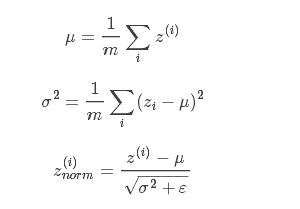
1. Batch Normalization-隐层的输入标准化处理

Batch Normalization：在神经网络中，第l层隐藏层的输入就是第l−1层隐藏层的输出A[l−1]。对A[l−1]进行标准化处理，从原理上来说可以提高W[l]和b[l]的训练速度和准确度。这种对各隐藏层的标准化处理就是Batch Normalization。值得注意的是，实际应用中，一般是对Z[l−1]进行标准化处理而不是A[l−1]，其实差别不是很大。

**Batch Normalization公式：**

对第l层隐藏层的输入Z[l−1]做如下标准化处理，忽略上标[l−1]：



其中，m是单个mini-batch包含样本个数，ε是为了防止分母为零，可取值10−8。这样，使得该隐藏层的所有输入z(i)均值为0，方差为1。

但是，大部分情况下并不希望所有的z(i)均值都为0，方差都为1，也不太合理。通常需要对z(i)进行进一步处理：



上式中，γ和β是learnable parameters，类似于W和b一样，可以通过梯度下降等算法求得。这里，γ和β的作用是让z~(i)的均值和方差为任意值，只需调整其值就可以了。例如，令：

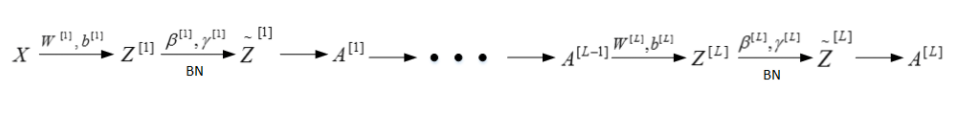


则z~(i)=z(i)，即identity function。可见，设置γ和β为不同的值，可以得到任意的均值和方差。这样，通过Batch Normalization，对隐藏层的各个z[l](i)进行标准化处理，得到z~[l](i)，替代z[l](i)。

**注意：**输入的标准化处理Normalizing inputs和隐藏层的标准化处理Batch Normalization是有区别的。Normalizing inputs使所有输入的均值为0，方差为1。而Batch Normalization可使各隐藏层输入的均值和方差为任意值。实际上，从激活函数的角度来说，如果各隐藏层的输入均值在靠近0的区域即处于激活函数的线性区域，这样不利于训练好的非线性神经网络，得到的模型效果也不会太好。这也解释了为什么需要用γ和β来对z[l](i)作进一步处理。

1. Batch Normalization应用于整个神经网络中

把Bath Norm应用到整个神经网络中。对于L层神经网络，经过Batch Norm的作用，整体流程如下：

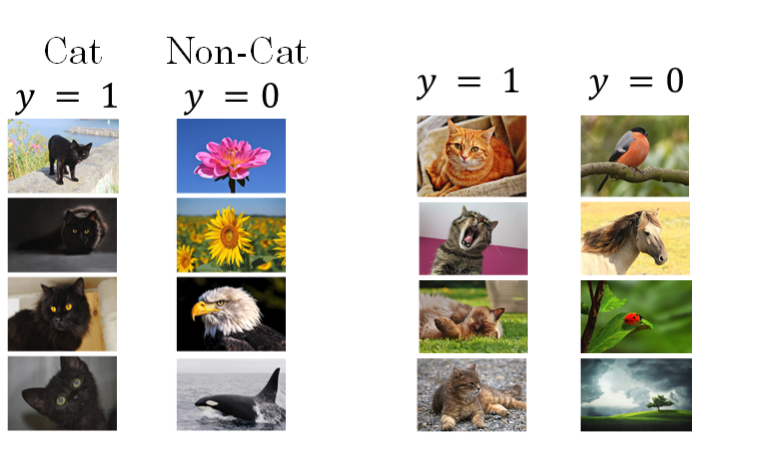


**注意：**实际上，Batch Norm经常使用在mini-batch上，这也是其名称的由来。因为Batch Norm对各隐藏层Z[l]=W[l]A[l−1]+b[l]有去均值的操作，所以这里的常数项b[l]可以消去，其数值效果完全可以由Z~[l]中的β来实现。因此，我们在使用Batch Norm的时候，可以忽略各隐藏层的常数项b[l]。在使用梯度下降算法时，分别对W[l]，β[l]和γ[l]进行迭代更新。除了传统的梯度下降算法之外，还可以使用我们之前介绍过的动量梯度下降、RMSprop或者Adam等优化算法。

1. 为什么Batch Normalization应用于整个神经网络中有效

我们可以把输入特征做均值为0，方差为1的规范化处理，来加快学习速度。而Batch Norm也是对隐藏层各神经元的输入做类似的规范化处理。总的来说，Batch Norm不仅能够提高神经网络训练速度，而且能让神经网络的权重W的更新更加“稳健”，尤其在深层神经网络中更加明显。比如神经网络很后面的W对前面的W包容性更强，即前面的W的变化对后面W造成的影响很小，整体网络更加健壮。

**例如：**假如用一个浅层神经网络（类似逻辑回归）来训练识别猫的模型。如下图所示，提供的所有猫的训练样本都是黑猫。然后，用这个训练得到的模型来对各种颜色的猫样本进行测试，测试的结果可能并不好。其原因是训练样本不具有一般性（即不是所有的猫都是黑猫），这种训练样本（黑猫）和测试样本（猫）分布的变化称之为covariate shift。



对于这种情况，如果实际应用的样本与训练样本分布不同，即发生了covariate shift，则一般是要对模型重新进行训练的。在神经网络，尤其是深度神经网络中，covariate shift会导致模型预测效果变差，重新训练的模型各隐藏层的W[l]和B[l]均产生偏移、变化。而Batch Norm的作用恰恰是减小covariate shift的影响，让模型变得更加健壮，鲁棒性更强。Batch Norm减少了各层W[l]、B[l]之间的耦合性，让各层更加独立，实现自我训练学习的效果。也就是说，如果输入发生covariate shift，那么因为Batch Norm的作用，对个隐藏层输出Z[l]进行均值和方差的归一化处理，W[l]和B[l]更加稳定，使得原来的模型也有不错的表现。针对上面这个黑猫的例子，如果我们使用深层神经网络，使用Batch Norm，那么该模型对花猫的识别能力应该也是不错的。

Batch Norm也起到轻微的正则化效果。具体表现在：

每个mini-batch都进行均值为0，方差为1的归一化操作；每个mini-batch中，对各个隐藏层的Z[l]添加了随机噪声，效果类似于Dropout；mini-batch越小，正则化效果越明显。但是，Batch Norm的正则化效果比较微弱，正则化也不是Batch Norm的主要功能。

1. Batch Normalization应用于单样本

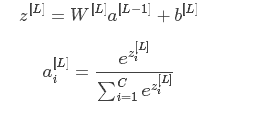
在单个mini-batch中，μ和σ2是对所有m个样本求和得到的。在测试过程中，如果只有一个样本，求其均值和方差是没有意义的，就需要对μ和σ2进行估计。估计的方法有很多，理论上我们可以将所有训练集放入最终的神经网络模型中，然后将每个隐藏层计算得到的μ[l]和σ2[l]直接作为测试过程的μ和σ2来使用。但是，实际应用中一般不使用这种方法，而是使用我们之前介绍过的指数加权平均的方法来预测测试过程单个样本的μ和σ2。

指数加权平均的做法很简单，对于第l层隐藏层，考虑所有mini-batch在该隐藏层下的μ[l]和σ2[l]，然后用指数加权平均的方式来预测得到当前单个样本的μ[l]和σ2[l]。这样就实现了对测试过程单个样本的均值和方差估计。最后，再利用训练过程得到的γ和β值计算出各层的z~(i)值。

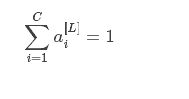
1. softmax分类器

1.应用于多分类问题，用C表示种类个数，神经网络中输出层就有C个神经元，即n[L]=C。其中，每个神经元的输出依次对应属于该类的概率，即P(y=c|x)。为了处理多分类问题，我们一般使用Softmax回归模型。

Softmax回归模型输出层的激活函数如下所示：



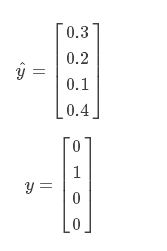
输出层每个神经元的输出ai[L]对应属于该类的概率，满足：



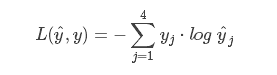
所有的ai[L]，即y^，维度为(C, 1)。

1. 训练一个softmax分类器

softmax分类器的loss function。举例来说，假如C=4，某个样本的预测输出y^和真实输出y为：



从y^值来看，P(y=4|x)=0.4，概率最大，而真实样本属于第2类，因此该预测效果不佳。我们定义softmax分类器的loss function为：

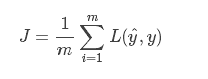


由于只有当j=2时，y2=1，其它情况下yj=0。所以，上式中的L(y^,y)可以简化为：



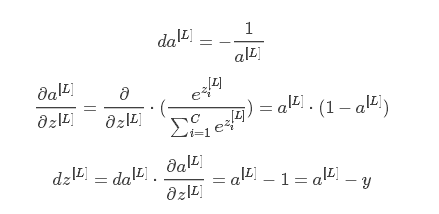
要让L(y^,y)更小，就应该让y^2越大越好。y^2反映的是概率，完全符合我们之前的定义。

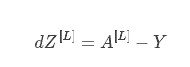
故所有m个样本的cost function为：



其预测输出向量A[L]即Y^的维度为(4, m)。

softmax分类器的**反向传播过程**仍然使用梯度下降算法，其推导过程与二元分类有一点点不一样。因为只有输出层的激活函数不一样，我们先推导dZ[L]：



对于m个样本：

1. 深度学习框架



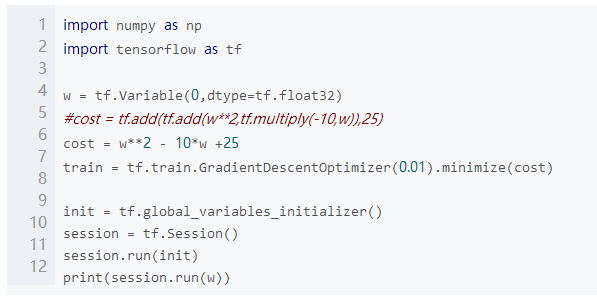
一般选择深度学习框架的基本准则是：

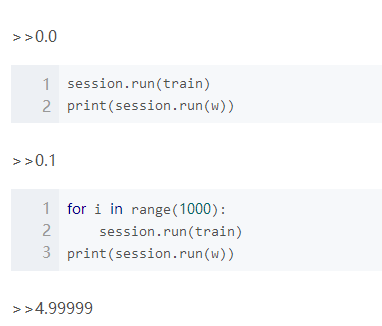
1. 易于编程
2. 运行速度快
3. 真正开源
4. 深度学习框架-tensorflow

例如cost function是参数w的函数：



如果使用TensorFlow对cost function进行优化，求出最小值对应的w，程序如下：





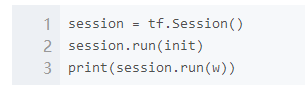
TensorFlow框架内可以直接调用梯度下降优化算法，不需要我们自己再写程序了，大大提高了效率。在运行1000次梯度下降算法后，w的解为4.99999，已经非常接近w的最优值5了。

针对上面这个例子，如果对w前的系数用变量x来代替，程序如下：

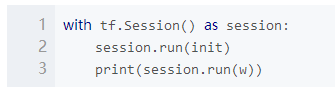


结果跟之前是一样的。除此之外，我们还可以更改x即cofficients的值，而得到不同的优化结果w。

另外，上段程序中的：



有另外一种写法：



TensorFlow的最大优点就是采用数据流图来进行数值运算。图中的节点表示数学操作，图中的线则表示在节点间相互联系的多维数据数组，即张量。而且它灵活的架构让你可以在多种平台上展开计算，例如台式计算机中的一个或多个CPU（或GPU），服务器，移动设备等等。