

第二篇第三周

1、调试处理：

学习率、指数加权平均（0.9）、mini-batch的大小，以确保最优算法运行有效，还要调试隐藏层单元，还有就是层数和学习率的衰减率有时也会产生影响。当使用Adam算法时，吴恩达从不调试 β_1 ， β_2 ， ϵ ，分别选择如下：

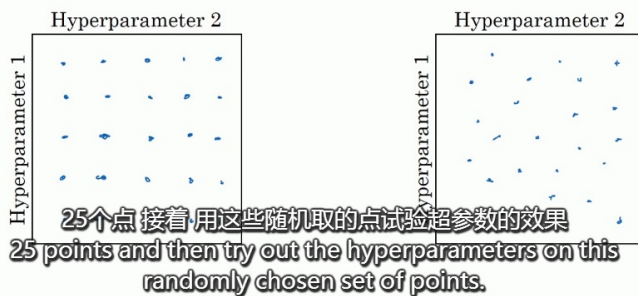
α
 β 0.9
 $\beta_1, \beta_2, \epsilon$
#layers
#hidden units
learning rate decay
mini batch size

我总是选定其分别为0.9，0.999和 10^{-8}

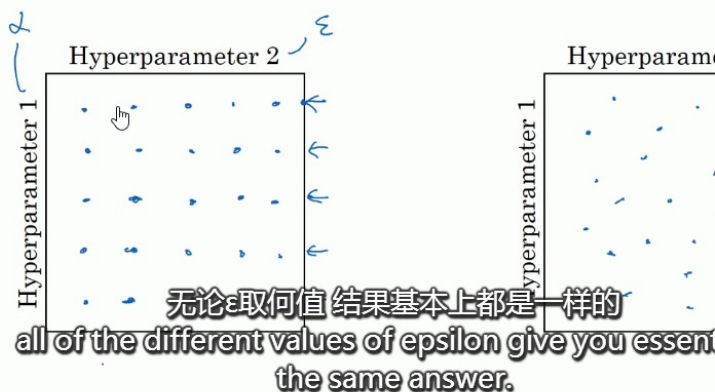
Pretty much I always use 0.9, 0.999 and tenth minus eight

最重要的参数是学习率 α ，调节超参数的技巧：

Try random values: Don't use a grid



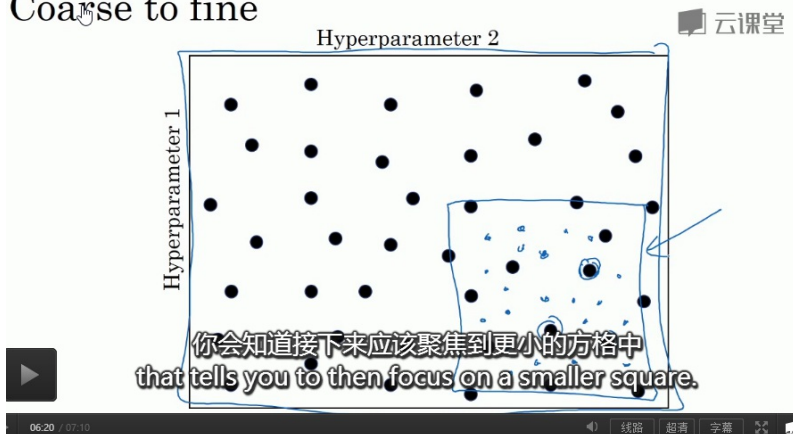
这样做的原因在于，你很难提前知道哪个超参数最重要，正如你之前看到的，一些超参数的确要比其他的更重要，假设超参数1是 α 学习速率，取一个极端的例子，假设超参数2，是Adam算法中 ϵ 的值，这种情况下， α 的取值很重要，而 ϵ 的取值无关紧要：



所以你知道有25中模型，但是进行试验的 α 只有5个，我认为这是很重要的，对比而言，如果你随机取值，你会试验25个独立的 α 值。

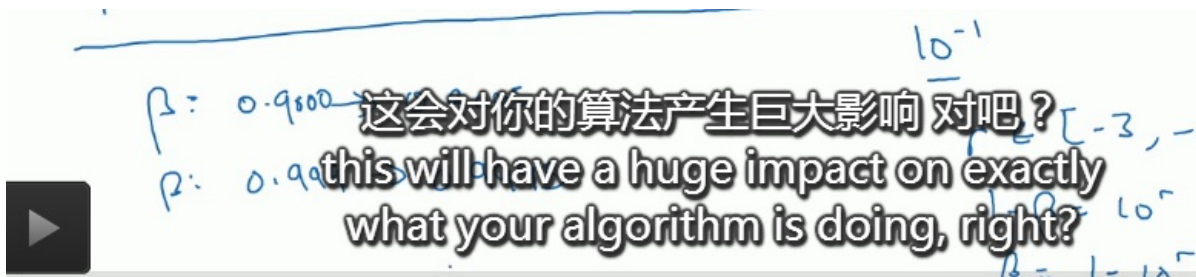
可以先进行粗略的搜索，然后聚焦到更小的方格中，也就是从粗到细的搜索，，

Coarse to fine



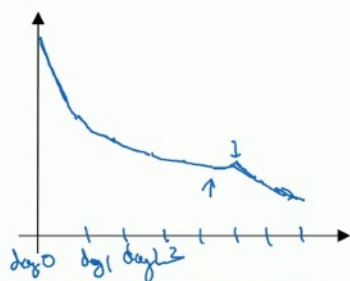
2、不采用线性取值的原因：

接近1的时候，一点小小的变化都可能对算法的影响很大。

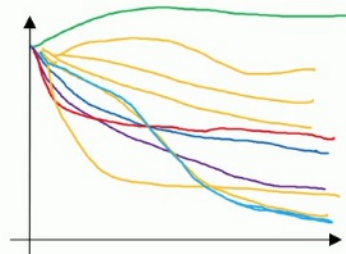


3、

Babysitting one model

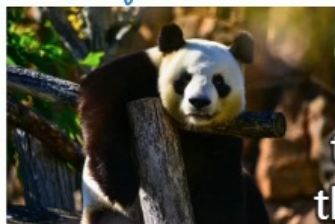
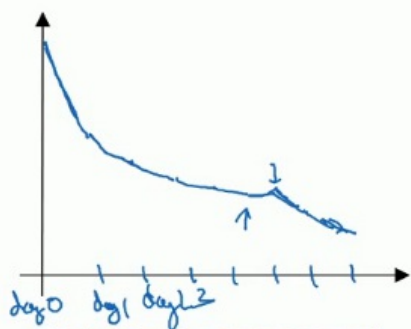


Training many models in parallel



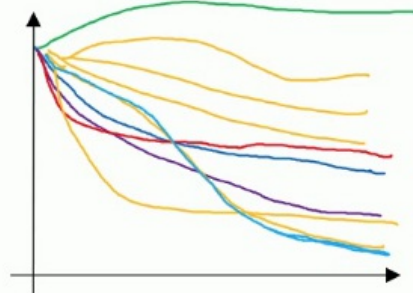
如果你拥有足够的计算机去平行试验许多模型
If you have enough computers to train a lot of models in parallel,

Babysitting one model



Panda

Training many models in parallel



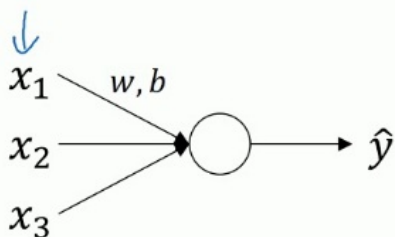
Caviar

那绝对采取鱼子酱方式
then by all means take
the caviar approach and

4、Batch归一化

batch归一化会使得你的参数搜索变得很容易，使神经网络对超参数的选择更加稳定，超参数的范围会变得更强大，工作效果也很好，也容易使你很容易训练，甚至是深层网络，我们还记得，归一化输入可以加快学习过程，根据方差归一化数据集，在深层神经网络中还可以做如下工作：

Normalizing inputs to speed up learning



$$\mu = \frac{1}{m} \sum_i x^{(i)}$$

$$X = X - \mu$$

$$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$$

$$X = X / \sigma^2$$

← element-wise



那归一化 $a^{[2]}$ 的平均值和方差岂不是很好？
then won't it be nice if you can normalize the mean and variance of $a^{[2]}$

以便使得 $W[3], b[3]$ 训练更加有意义，会影响 $W[3], b[3]$ 的计算，这就是batch归一化的作用，严格来说，我们归一化的不是 $a^{[2]}$ 而是 $z^{[2]}$ 。

在神经网络中，已知一些中间值，假设你有一些隐藏单元值，从 $z^{[1]}$ 到 $z^{[m]}$ ， z 为隐藏层，先计算所有 z 层的平均值，然后是所有 z 层共同的方差，然后就可以标准化（归一化）每一层了：

Given some intermediate values in NN $z^{(1)}, \dots, z^{(n)}$ $z^{[k]}(i)$

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

为了使数值稳定 通常将 ϵ 作为分母

For numerical stability, you usually add epsilon to denominator like that,



加上 ϵ 作为分母，以防止 σ 为零的情况，所以现在，我们把这些Z值都标准化了，化为含平均值为0和标准单位方差，所以Z的每个分量都含有平均值为0和方差1，但是我们不想让隐藏单元总是含有平均值为0和标准单位方差，也许隐藏单元有了不同的分布会有意义，所以我们需要做的运算就是计算 \hat{z} :

$$\hat{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

$$= \gamma z^{(i)}_{\text{norm}} + \beta$$

equals gamma $z^{(i)}_{\text{norm}}$ plus beta.



Given some intermediate values in NN $z^{(1)}, \dots, z^{(n)}$ $z^{[k]}(i)$

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\hat{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

这里 γ 和 β 是你模型的学习参数

And here gamma and beta are learnable parameters of your model.



所以我们使用梯度下降，或者类似梯度下降的算法，你会更新上面那两个参数，正如更新神经网络的权重一样：

Given some intermediate values in NN $z^{(1)}, \dots, z^{(n)}$ $z^{[k]}(i)$

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\hat{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

learnable parameters of model.

那 $\gamma z^{(i)}_{\text{norm}} + \beta$ 的作用在于

then the effect of gamma z_{norm} plus beta is



values in NN $z^{(1)}, \dots, z^{(n)}$

If $\sigma = \sqrt{\sigma^2 + \epsilon}$

$\beta = \mu$

then $\hat{z}^{(i)} =$

learnable parameters of model.

β

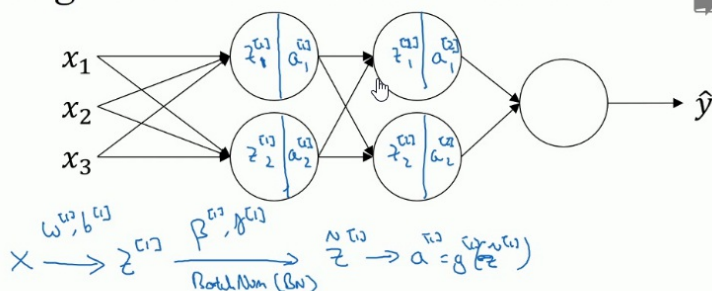
那么 $\hat{z}^{(i)} = z^{(i)}$

batch归一化的作用是它适用的归一化过程不只是输入层，甚至同样适用于神经网络的深度隐藏层。你可以使用batch归一化一些隐藏层单元值中的平均值和方差，不过训练输入和这些隐藏层单元值的一个区别是：你也许不想这些隐藏层单元的平均值为0，方差为1，假设你是使用Sigmoid函数，你并不想使得它落在0附近，也就是比较线性的那块区域，你想充分利用好非线性的Sigmoid函数，而不是使所有的值集中在线性版本中，这就是为什么有了 γ 和 β 之后，你可以确保所有的 $z^{(i)}$ 的值可以是你想要赋予的任意值。或者他的作用是确保隐藏层单元已使均值和方差标准化，这里均值和方差由两个参数控制，即 γ 和 β ，学习算法可以设置为任意值。

5、Batch归一化拟合进神经网络

Adding Batch Norm to a network

云课堂



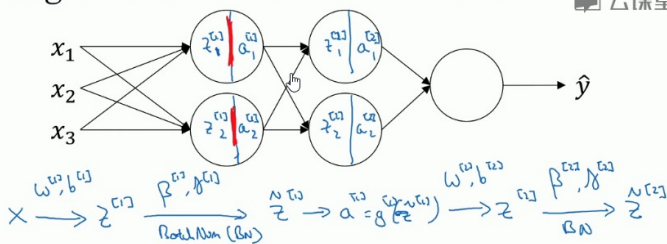
现在你已在第一层进行了计算

Now, you've done the computation for the first layer,

第一层计算如上图的笔记所示，此时，这项Batch归一化发生在 z 的计算和 a 之间，接下来，你需要应用 $a[1]$ 值计算 $z[2]$ ，此过程是由 $w[1]$ 和 $b[1]$ 控制的，与你在第一层所做的类似，你会将 $z[2]$ 进行Batch归一化，我们现在简称BN，如下图蓝色笔记：

Adding Batch Norm to a network

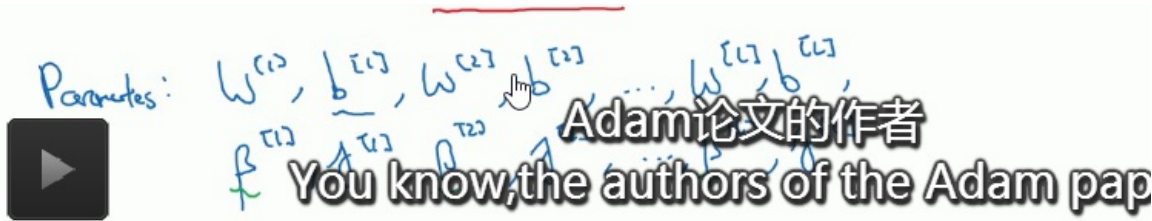
云课堂



现在你得到 $z^{(2)}$

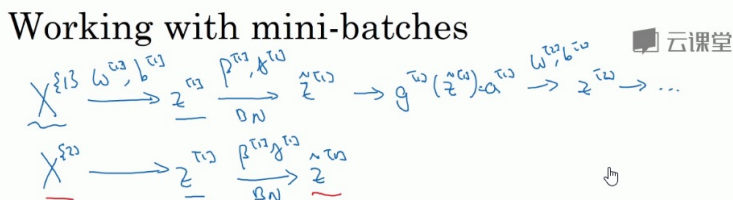
and now this gives you $z^{(2)}$

再通过激活函数计算 $a[2]$ ，所以，需要强调的是，Batch归一化发生在计算 z 和 a 之间，参数如下：



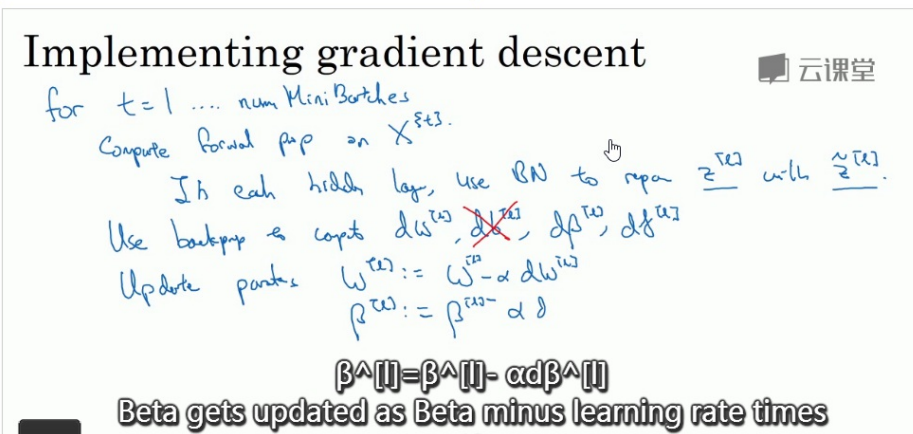
有了参数，你就可以选择你想要的优化算法，比如梯度下降等等，更新参数过程和之前的还是一样子，如果使用深度学习框架，把batch归一化步骤应于batch归一化层，只需要一个函数就解决问题，知道这个过程你就可以更好的理解代码。

Batch归一化经常和mini-batch一起使用：：



你用第二个mini-batch中的数据使归一化
You would be normalizing \tilde{z} using just the data in your second mini-batch,

过程如下，也是先正向传播，然后反向传播更新参数。：



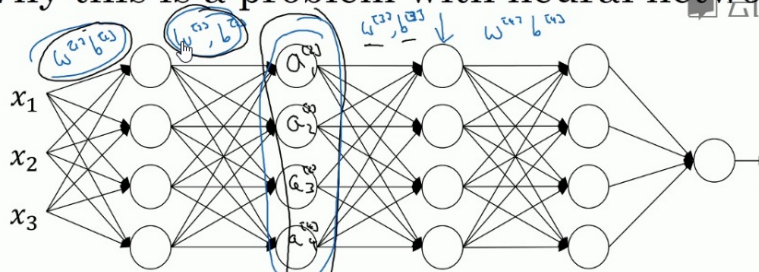
6、为什么Batch归一化会其作用：

其实跟前面的归一化原理类似，但是这里不仅仅是对于输入值，还对于隐藏层单元的值，这只是Batch归一化作用的冰山一角。

Batch归一化有效的另外一个重要原因是，它可以使权重，比你的网络更滞后或者更深层，比如第十层的权重更加能够经受得住变化，相比于神经网络中前层的权重，比如层一。

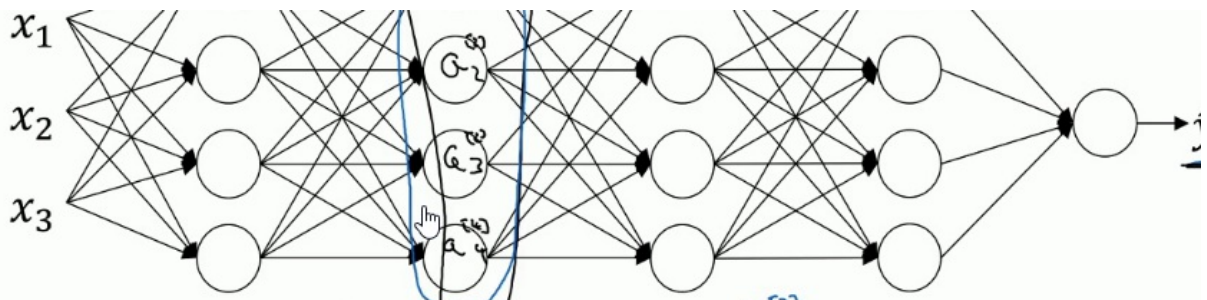
batch归一化做的，就是减少这些隐藏层分布变化的数量，如果是绘制这些隐藏层的单元值的分布，Batch归一化要求的是 $z[1], z[2]$ 可变：

Why this is a problem with neural network



batch归一化可确保 无论其怎样变化

But what batch norm ensures is that no matter how it changes,

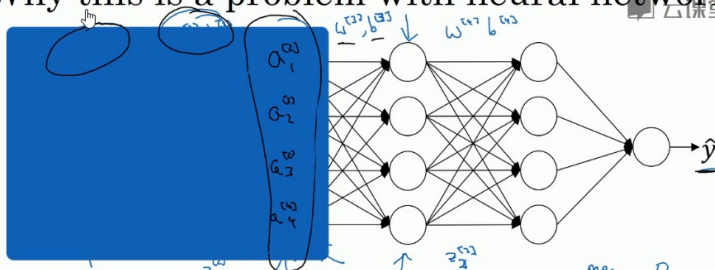


$z^{[2]}_1$ $z^{[2]}_2$ 的均值和方差保持不变

the mean and variance of $z^{[2]}_1$ and $z^{[2]}_2$ will remain the same.

但是方差和均值保持不变，并不一定是0或者1，也可以是 γ 和 β 决定的值，Batch归一化减少了输入值改变的问题：

Why this is a problem with neural network



它的确是这些值变得更稳定

it really causes these values to become more stable,

神经网络之后的层就会有更加坚实的基础，即使输入分布改变了一些，它会改变的更少，它做的是，当前层保持学习，当前层改变时，迫使后层，适应的程度变小了，你可以这样想，他减弱了前层参数的作用与后层参数的作用之间的联系，它使得神经网络的每一层都可以自己学习，稍稍独立于其他层，有助于加速整个网络的学习，重点Batch归一化的意思是，尤其从神经网络后一层的角度而言，前层不会左右移动很多，因为它们被同样的均值和方差限制，所以会使后层的学习工作变得更加容易一些。

另外一个作用是：它有轻微正则化效果，因为mini-batch并不是在全体数据上，所以会有噪声，这些噪声会传递下去，它有点像dropout，给隐藏层单元添加了噪声，这迫使后面的单元，不过分依赖任何一个隐藏层单元，因为添加的噪声微小，所以并不是巨大的正则化效果，所以有轻微正则化的效果，你可以将batch归一化和dropout一起使用，可以得到dropout更强大的正则化效果，另外，使用更大的mini-batch会减少噪声的存在，也会较小正则化的效果，最后需要注意的是，batch归一化一次只能处理一个mini-batch的数据，它在mini-batch上计算均值和方差。

7、测试时的Batch归一化：

batch归一化将你的数据以mini-batch的形式逐一处理，但在测试时，你可能需要对每个样本逐一处理：

Batch Norm at test time

云课堂

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2}}$$

这些就是用来执行Batch归一化的等式

here are the equations you'd use to implement batch



$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

norm.

在一个mini-batch中，你将mini-batch求和，计算均值，所以这里你只要把一个mini-batch的样本都加起来，然后计算方差，Znorm:

$$\rightarrow z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2}}$$

即用均值和标准差来调整 加上ε是为了数值稳定性



$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

scaling by the mean and standard deviation with Epsilon added for numerical stability.

$$\rightarrow z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

z是用γ和β再次调整z norm



$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

And then Z tilde is taking Z norm and rescaling by gamma and beta.

注意：σ²和μ是在整个mini-batch上进行计算的。

测试时，你需要单独计算σ²和μ，你需要用一个指数加权平均来估算，这个平均数涵盖了所有的mini-batch:

Batch Norm at test time

云课堂

$$\rightarrow \mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\rightarrow \sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2}}$$

那么在为L层训练X^[1]时 你就得到了μ^[L]

So, when training on X^[1] for that layer L, you get some

每一个batch都得到一个μ，你对某一层隐藏层Z均值的估值，同样的你可以用指数加权平均来追踪，你在这一层的第一个mini-batch中所见到的σ²的值，第二个mini-batch中所见到的σ²的值：

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sigma}$$

$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)}$$

weighted average across mini-batches

能够得到你所查看的每一层的 μ 和 σ^2 的平均数的实时数值
seeing for each layer as you train the neural network
across different mini batches.

测试的时候， σ^2 和 μ 用指数加权平均就可以了。

总结一下：在训练时， σ^2 和 μ 是在整个mini-batch上计算出来的，包含了一定数量的样本，但是在测试时，你可能需要逐一处理样本，方法是根据你的训练集估算 σ^2 和 μ ，估算的方法有很多，你可以在最终的神经网络中运行整个训练集，来得到 σ^2 和 μ ，但是在实际应用中，我们通常应用指数加权平均，来追踪在训练过程中你看到的 σ^2 和 μ 的值，然后用测试中 σ^2 和 μ 的值，来进行你需要的隐藏层单元 z 值的调整，如果你用的是深度学习框架，会有默认的方式估计 σ^2 和 μ 的值，使用batch归一化，你可以训练更深的网络，让你的学习算法运行的更快。

8、Softmax回归

可以用于进行多分类问题，例子：



3 1 2 0 3 2 0 1

我把猫叫做类1 狗为类2 小鸡是类3
So I'm going to call cats class 1, dogs class 2, baby chicks class 3.

如果不属于上面的某一类，我们把它归为其他类别，或者叫做类0，用 C 表示你的输入会被分入的类别总个数，这个例子 $C=4$ ，(0,1,2,3)在这个例子中，我们建立一个神经网络，其输出层为4：

$$C = \# \text{classes} = 4 \quad (0, \dots, 3)$$



因此 n 即输出层也就是 L 层的单元数量
So n , the number of units output layer which is layer L

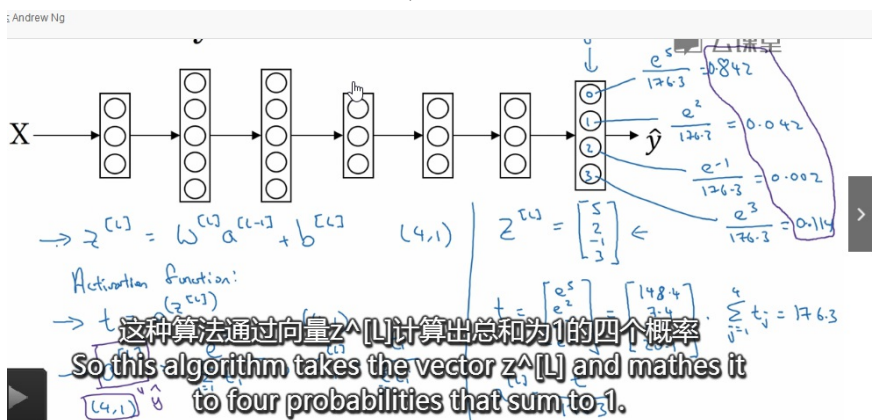
每一个输出层代表的是某一个类别的概率，在这个例子中，是一个 4×1 的向量，向量里面的数值加起来是

1, 你用softmax,以及输出层来生成输出:

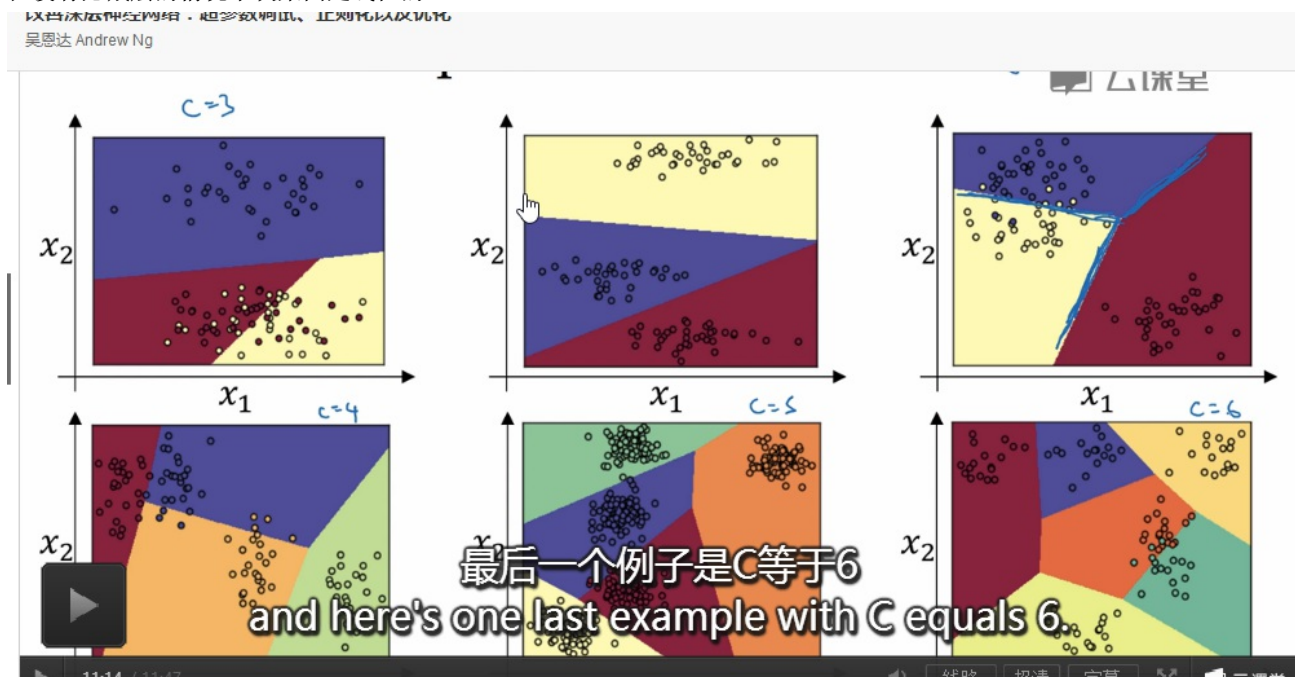
4.1) $z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$

$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix} \cdot \sum_{j=1}^4 t_j = 17.$

然后归一化。得到向量的总和为1,:



在没有隐藏层的情况下决策面是线性的:



有隐藏层, 就可以学习更复杂的非线性决策边界, 来区分多种不同的分类,

9、训练一个softmax分类器

上一层的例子:

Understanding softmax

云课堂

(4,1)

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

C=4

$g^{(L)}(\cdot)$

$$g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5 / (e^5 + e^2 + e^{-1} + e^3) \\ e^2 / (e^5 + e^2 + e^{-1} + e^3) \\ e^{-1} / (e^5 + e^2 + e^{-1} + e^3) \\ e^3 / (e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

简单来说就是用临时变量t将它归一化

It's basically taking the temporary variable t

hard max是将最大的设为1，其他设置为0，

Understanding softmax

云课堂

(4,1)

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

C=4

$g^{(L)}(\cdot)$

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5 / (e^5 + e^2 + e^{-1} + e^3) \\ e^2 / (e^5 + e^2 + e^{-1} + e^3) \\ e^{-1} / (e^5 + e^2 + e^{-1} + e^3) \\ e^3 / (e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

"hard max"

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

与hard max正好相反

all this in contrast to the hard max.

怎么训练带softmax输出层的神经网络：

1、先定义损失函数。

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \leftarrow \text{cat} \quad a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \leftarrow$$

这实际上是一只猫 但却只分配到20%是猫的概率

because this is actually a cat and assigned only a 20% chance that this is a cat.

所以这是个不佳的分类器了。

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \leftarrow \text{cat} \quad a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \leftarrow$$

$$L(\hat{y}, y) = - \sum_{j=1}^C y_j \log \hat{y}_j$$

$$- y_1 \log \hat{y}_1 = - \log \hat{y}_2 \quad \text{Make } \hat{y}_2 \text{ big.}$$

$$J(w^{(L)}, b^{(L)}, \dots) = \frac{1}{n} \sum_{i=1}^n L(\hat{y}^{(i)}, y^{(i)})$$

那么y就是0.3 0.2 0.1 0.4等等
Then y hat with these 0.3, 0.2, 0.1, and 0.4, and so on.

10、Tensorflow:

```
In [1]: import numpy as np
import tensorflow as tf
```

```
In [5]: w = tf.Variable(0, dtype=tf.float32)
#cost = tf.add(tf.add(w**2, tf.multiply(-10., w)), 25)
cost = w**2 - 10*w + 25
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))

0.0
```

```
In [6]: session.run(w)
print(session.run(w))
```

```
In [7]: for i in range(1000):
        session.run(train)
        print(session.run(w))
```

TensorFlow还有一个特点我想告诉你
Now, there's just one more feature of TensorFlow that
I want to show you,