稀疏自编码神经网络

张晋

2018 年 1 月 25 日

1 Introduction

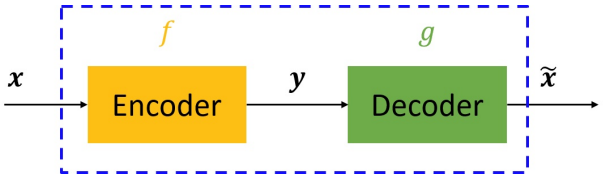
监督学习是人工智能最强大的工具之一，在自动压缩编码识别，语音识别，自动驾驶等方面都取得了重大成就，然而其学习过程仍然要求我们手动指定给算法的输入特征 x，如果我们给出了一个好的特征表示，那么这个监督学习算法就可以做得很好，但是手动标记特征是一个很慢的方法，因而其学习过程受到了极大限制。

因此，我们希望有一种算法能够自动学习比手动设计更好的特征表示，那就是稀疏自动编码器学习算法，这是从未标记数据自动学习特征的一种方法。

2 Autoencoders and sparsity

2.1 自编码器

自编码器可以理解为一个试图去还原其原始输入的系统。如下图所示。



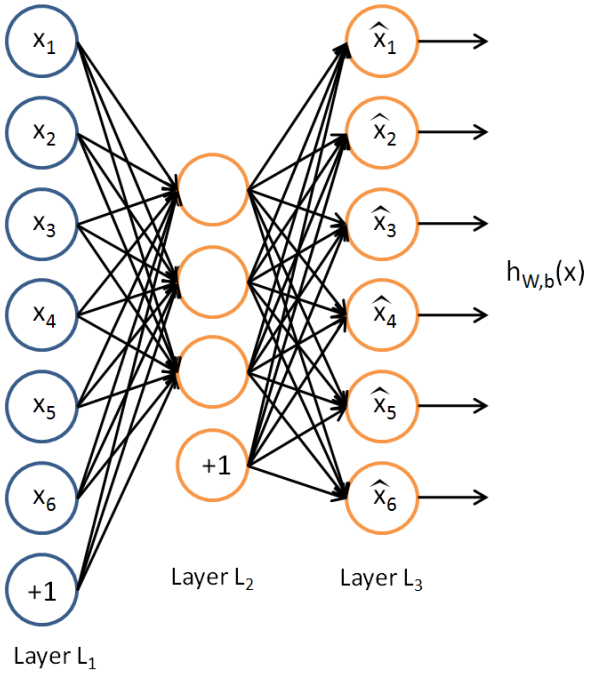
虚线蓝色框内就是一个自编码器模型，它由编码器 (Encoder) 和解码器 (Decoder) 两部分组成，编码器将输入信号 x 通过 f 变换成编码信号 $y = (x)$, 而解码器将编码信号 y 通过 g 转换成输出信号 $\tilde{x} = g(y) = g(f(x))$

而自编码器的目的是，让输出 $\tilde{x}$ 尽可能复现输入 x，然而如果 f 和 g 都是恒等映射，那不就恒有 $\tilde{x} = x$ 了吗, 因此，我们经常对中间信号 y（也叫作“编码”）做一定的约束，这样，系统往往能学出很有趣的编码变换 f 和编码 y

2.2 自编码神经网络

我们可以通过一个三层的神经网络来完成一个自编码器，它使用了反向传播算法来使让目标值等于输入值[1]，比如 $y^{(i)} = x^{(i)}$。

注意，其输出层的神经元个数与输入层是相等的，常用激活函数有：sigmoid、tanh、$\max(x, 0)$. 下图是一个自编码神经网络的示例。



自编码神经网络尝试学习一个 $h_{W,b}(x) \approx x$ 的函数，也就是恒等函数，从而使得输出 $\hat{x}$ 接近于输入 x 。

[1]注意自编码神经网络是一个无监督学习算法，因此训练集 $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)} | x^{(i)} \in \Re^n\}$ 是没有带标签的.

2.3 自编码神经网络的作用

有人可能会有疑惑，学习这样一个恒等函数有什么用呢？

这就要谈到压缩编码，一般来说，我们的输入 $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}$ 之间是有内在联系的，从信息论的角度来说，就是我们能找到一种更简洁的编码形式来表示它们。

在 Hinton 的论文 *Learning Intemal Representations by Error Propagation* 中，就举出了这样一个例子：构建一个神经网络将以下 N 位的单位向量映射到本身。

表 1: 编码问题

输入		输出
10000000	→	10000000
01000000	→	01000000
00100000	→	00010000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

从这个输入中，我们可以发现这是一串二进制数字，尽管位数很长，每个输入 $x^{(i)}$ 占了 8bit，但信息量却很低，用信息熵公式

$$\mathrm{H}(X) = -\sum_i \mathrm{P}(x_i) \log_2 \mathrm{P}(x_i)$$

可以算得每个输入 $x^{(i)}$ 蕴含的信息为 3/8bit，从而整个样本集蕴含的信息量为 3bit，那么从理论上来说，我们就可以把它的信息压缩，从 8 位压缩到 3 位。

对应到神经网络上，就是设置 8 个输入神经元，3 个隐藏神经元，从输入神经元到隐藏神经元，这是压缩编码的过程；再设置 8 个输出神经元，从隐藏神经元到输出神经元，对应的就是解码过程。

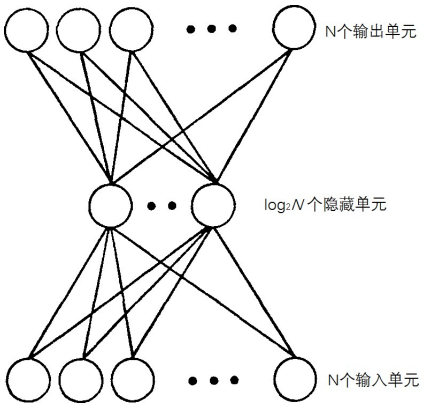


图 1: 解决该问题的神经网络的基本架构

现在我们来设置神经网络的参数：学习率 $\eta = 1$，动量学习率 $\mu = 0.8$，最大迭代次数 $maxcount = 100000$，终止精度 $eps = 10e - 4$[2]，minibatch 的 $size = 7$，运行`Encode_test.m` 程序，运行结果如下：

```
Cost =
   4.2151e-07

L(3).a =
0.9994   0.0001   0.0000   0.0003   0.0000   0.0000   0.0000   0.0001
0.0002   0.9997   0.0002   0.0000   0.0000   0.0001   0.0000   0.0000
0.0000   0.0002   0.9996   0.0001   0.0000   0.0000   0.0001   0.0000
0.0002   0.0000   0.0002   0.9994   0.0002   0.0000   0.0000   0.0000
0.0000   0.0000   0.0000   0.0002   0.9996   0.0000   0.0001   0.0002
0.0000   0.0002   0.0000   0.0000   0.0000   0.9994   0.0003   0.0003
0.0000   0.0000   0.0002   0.0000   0.0002   0.0000   0.9997   0.0000
0.0004   0.0000   0.0000   0.0000   0.0002   0.0001   0.0000   0.9994

L(2).a =
0.0013   0.0020   0.0023   0.2154   0.9972   0.9993   0.9235   0.7239
0.0028   0.9874   0.9979   0.0835   0.0034   0.9976   0.9912   0.0025
0.6127   0.9988   0.1309   0.0008   0.2528   0.9943   0.0017   0.9991

The corresponding binary:
  001      011      010      000      100      111      110      101
```

[2]程序中的损失函数定义为交叉熵函数，而最后输出的为均方误差值

可以发现，我们的神经网络完美地完成了自编码的过程，并且观察隐藏神经元的激活值，可以看出神经网络将每一个输入都匹配了不同的二进制编码，如果只保留前面两层的数据，那么我们就得到了一个压缩函数 $f(x)$.

在以上例子中，我们限制了隐藏神经元的个数**小于**输入神经元的个数，这就迫使自编码神经网络去学习输入数据的压缩表示，网络试图以更小的维度去描述原始数据而尽量不损失数据信息，因为它还必须要能从压缩表示中**重构**出原来的输入。

这种编码维度小于输入维度的自编码器称为**欠完备 (undercomplete)** 自编码器。由于被迫的降维，欠完备自编码器会自动习得训练样本最显著的特征（变化最大，信息量最多的维度）。[3]

2.4 稀疏自编码器

我们知道，编码维度小于输入维度时，自编码器可以训练数据中最显著的特征，但是，当隐藏编码的维数与输入相等，甚至大于输入的**过完备 (overcomplete)** 情况下,可能会出现 f, g 都是恒定函数的情况,或者 $f(x) = x + 1, g(x) = x - 1$ 的情况，这样即使自编码器能使输出 $\hat{x}$ 接近于输入 x ，也不能学到任何有用的信息。

这时，我们需要对自编码器加入一些限制条件来发现输入数据中的结构，比如**稀疏性限制**，这将迫使系统不得不在大量维度中找出真正重要的维度。

稀疏性可以被简单地解释如下：当神经元的输出接近于 1 的时候我们认为它被激活，而输出接近于 0 的时候认为它被抑制，那么使得神经元大部分的时间都是被抑制的限制则被称作稀疏性限制。[4]

我们用 $a_j^{(2)}(x)$ 来表示在给定输入样本为 x 情况下，自编码神经网络隐藏神经元 j 的激活值，让

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x^{(i)}) \right]$$

表示隐藏神经元 j 的平均激活值（在训练集上取平均）。

[3] 当编码器是线性，误差函数为二次函数时，欠完备自编码器会学习出与 PCA 相同的生成子空间，即主元子空间；而当编码器非线性时，它能学到更强大的 PCA 非线性推广。

[4] 这里我们假设的神经元的激活函数是 sigmoid 函数。如果使用 tanh 作为激活函数的话，当神经元输出为-1 的时候，我们认为神经元是被抑制的.

我们加入稀疏性限制: 令 $\hat{\rho}_j = \rho$。其中，ρ 是**稀疏性参数**，我们理想中 ρ 最好是趋于 0 的，但这在现实中不可能达到（就算达到了，也等于什么都没学到）。所以我们退而求其次，让 ρ 是一个接近于 0 的较小的值（比如 $\rho = 0.05$）。

为了实现这一限制，我们在目标函数中加入一个罚函数：

$$\sum_{j=1}^{s_2} \mathrm{KL}(\rho||\hat{\rho}_j) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

这里，s_2 是隐藏层中隐藏神经元的数量，而索引 j 依次代表隐藏层中的每一个神经元。

现在，我们的总体代价函数可以表示为

$$J_{\mathrm{sparse}}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} \mathrm{KL}(\rho||\hat{\rho}_j),$$

其中 $J(W,b)$ 如之前所定义，而 β 控制稀疏性惩罚因子的权重。$\hat{\rho}_j$ 项则也（间接地）取决于 W,b ，因为它是隐藏神经元 j 的平均激活值，而隐藏层神经元的激活值取决于 W,b 。

2.5 计算

在给代价函数增添了罚函数后，我们需要对原 BP 算法进行改动，我们记 $B = \beta \sum_{j=1}^{s_2} \mathrm{KL}(\rho||\hat{\rho}_j)$， $J_{\mathrm{sparse}}(W,b) = J + B$.

$$\begin{aligned}
\delta_i^{(3)} &= \frac{\partial J_{\mathrm{sparse}}(W,b)}{\partial z_i^{(3)}} \\
&= \frac{\partial J(W,b)}{\partial z_i^{(3)}} + \frac{\partial B}{\partial z_i^{(3)}} \\
&= \frac{\partial J(W,b)}{\partial z_i^{(3)}} \\
\delta_i^{(2)} &= \frac{\partial J_{\mathrm{sparse}}(W,b)}{\partial z_i^{(2)}} \\
&= \frac{\partial J(W,b)}{\partial z_i^{(2)}} + \frac{\partial B}{\partial z_i^{(2)}} \\
&= \frac{\partial J(W,b)}{\partial z_i^{(2)}} + \frac{\partial B}{\partial \hat{\rho}_i} \frac{\partial \hat{\rho}_i}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \\
&= \frac{\partial J(W,b)}{\partial z_i^{(2)}} + \beta \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) f'(z_i^{(2)})
\end{aligned}$$

在原 BP 算法中，我们已经计算了

$$\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) f'(z_i^{(2)}),$$

现在我们只需将其换成

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) + \beta \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{(2)}).$$

需要注意的是：在计算梯度之前，我们需要知道 $\hat{\rho}_i$ 。所以在反向传播之前，我们需要对所有的训练样本计算一遍前向传播，从而获取平均激活值。

如果训练样本可以小到被整个存到内存之中，我们可以就可以直接使用事先计算好的激活值来对所有的训练样本进行后向传播的计算。

如果训练样本的数据量太大，无法全部存入内存，我们就必须一个个的将训练样本 $x^{(i)}$ 前向传播得到 $a^{(2)}(x^{(i)})$，然后将获得的结果累积起来并计算平均激活值 $\hat{\rho} = \frac{1}{m} \sum_{i=1}^{m} a^{(2)}(x^{(i)})$ 。在完成平均激活值 $\hat{\rho}_i$ 的计算之后，我们才能重新对每一个训练样本做一次前向传播然后进行后向传播。这相当于对每一个训练样本需要计算两次前向传播，所以在计算上的效率会稍低一些。

3 Visualization

3.1 实验 1

实验 1 是基于 Stanford 大学 UFLDL Tutorial 的Exercise:Sparse Autoencoder，数据集及部分程序[5]来源主页上的sparseae_exercise.zip

文件说明

- **IMAGES.mat：** 作为训练集的图片库
- **train.m：** 实验的主程序
- **sampleIMAGES.m：**随机选择照片生成训练集
- **display_network.m：**用来将数字矩阵转化为图像输出

[5]由于是课后编程练习，所以只给出了程序的大致框架和可供调用的 L-BGFS 程序，重要部分都需要自己完成

- **computeNumericalGradient.m**：用数值方法计算梯度
- **checkNumericalGradient.m**：并排输出反向传播计算的梯度和数值方法计算的梯度[6]
- **initializeParameters.m**：对神经网络的权值和偏置初始化
- **sparseAutoencoderCost.m**：计算代价函数[7]和梯度
- **lbfgs.m**：可供调用的拟牛顿法程序

训练集

数据集 IMAGES 内包含 10 张像素为 512×512 的图片，我们从中随机选取一张图片，然后在图片中随机框出 10000 张 8×8 的小图片，并根据每张小图片生成一个 64 位的列向量，然后将这些列向量排成一个矩阵作为训练集，其中每一个列向量都是神经网络的输入。

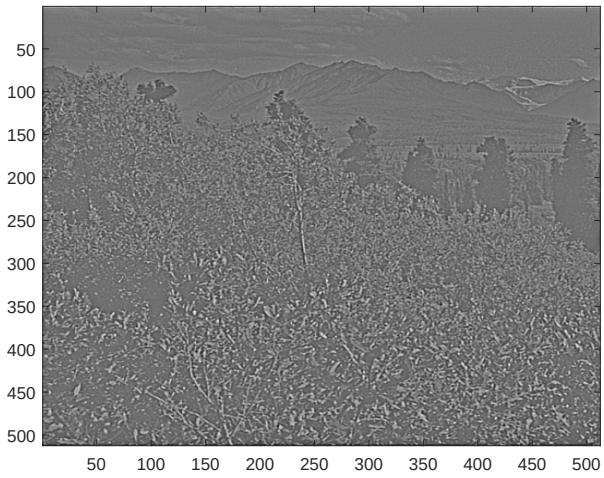


图 2: 数据集 IMAGES 中的一张图片

[6] 先在小样本检验程序是否有梯度计算错误，验证完程序正确后不需要再调用

[7] 原代价函数为均方误差函数，添加了正则化项和罚函数

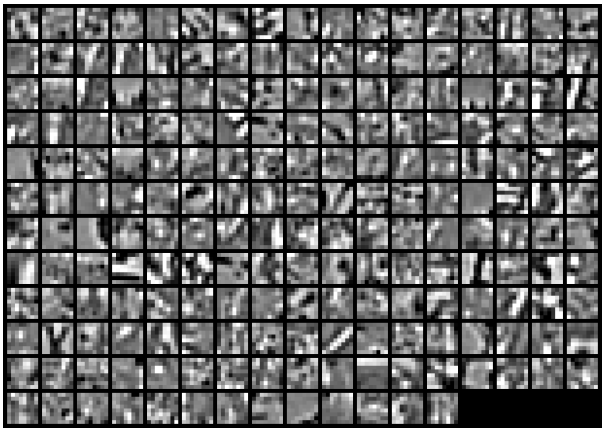


图 3: 随机选出的 200 张小图片

可以看到，这些小格子都是从原图片中分割出来的，从我们看来，它们杂乱而无规律，但是实际上它们是这张图片的局部缩影，每一张都包含着图片的部分特征。我们将它们作为输入，让神经网络从中学习到特征。

神经网络参数

我们选取神经网络的结构为 $[64, 25, 64]$，稀疏性参数 $\rho = 0.01$，控制稀疏性惩罚因子的权重 $\beta = 3$，除此之外，我们还加上正则化项，其权重衰减系数 $\lambda = 0.0001$，

训练神经网络

运行程序 `train.m` 就可以开始训练神经网络了。程序先会显示分割出的训练集图片，不要关闭图片窗口。然后程序会调用封装好的`lbfgs.m` 程序进行搜索确定下降方向和步长，并显示显示迭代情况，最后自动输出并保持权值矩阵的图像。

权值可视化

在训练完稀疏自编码器后，我们需要知道这个神经网络到底学到了什么，我们知道，神经网络学习到的东西都存在权值上，而作为自编码器，最重要的权值为 $W^{(1)}$，这个权值矩阵是 25×64 的，我们将每一个行向量转化成 8×8 的矩阵，然后作为图像输出，一共 25 张图像并排输出如下：

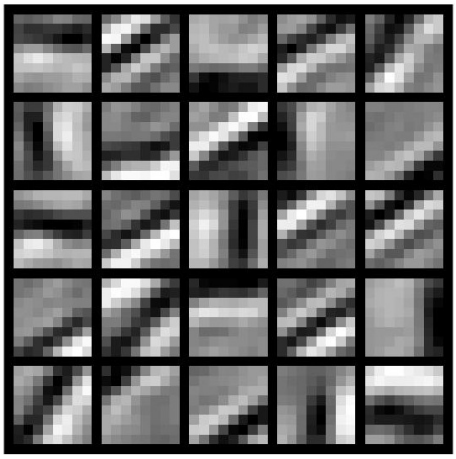


图 4: 权值矩阵的可视化

我们可以看到的是，比起之前的输入，这个图像似乎呈现出了一丝规律性，但这丝规律性又说不清道不明，这个权值矩阵的图像意味着什么呢？

可能的解释

我们可以从生物学的角度来理解，就像大脑受到某一刺激时某些神经元会兴奋一样，对于输入数据的不同特征，神经网络中会有相应的神经元被激活，如果对于任意特征，都有大量的神经元激活响应，那么神经网络就显得杂乱而低效，所以我们希望单个神经元只对某种特征兴奋。

也就是说，对于由一张图片分割出来的大量小图片，我们希望单个神经元对所有小图片的响应的均值接近 0，这就是稀疏性限制，

但是，我们还是需要神经元对于某些图片做出响应，所以我们只需把稀疏性参数 ρ 调小即可。同时，过小的 ρ 会使得神经元被过分压抑，出现这种情况:W, b 都是负的，且绝对值极大，这就使得对于任意的输入，神经元都不会被激活，为了避免这种情况，所以我们需要加入正则化项，防止极端情况的出现。

那么，对于什么样的输入 $\boldsymbol{x}$，神经元 i 的激活值最大呢？我们记 W_i 是由 $W_{i1}^{(2)}, W_{i1}^{(2)}, \cdots, W_{i25}^{(2)}$ 排成的行向量 $[W_{i1}^{(2)}, \cdots, W_{i25}^{(2)}]$.

那么

$$a_i^{(2)}(x) = \sigma\left(W_i \boldsymbol{x} + b_i^{(1)}\right)$$

给定范数约束 $\|\boldsymbol{x}\|^2 = \sum_{i=1}^{64} x_i^2 \leq 1$ ，由于 $\sigma(\cdot)$ 是单调递增的，$W_i, b_i^{(1)}$ 也是固定值，在这个条件下，要使 $W_i\boldsymbol{x}$ 最大，显然

$$\boldsymbol{x} = \frac{W_i}{\|W_i\|^2}$$

也就是，当输入为权值矩阵 $W^{(2)}$ 第 i 行的行向量时，隐藏层第 i 个神经元的激活值达到最大，或者说，对于这种模式，神经元的响应最剧烈。因此当我们把权值矩阵 $W^{(2)}$ 可视化时，我们看到的就是隐层一个个神经元学习到的模式——不同的隐藏单元学会了在图像的不同位置和方向进行边缘检测。

在这之后，我将抽取的小图片的大小改为 10×10，将神经网络的结构改成 $[100, 100, 100]$，最终得到了与原论文类似的图像.

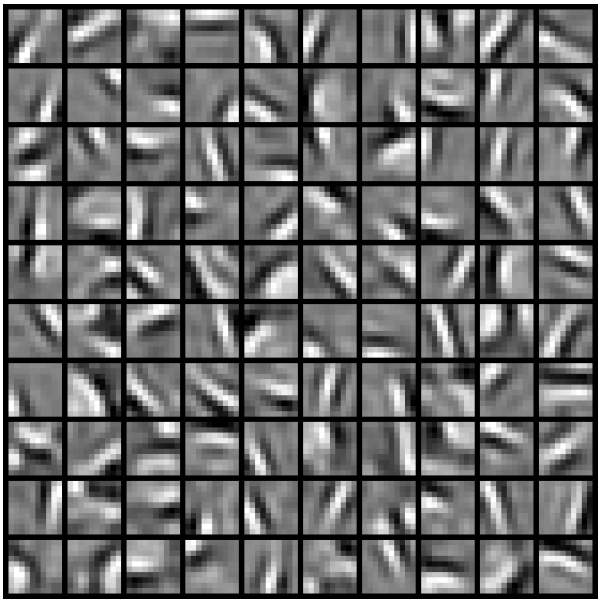


图 5: 新程序学到的特征

3.2 实验 2

实验 2 是也是基于UFLDL Tutorial的Exercise:Vectorization.

从以下两个链接分别下载 MNIST 手写数字库的训练集图像、训练集分类标签和将这两个文件导入 MATLAB 的程序：

- MNIST Dataset (Training Images)
- MNIST Dataset (Training Labels)
- Support functions for loading MNIST in Matlab

文件说明

- **SAE_MNIST.m**： 实验的主程序
- **display_network.m**：用来将数字矩阵转化为图像输出
- **loadMNISTImages.m**：将 MNISTI 数据集的图像导入 MATLAB
- **sparseAutoencoderCost.m**：计算代价函数和梯度
- **lbfgs.m**：可供调用的拟牛顿法程序

由于 MNIST Dataset 中的图像是 28×28 的，所以需要在原程序上进行参数上的改动，根据 Exercise 上的说明，将隐藏层神经元的个数设置为 196 个，那么神经网络的结构为 [784,196,784]，另外，设置稀疏性参数 $\rho = 0.1$，控制稀疏性惩罚因子的权重 $\beta = 3$，权重衰减系数 $\lambda = 0.003$，由于计算量过大，所以只选取 5000 张图片作为训练集，之后启动主程序`SAE_MNIST.m`开始训练神经网络，在 400 次迭代后输出权值矩阵图像。

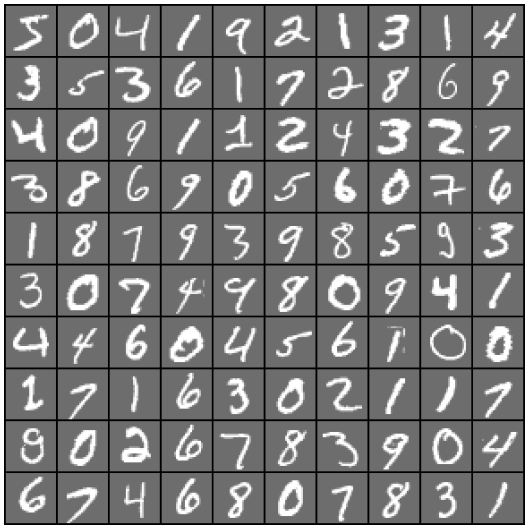


图 6: MNISTI 手写数字

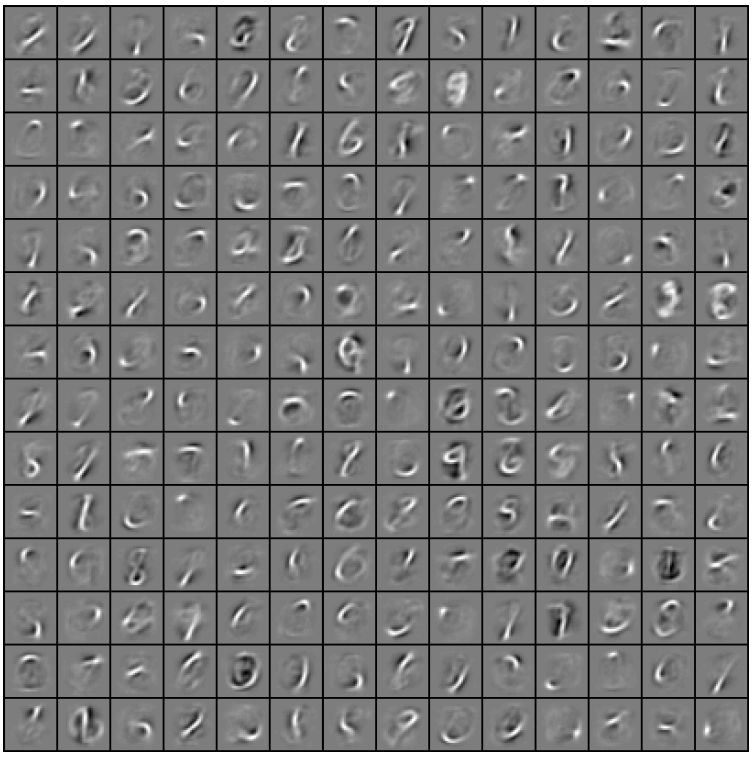


图 7: 学习到的模式

观察图像，总体上来说，神经网络还是学习到了手写数字的特征——有的神经元学到了点、横、竖、撇、捺等，但也能看出，有的神经元学习得并不彻底，隐约还能看到原来数字的轮廓。

总结: 这些特征对物体识别等计算机视觉任务是十分有效的。若将其用于其他输入域（如音频），该算法也可学到对这些输入域有用的表示或特征。