# 作业5:自动编码器神经网络的实现(选做)

## 1 作业说明

本次作业不做硬性要求，仅供有兴趣的同学进行尝试，完成本次作业的同学将在期末总分基础上获得额外最高3分奖励。

这次作业的目的是帮助各位同学重温神经网络反向传播算法的推导，学习“矩阵-向量化”的编程方法，完成一个“自动编码器”的设计与实现。通过可视化神经网络的学习结果，大家能更深入地理解神经网络的基本原理(比如说，神经网络是如何做到同时完成特征选择和预测这两个任务的)。相信大家在完成本次作业后会有很大收获。

作业内容摘录改编自斯坦福大学的深度学习教程：`http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial` ，感兴趣的同学可以进一步学习。

在题目中，激活函数采用sigmoid 函数：

$$f(z) = \frac{1}{1 + \exp(-z)} \tag{1}$$

## 2 稀疏自动编码器的实现

在本问题中，你将实现稀疏自动编码器的算法，展示出它发掘的各种图像边沿特征。这种特殊神经网络的训练目标是使得任意输入一个样本后的输出与输入差异最小，中间层的输出可以看作是输入的一个压缩编码(希望详细了解的同学可以阅读文献[1,2,5,7])。其中的程序框架已经用Matlab 实现了(sparseae_exercise.zip)，你需要完成几个主要函数的实现，这些地方都标注了("YOUR CODE HERE")。你需要完成以下文件：sampleIMAGES.m, sparseAutoencoderCost.m, computeNumericalGradient.m。在启动程序文件train.m 中展示了上述函数的调用方法和次序，你只需运行这个文件即可得到最后的结果。

要求：请提交所生成的图片和所有代码(请维持原来的文件结构，包括所有提供的代码，保证无需修改即可执行)。

### 2.1 产生训练集：将代码sampleIMAGES.m补充完整

**(1)**在10幅图片中随机抽取一个$8 \times 8$ 像素大小的图片，并转化为$n = 64$ 维向量(行主序和列主序均可)，这样就可以得到一个训练样本$x$。总共采集$N = 10000$ 幅这样的图片,得到一个$64 \times 10000$的矩阵，即样本集合为$T = \{\boldsymbol{x}^{(i)}\}_{i=1}^{10000}$，之后我们将用$\boldsymbol{x}^{(i)} \in \Re^{64}$代表第$i$个样本对应的向量。

**(2)**运行文件train.m中标有"Step 1" 的代码(请选择Matlab中运行模块的“运行节”，后面的很多步骤也是如此)，它将画出其中200幅图片(如图1 所示)。

提示：在5秒钟以内，函数sampleImages()就能够运行完毕。如果超过了30 秒，请调整你的算法，提高算法效率。

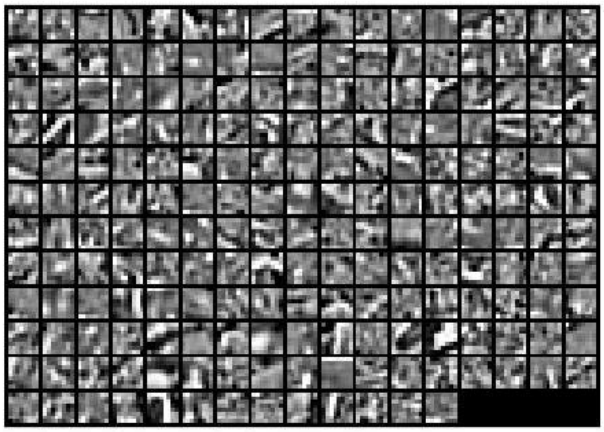

图 1: 样本生成结果实例

## 2.2 计算稀疏自动编码器参数梯度：将代码sparseAutoencoderCost.m补充完整

该神经网络共三层(如图2所示)，它们的结点数目为：输入层($n = 64$)，隐层($n_1 = 25$)，输出层($m = 64$)。稀疏自动编码器的目标函数$J(\boldsymbol{W}, \boldsymbol{w}_0)$ 定义如下：

$$J(\boldsymbol{W}, \boldsymbol{w_0}) = \left[ \frac{1}{N} \sum_{i=1}^{N} \left( \frac{1}{2} \left\| \hat{\boldsymbol{x}}(\boldsymbol{x}^{(i)}) - \boldsymbol{x}^{(i)} \right\|^2 \right) \right] + R + S, \tag{2}$$

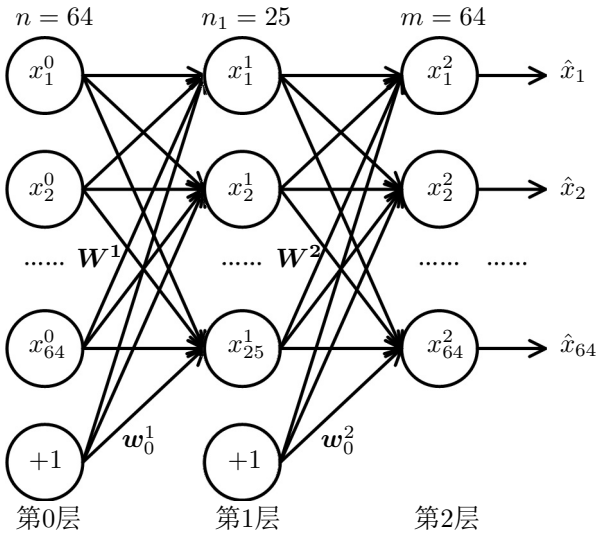


图 2: 自动编码器

其中第一项为我们上课提到的均方误差；第二项$R$ 是一个正则化项(也叫权重衰减项)，其目的是减小权重的幅度，防止过度拟合，具体可以参考文献[3,6]；第三项$S$是稀疏项(KL 距离)，目的是使得在任意样本输入网络后$\boldsymbol{x^1}$ 中仅有少数是接近1，其余均接近0，具体可以参考文献[4,7], 它们的定义如下：

$$R = \frac{\lambda}{2}\Big(\sum_{i=1}^{n}\sum_{j=1}^{n_1}\left(W_{ij}^1\right)^2 + \sum_{i=1}^{n_1}\sum_{j=1}^{m}\left(W_{ij}^2\right)^2\Big), \tag{3}$$

$$S = \beta\sum_{j=1}^{n_1}\left(\rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}\right) \tag{4}$$

其中, $\hat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N}\left[x_j^1(\boldsymbol{x}^{(i)})\right]$，而$\lambda = 0.0001, \beta = 3, \rho = 0.01$ 是定值。

请在反向传播算法的基础上进行一定的修改(例如，中间隐层的$\delta_i^1$需要修改为(5)式)，实现代码以求解相关参数导数(包括以下参数：权值矩阵$\boldsymbol{W}^1, \boldsymbol{W}^2$，截距向量$\boldsymbol{w_0^1}, \boldsymbol{w_0^2}$)。

$$\delta_i^1 = \left(\left(\sum_{j=1}^{n_1} W_{ij}^2\delta_j^2\right) + \beta\left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i}\right)\right)x_i^1(1-x_i^1) \tag{5}$$

提示：a)请尽量让你的程序矩阵-向量化，否则会很影响效率。b)激活函数的导数很容易计算，可显式直接求解出来。c)在目标函数中，你可以先忽略后两项，调试通过后再依次增加。

## 2.3 梯度检测：将代码computeNumericalGradient.m 补充完整

**(1)**完成梯度检验函数。

我们知道：

$$f'(x) = \lim_{\epsilon\to 0}\frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \tag{6}$$

所以，$\epsilon$很小时可以做如下近似(本例中取$\epsilon = 10^{-4}$ ):

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \tag{7}$$

由此我们可以通过该数值方法近似计算出目标函数$J$对各个参数的梯度，以对我们的解析解进行检验，判断程序中是否有误。请在computeNumericalGradient.m 中实现该数值计算过程。

**(2)**利用在checkNumericalGradient.m代码来测试你编写的computeNumericalGradient.m代码正确与否。

该文件中定义了一个简单的二元函数$h : \Re^2 \mapsto \Re, h(x) = x_1^2 + 3x_1x_2$，测试点为$x = (4, 10)^T$。如果你的程序是正确的，数值解和解析解将非常接近。

**(3)**使用computeNumericalGradient.m 来检测你在sparseAutoencoderCost.m 中实现的代码是否正确。关于细节，你可以看train.m 中的“Step 3”。

提示：在你调试错误的时候，你可以使用少量样本和少量隐藏节点，这样会使你节约更多时间。

### 2.4 训练稀疏自动编码器

运行train.m中的”Step 4”，等待训练完成。

提示：请注意，如果已经调试完毕，进入正式的训练阶段，请不要在每次迭代中都调用上一步的梯度检测，否则将显著降低计算速度。训练时间因计算机软硬件和实现形式的不同有差异，但一般不会超过到1小时，矩阵-向量形式实现的代码花费大约在10分钟左右。

### 2.5 可视化

在训练完稀疏自动编码器以后，使用display_network.m来可视化训练结果(详见train.m文件中的Step 5)。运行“print -djpeg weights.jpg”来保存文件“weights.jpg”。

提示：生成的结果与图3应该比较相似，如果出现类似于图4或其它情况可能是程序有问题，或者参数值有误。

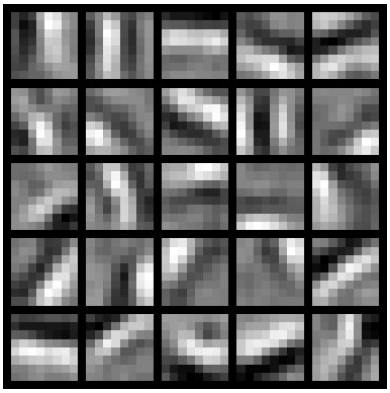

图 3: 稀疏自动编码器编码示例

## References


[1] 斯坦福大学的深度学习教程：http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

[2] 稀疏自动编码器，wiki: https://en.wikipedia.org/wiki/Autoencoder

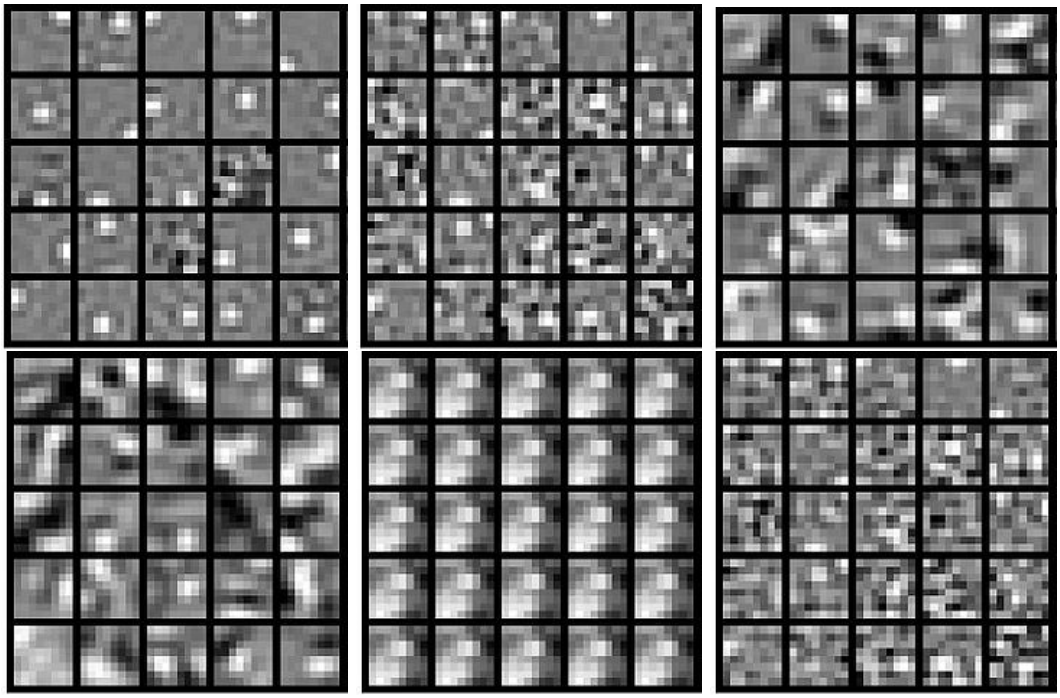

图 4: 有问题的示例


[3] 正则项，wiki：https://en.wikipedia.org/wiki/Norm_(mathematics)

[4] KL距离，wiki：https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

[5] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." science 313.5786 (2006): 504-507.

[6] 反向传播算法：http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm

[7] 稀疏自动编码器的定义：http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity