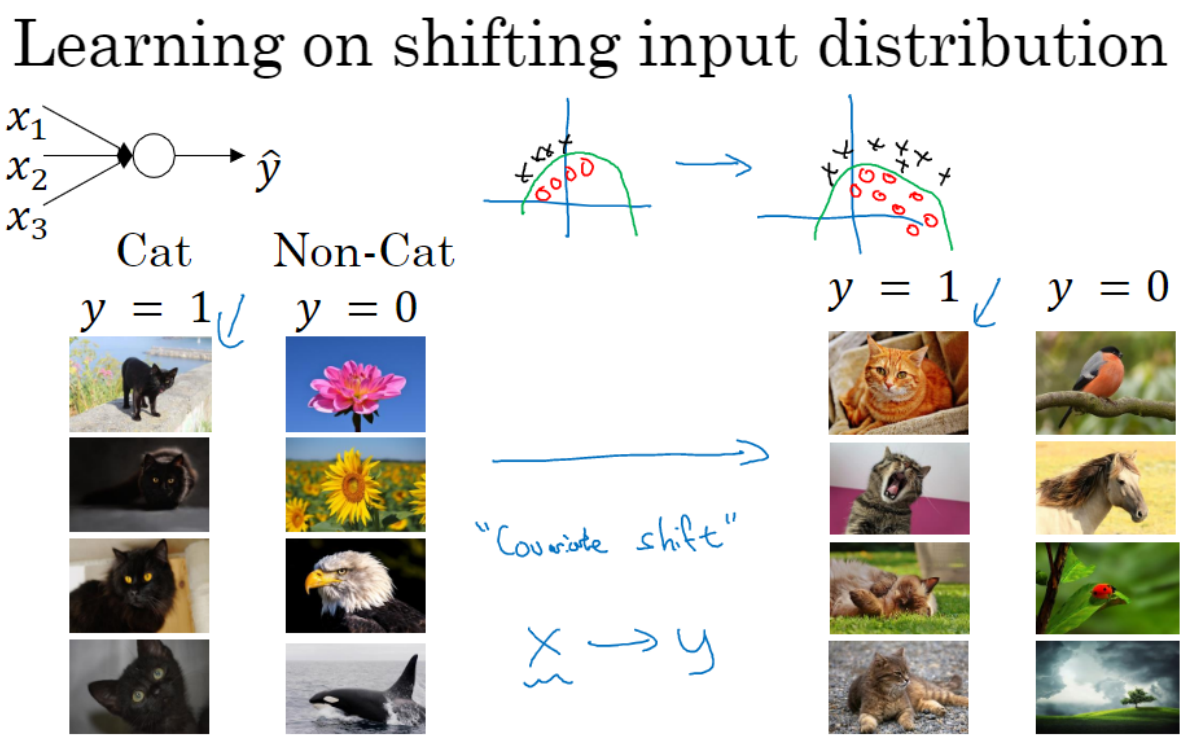
在梯度下降中使用



注意，beta和gamma也是和W、b同等级的参数，反向传播也需要他们。

（六）batch norm的优秀之处

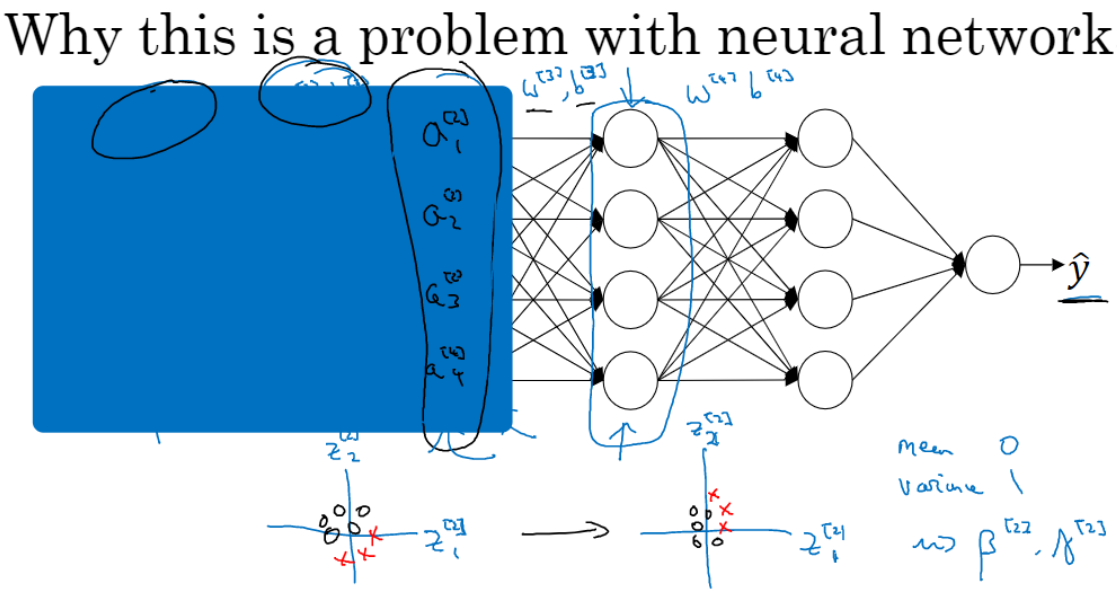
问题的提出



如上图，如果你的训练集是左边的那些黑猫，那么你的训练样本分布可以由左侧坐标系图代替。如果你的训练集是右边那些黑猫，那么你的样本分布可由右侧坐标图代替。

假设你分别用两组训练集训练两个模型，那么由于样本分布不同，最后得到的“找猫函数”也会不同。

现在你希望有一个模型，能同时识别黑猫和不同颜色的猫，但是由上可知，如果你同时用黑猫和多种颜色猫的训练集进行训练，就容易让神经元感到“迷惑”



又如上图所示，第三层神经元希望把上一层的输入正确的映射到y上，但是如果不用batch norm，第三层神经元获得的输入，就如同黑猫和多色猫一样，样本分布很乱，训练效果就会下降。

batch norm的作用

减少了某层神经元接收到的样本的分布的变化范围，使得该层神经元学习效果更好

batch归一化减少了输入值改变的问题，它使得学习效果变得更稳定、神经网络的之后层具有更坚实的基础，它减少了每层数据与之后层的联系，使得每层网络都可以自己学习，稍稍独立于其他层。这有助于加速神经网络的学习。

当batchnorm运行于mini-batch上的时候，它具有轻微的正则化作用。理由：它在处理均值和方差的时候产生了噪音。因为产生了噪音，就是的后续神经元不能过于依赖前面的神经元，起到了正则化（减轻过拟合）的作用。这个噪音很轻微，因此batch-norm可以和其他正经的正则化方法一起用。

使用较大的mini-batch，会减弱dropout的效果

（七）测试时的batch norm

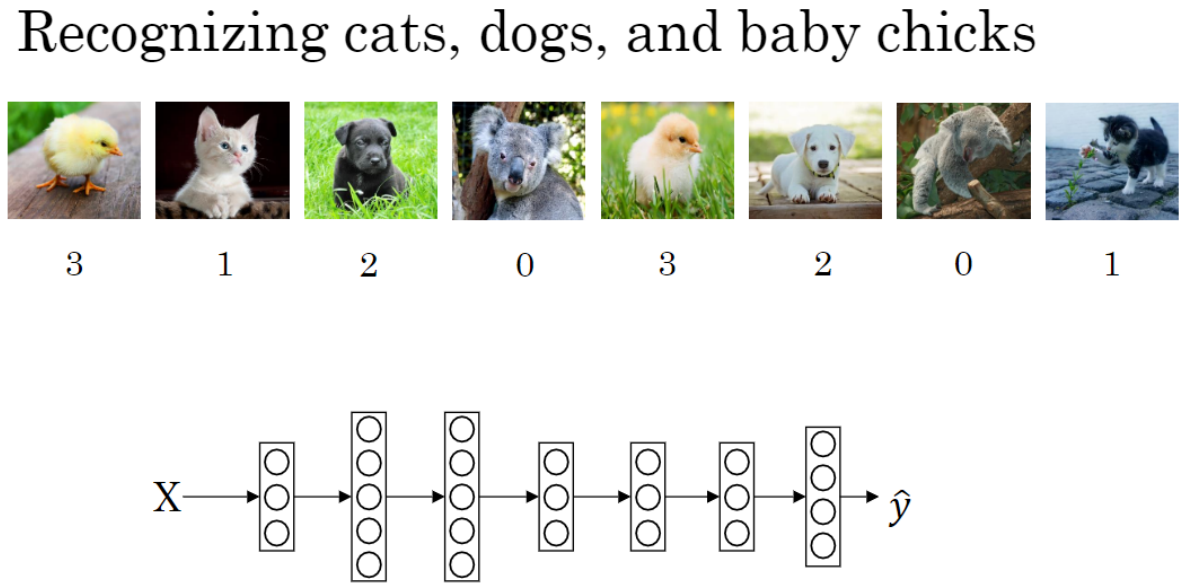
测试时，我们可能不使用minibatch，而是一个一个的过训练样本。这个时候训练集的平均数和方差怎么获得呢?

1. 估计一个数
2. 使用指数加权平均计算
3. 使用深度学习框架自带的机能来计算

（八）softmax回归

简介

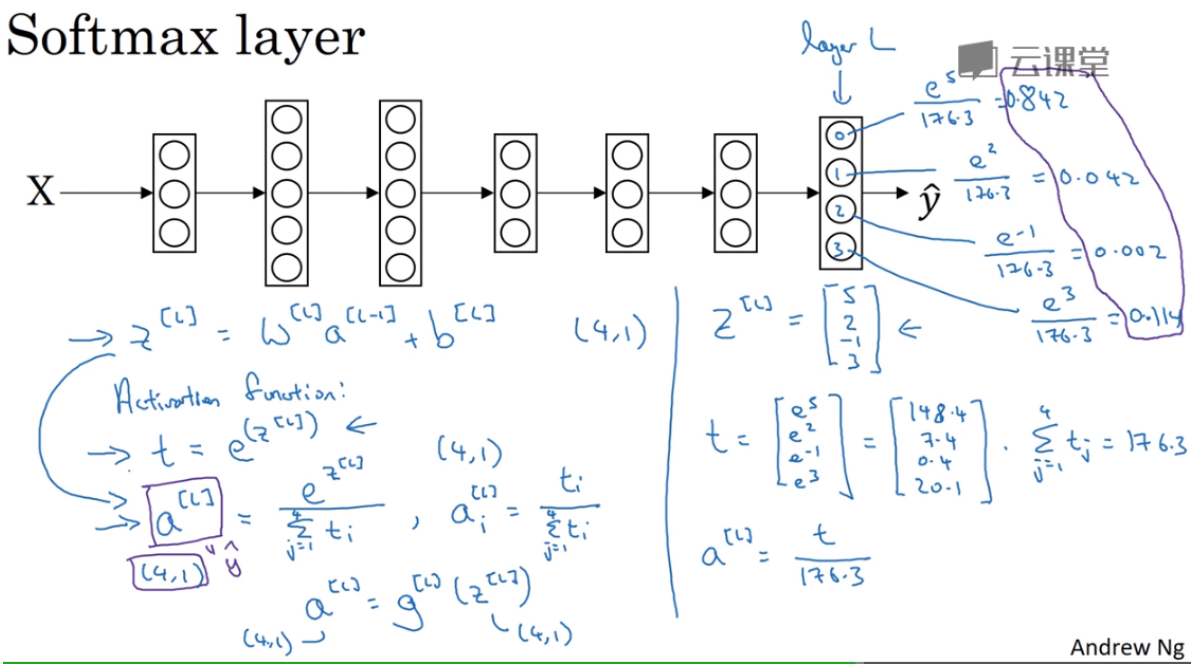
softmax回归是logistic回归的一般形式，它做的不只是二分分类，也可以做多分分类



现在我们想区分四个种类（class），0-其他、1-猫、2-狗、3-鸡

定义C为种类数，这里C=4，可以看到输出层有四个神经元，他们分别输出结果是0、1、2、3的概率，且总和为1 。可想而知，这样的输出结果是一个4*1的向量

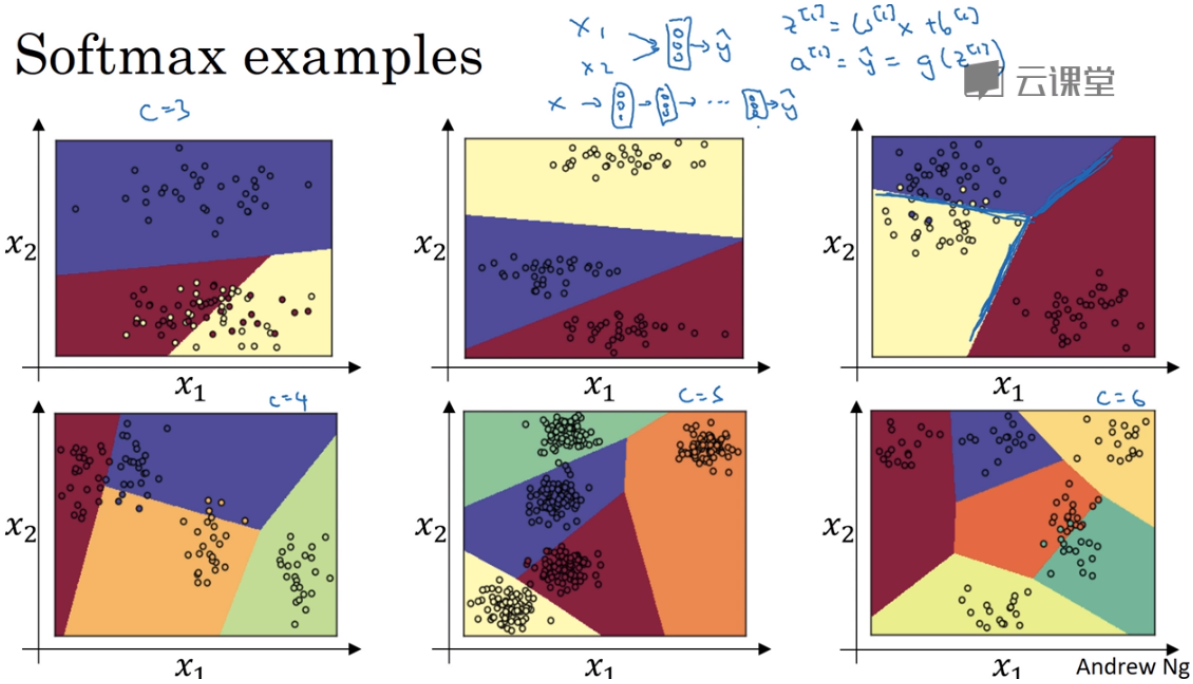
计算方法



左侧是softmax输出层激活函数计算方法，右侧是一个例子。

计算是先把z向量做一计算处理，得到同尺寸向量t，再对t进行归一化得到a，也就是yhat。具体可以看右侧的例子。

softmax回归举例



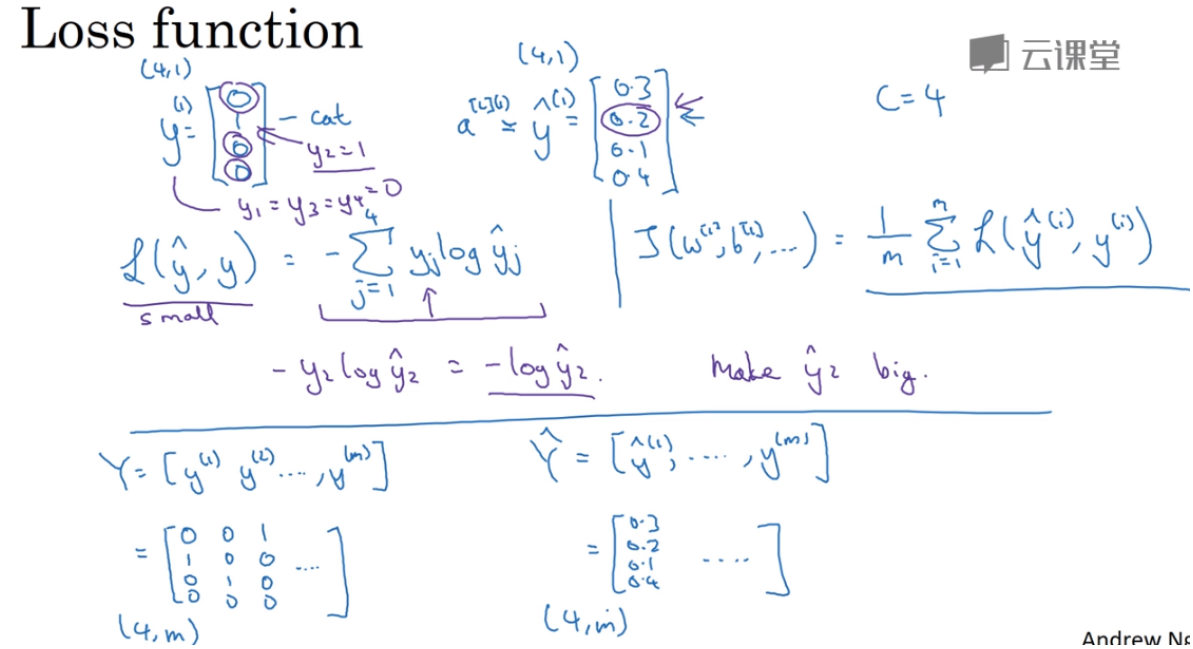
图中是没有隐藏层的softmax分类器对二维点集做的划分处理。由于没有隐藏层，所以划分是线性的。

（九）训练一个softmax分类器

损失函数

$$L(\hat{y},y)=-\sum_{i=1}^{C}[y_i log y_i+(1-y_i)log(1-\hat{y}_i)]$$

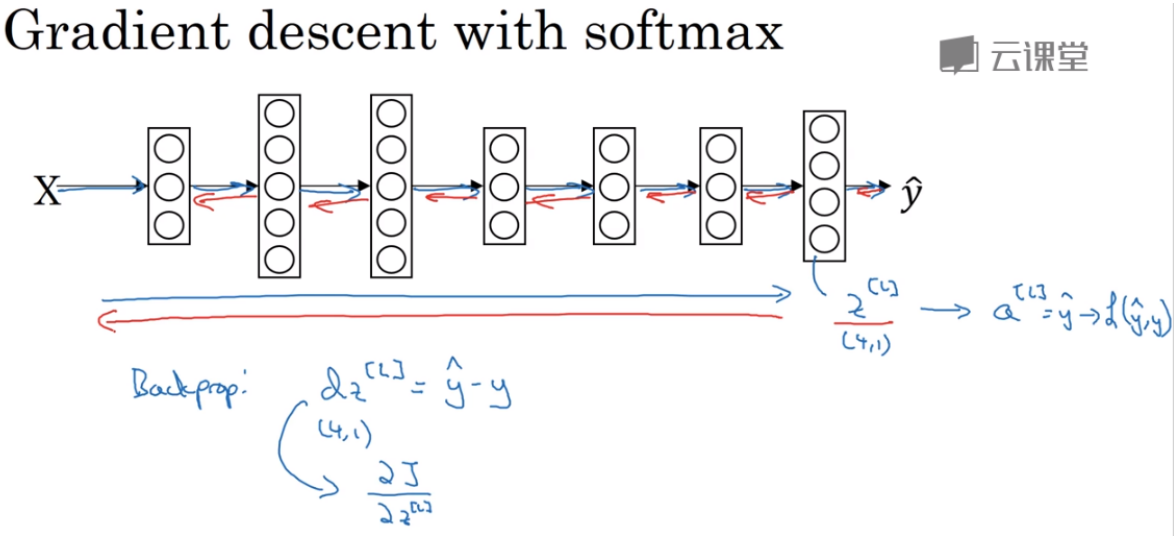
其中yhat和y是nx维向量。



如上图中紫色字迹举出的例子。假设训练集中$y^{(1)}$应该是0100，那么损失函数计算之后就等于$-logy_2^{(1)}$，要想使损失函数减小，就需要使$yhat_2^{(1)}$变大（接近1），这正是我们想要的。

梯度下降

和以前相同，只是以前的dz[L]是一维向量，现在是C维的了。



（十）Tensorflow

Tip

1. 如果电脑里安装的是v2版本的tensorflow，项目里要用的是v1，那么导包的时候应该

```
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
```

初始化变量

输入

```
a = tf.constant(2)
b = tf.constant(10)
c = tf.multiply(a,b)
print(c)
```

输出

```
Tensor("Mul:0", shape=(), dtype=int32)
```

创建session，执行表达式

输入

```
sess = tf.Session()
print(sess.run(c))
```

输出

```
20
```

创建placeholder（占位符）

placeholder只能在之后进行初始化，你可以通过使用“feed dictionary”来为placeholder传值。

```
tf.placeholder(
    dtype,
    shape=None,  #用[2,3]、[None,4]等表示
    name=None
)
```

输入

```
# Change the value of x in the feed_dict

x = tf.placeholder(tf.int64, name = 'x')
print(sess.run(2 * x, feed_dict = {x: 3}))#注意这
sess.close()
```

输出

```
6
```

当你创建变量、算式的时候，你只是让tensorflow在制作计算图，只有在run session的时候，计算图才真正被执行（在这之前你要先feed你的placeholder）

计算线性函数

```
# GRADED FUNCTION: linear_function

def linear_function():
    """
    Implements a linear function:
            Initializes W to be a random tensor of shape (4,3)
            Initializes X to be a random tensor of shape (3,1)
            Initializes b to be a random tensor of shape (4,1)
    Returns:
    result -- runs the session for Y = WX + b
    """

    np.random.seed(1)

    ### START CODE HERE ### (4 lines of code)
    X = np.random.randn(3,1)
    W = np.random.randn(4,3)
    b = np.random.randn(4,1)
    Y = tf.add(tf.matmul(W,X), b)
    ### END CODE HERE ###

    # Create the session using tf.Session() and run it with sess.run(...) on the
variable you want to calculate

    ### START CODE HERE ###
    sess = tf.Session()
    result = sess.run(Y)
    ### END CODE HERE ###
```

```
    # close the session
    sess.close()

    return result
```

计算sigmoid函数

```
# GRADED FUNCTION: sigmoid

def sigmoid(z):
    """
    Computes the sigmoid of z

    Arguments:
    z -- input value, scalar or vector

    Returns:
    results -- the sigmoid of z
    """

    ### START CODE HERE ### ( approx. 4 lines of code)
    # Create a placeholder for x. Name it 'x'.
    x = tf.placeholder(tf.float32, name = "x")

    # compute sigmoid(x)
    sigmoid = tf.sigmoid(x)

    # Create a session, and run it. Please use the method 2 explained above.
    # You should use a feed_dict to pass z's value to x.
    with tf.Session() as sess:
        # Run session and call the output "result"
        result = sess.run(sigmoid, feed_dict = {x: z})

    ### END CODE HERE ###

    return result
```

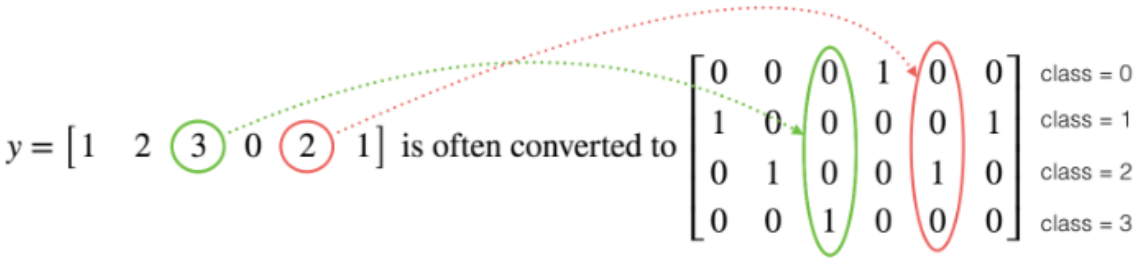
计算成本函数

```
# GRADED FUNCTION: cost

def cost(logits, labels):
    """
    Computes the cost using the sigmoid cross entropy

    Arguments:
    logits -- vector containing z, output of the last linear unit (before the
final sigmoid activation)
    labels -- vector of labels y (1 or 0)

    Note: What we've been calling "z" and "y" in this class are respectively
called "logits" and "labels"
```

```
    in the TensorFlow documentation. So logits will feed into z, and labels into
y.

    Returns:
    cost -- runs the session of the cost (formula (2))
    """

    ### START CODE HERE ###

    # Create the placeholders for "logits" (z) and "labels" (y) (approx. 2
lines)
    z = tf.placeholder(tf.float32, name = "z")
    y = tf.placeholder(tf.float32, name = "z")

    # Use the loss function (approx. 1 line)
    cost = tf.nn.sigmoid_cross_entropy_with_logits(logits = z, labels = y)

    # Create a session (approx. 1 line). See method 1 above.
    sess = tf.Session()

    # Run the session (approx. 1 line).
    cost = sess.run(cost, feed_dict = {z: logits, y: labels})

    # Close the session (approx. 1 line). See method 1 above.
    session.close()

    ### END CODE HERE ###

    return cost
```

独热编码

*One-Hot*编码,又称为一位有效编码,主要是采用N位状态寄存器来对N个状态进行编码,每个状态都由他独立的寄存器位,并且在任意时候只有一位有效。




```
def one_hot_matrix(labels, C):
    """
    Creates a matrix where the i-th row corresponds to the ith class number and
the jth column
                     corresponds to the jth training example. So if example j had
a label i. Then entry (i,j)
                     will be 1.

    Arguments:
    labels -- vector containing the labels
    C -- number of classes, the depth of the one hot dimension
```

```
    Returns:
    one_hot -- one hot matrix
    """

    ### START CODE HERE ###

    # Create a tf.constant equal to C (depth), name it 'C'. (approx. 1 line)
    C = tf.constant(C, name = 'C')

    # Use tf.one_hot, be careful with the axis (approx. 1 line)
    one_hot_matrix = tf.one_hot(labels, C, axis = 0)

    # Create the session (approx. 1 line)
    sess = tf.Session()

    # Run the session (approx. 1 line)
    one_hot = sess.run(one_hot_matrix)

    # Close the session (approx. 1 line). See method 1 above.
    sess.close()

    ### END CODE HERE ###

    return one_hot
```

其中axis表示插入的维度。这里是0，说明会把输入数组放在矩阵的第一维上。换种说法，就是把原来的数组改为一个“行向量”，这个向量每一个位置上放的是一个列向量，表示原来数组中对应位置的数。如以下例子：

输入

```
labels = np.array([1,2,3,0,2,1])
one_hot = one_hot_matrix(labels, C = 4)#axis在函数中已设为0
print ("one_hot = " + str(one_hot))
```

输出

```
one_hot = [[0. 0. 0. 1. 0. 0.]#1
 [1. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0.]]
```

用 0和1初始化

```
tf.ones(shape)
tf.zeros(shape)
```

```
# GRADED FUNCTION: ones

def ones(shape):
    """
```

```
    Creates an array of ones of dimension shape

    Arguments:
    shape -- shape of the array you want to create

    Returns:
    ones -- array containing only ones
    """

    ### START CODE HERE ###

    # Create "ones" tensor using tf.ones(...). (approx. 1 line)
    ones = tf.ones(shape)

    # Create the session (approx. 1 line)
    sess = tf.Session()

    # Run the session to compute 'ones' (approx. 1 line)
    ones = sess.run(ones)

    # Close the session (approx. 1 line). See method 1 above.
    sess.close()

    ### END CODE HERE ###
    return ones
```

input

```
print ("ones = " + str(ones([3])))
```

output

```
ones = [1. 1. 1.]
```