**前言**

　　训练神经网络模型时，如果训练样本较少，为了防止模型过拟合，Dropout可以作为一种trikc供选择。Dropout是hintion最近2年提出的，源于其文章*Improving neural networks by preventing co-adaptation of feature detectors*.中文大意为：通过阻止特征检测器的共同作用来提高神经网络的性能。本篇博文就是按照这篇论文简单介绍下Dropout的思想，以及从用一个简单的例子来说明该如何使用dropout。

**基础知识：**

　　Dropout是指在模型训练时随机让网络某些隐含层节点的权重不工作，不工作的那些节点可以暂时认为不是网络结构的一部分，但是它的权重得保留下来（只是暂时不更新而已），因为下次样本输入时它可能又得工作了（有点抽象，具体实现看后面的实验部分）。

　　按照hinton的文章，他使用Dropout时训练阶段和测试阶段做了如下操作：

　　在样本的训练阶段，在没有采用pre-training的网络时（Dropout当然可以结合pre-training一起使用），hintion并不是像通常那样对权值采用L2范数惩罚，而是对每个隐含节点的权值L2范数设置一个上限bound，当训练过程中如果该节点不满足bound约束，则用该bound值对权值进行一个规范化操作（即同时除以该L2范数值），说是这样可以让权值更新初始的时候有个大的学习率供衰减，并且可以搜索更多的权值空间（没理解）。

　　在模型的测试阶段，使用”mean network(均值网络)”来得到隐含层的输出，其实就是在网络前向传播到输出层前时隐含层节点的输出值都要减半（如果dropout的比例为50%），其理由文章说了一些，可以去查看（没理解）。

　　关于Dropout，文章中没有给出任何数学解释，Hintion的直观解释和理由如下：

　　1. 由于每次用输入网络的样本进行权值更新时，隐含节点都是以一定概率随机出现，因此不能保证每2个隐含节点每次都同时出现，这样权值的更新不再依赖于有固定关系隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况。

　　2. 可以将dropout看作是模型平均的一种。对于每次输入到网络中的样本（可能是一个样本，也可能是一个batch的样本），其对应的网络结构都是不同的，但所有的这些不同的网络结构又同时share隐含节点的权值。这样不同的样本就对应不同的模型，是bagging的一种极端情况。个人感觉这个解释稍微靠谱些，和bagging，boosting理论有点像，但又不完全相同。

　　3. native bayes是dropout的一个特例。Native bayes有个错误的前提，即假设各个特征之间相互独立，这样在训练样本比较少的情况下，单独对每个特征进行学习，测试时将所有的特征都相乘，且在实际应用时效果还不错。而Droput每次不是训练一个特征，而是一部分隐含层特征。

　　4. 还有一个比较有意思的解释是，Dropout类似于性别在生物进化中的角色，物种为了使适应不断变化的环境，性别的出现有效的阻止了过拟合，即避免环境改变时物种可能面临的灭亡。

　　文章最后当然是show了一大把的实验来说明dropout可以阻止过拟合。这些实验都是些常见的benchmark，比如Mnist, Timit, Reuters, CIFAR-10, ImageNet.

**实验过程：**

　　本文实验时用mnist库进行手写数字识别，训练样本2000个，测试样本1000个，用的是matlab的<https://github.com/rasmusbergpalm/DeepLearnToolbox>，代码在test\_example\_NN.m上修改得到。关于该toolbox的介绍可以参考网友的博文[【面向代码】学习 Deep Learning（一）Neural Network](http://blog.csdn.net/dark_scope/article/details/9421061)。这里我只用了个简单的单个隐含层神经网络，隐含层节点的个数为100，所以输入层-隐含层-输出层节点依次为784-100-10. 为了使本例子简单话，没用对权值w进行规则化，采用mini-batch训练，每个mini-batch样本大小为100，迭代20次。权值采用随机初始化。

**实验结果：**

**没用Dropout时：**

　　训练样本错误率(均方误差)：**0.032355**

　　测试样本错误率：**15.500%**

**使用Dropout时：**

　　训练样本错误率（均方误差）：**0.075819**

　　测试样本错误率：**13.000%**

　　可以看出使用Dropout后，虽然训练样本的错误率较高，但是训练样本的错误率降低了，说明Dropout的泛化能力不错，可以防止过拟合。

**实验主要代码及注释：**

***test\_dropout.m:***

[复制代码](javascript:void(0);)

%% //导入minst数据并归一化

load mnist\_uint8;

train\_x = double(train\_x(1:2000,:)) / 255;

test\_x = double(test\_x(1:1000,:)) / 255;

train\_y = double(train\_y(1:2000,:));

test\_y = double(test\_y(1:1000,:));

% //normalize

[train\_x, mu, sigma] = zscore(train\_x);% //归一化train\_x,其中mu是个行向量,mu是个列向量

test\_x = normalize(test\_x, mu, sigma);% //在线测试时，归一化用的是训练样本的均值和方差，需要特别注意

%% //without dropout

rng(0);

nn = nnsetup([784 100 10]);% //初步构造了一个输入-隐含-输出层网络，其中包括了

% //权值的初始化，学习率，momentum，激发函数类型，

% //惩罚系数，dropout等

opts.numepochs = 20; % //Number of full sweeps through data

opts.batchsize = 100; % //Take a mean gradient step over this many samples

[nn, L] = nntrain(nn, train\_x, train\_y, opts);

[er, bad] = nntest(nn, test\_x, test\_y);

str = sprintf('testing error rate is: %f',er);

disp(str)

%% //with dropout

rng(0);

nn = nnsetup([784 100 10]);

nn.dropoutFraction = 0.5; % //Dropout fraction，每一次mini-batch样本输入训练时，随机扔掉50%的隐含层节点

opts.numepochs = 20; % //Number of full sweeps through data

opts.batchsize = 100; % //Take a mean gradient step over this many samples

nn = nntrain(nn, train\_x, train\_y, opts);

[er, bad] = nntest(nn, test\_x, test\_y);

str = sprintf('testing error rate is: %f',er);

disp(str)

[复制代码](javascript:void(0);)

　　下面来分析与dropout相关的代码，集中在上面test.m代码的后面with drop部分。首先在训练过程中需要将神经网络结构nn的dropoutFraction设置为一定比例，这里设置为50%：nn.dropoutFraction = 0.5;

　　然后进入test\_dropout.m中的nntrain()函数，没有发现与dropoutFraction相关的代码，继续进入网络前向传播函数nnff()函数中，在网络的隐含层节点激发函数值被计算出来后，有下面的代码：

[复制代码](javascript:void(0);)

　　　　if(nn.dropoutFraction > 0)

if(nn.testing)

nn.a{i} = nn.a{i}.\*(1 - nn.dropoutFraction);

else

nn.dropOutMask{i} = (rand(size(nn.a{i}))>nn.dropoutFraction);

nn.a{i} = nn.a{i}.\*nn.dropOutMask{i};

end

end

[复制代码](javascript:void(0);)

      由上面的代码可知，隐含层节点的输出值以dropoutFraction百分比的几率被随机清0（注意此时是在训练阶段，所以是else那部分的代码），既然前向传播时有些隐含节点值被清0了，那么在误差方向传播时也应该有相应的处理，果然，在反向传播函数nnbp()中，有下面的代码：

　　　　if(nn.dropoutFraction>0)

d{i} = d{i} .\* [ones(size(d{i},1),1) nn.dropOutMask{i}];

end

　　也就是说计算节点误差那一项时，其误差项也应该清0。从上面可以看出，使用dropout时，其训练部分的代码更改很少。

**(**有网友发私信说，反向传播计算误差项时可以不用乘以dropOutMask{i}矩阵，后面我仔细看了下bp的公式，一开始也感觉不用乘有道理。因为源码中有为：

[复制代码](javascript:void(0);)

for i = 1 : (n - 1)

if i+1==n

nn.dW{i} = (d{i + 1}' \* nn.a{i}) / size(d{i + 1}, 1);

else

nn.dW{i} = (d{i + 1}(:,2:end)' \* nn.a{i}) / size(d{i + 1}, 1);

end

end

[复制代码](javascript:void(0);)

　　代码进行权重更新时，由于需要乘以nn.a{i},而nn.a{i}在前向过程中如果被mask清掉的话（使用了dropout前提下），则已经为0了。但其实这时错误的，因为对误差

敏感值作用的是与它相连接的前一层权值，并不是本层的权值，而本层的输出a只对它的下一层权值更新有效。**)**

　　再来看看测试部分，测试部分如hintion论文所说的，采用mean network,也就是说前向传播时隐含层所有节点的输出同时减小dropoutFraction百分比，即保留(1- dropoutFraction)百分比，代码依旧是上面贴出的nnff()函数里满足if(nn.testing)的部分：

[复制代码](javascript:void(0);)

　　　　if(nn.dropoutFraction > 0)

if(nn.testing)

nn.a{i} = nn.a{i}.\*(1 - nn.dropoutFraction);

else

nn.dropOutMask{i} = (rand(size(nn.a{i}))>nn.dropoutFraction);

nn.a{i} = nn.a{i}.\*nn.dropOutMask{i};

end

end

[复制代码](javascript:void(0);)

 　　上面只是个简单的droput实验，可以用来帮助大家理解dropout的思想和使用步骤。其中网络的参数都是采用toolbox默认的，并没有去调整它，如果该实验将训练样本增大，比如6w张，则参数不变的情况下使用了dropout的识别率还有可能会降低（当然这很有可能是其它参数没调到最优，另一方面也说明在样本比较少的情况下，droput确实可以防止过拟合），为了体现droput的优势，这里我只用了2000张训练样本。

**参考资料：**

　　Hinton, G. E., et al. (2012). "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580.

<https://github.com/rasmusbergpalm/DeepLearnToolbox>

[【面向代码】学习 Deep Learning（一）Neural Network](http://blog.csdn.net/dark_scope/article/details/9421061)