深度学习技术与应用（4）

Deep Learning: Techniques and Applications (4)

Ge Li

Peking University

Table of contents

关于 *Loss Function* 的补充

条件概率的最大似然估计

The maximum likelihood estimator can readily be generalized to the case where our goal is to estimate a conditional probability $P(y|x;\theta)$.
If X represents all our inputs and Y all our observed targets, then the conditional maximum likelihood estimator is:

$$\theta_{ML} = \operatorname*{argmax}_{\theta} P(Y|X;\boldsymbol{\theta}). \tag{1}$$

If the examples are assumed to be i.i.d. (independent identically distributed), then this can be decomposed into:

$$\theta_{ML} = \operatorname*{argmax}_{\theta} \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\boldsymbol{\theta}). \tag{2}$$

基于上述原则推导 MSE 的合理性

最小均方误差函数 (Mean Squared Error, MSE)：对于一组有 m 个样本的训练集，代价函数 MSE 定义如下:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

- 若，$h_\theta(x_i) = \theta^T x_i$，则为 Linear Regression.
- 此处，为不失一般性，仅假设 $h_\theta(x_i)$ 为神经网络输出层的输出值.

基于上述原则推导 MSE 的合理性

假设目标值与输入变量之间存在如下关系：

$$y_i = h_\theta(x_i;\theta) + \epsilon_i$$

由前文推导，可合理假设：$\epsilon_i \sim \mathcal{N}(0,\sigma^2)$，则其概率密度函数为：

$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right)$$

代入上式，并由误差的定义，可以推知：

$$p(y_i|x_i;\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y_i - h_\theta(x_i))^2}{2\sigma^2}\right)$$

基于上述原则推导 MSE 的合理性

根据前文结论，求取代价函数的基本原理是：

在已知 m 个 (x_i, y_i) 的前提下，若对代价函数进行最小化，则能够使"按照上述概率模型所得到的最大似然值"最大化。

于是，这里，我们首先写出"按照上述概率模型所得到的最大似然值"：

$$
\begin{aligned}
L(\theta) &= \prod_{i=1}^{m} p(y_i|x_i;\theta) \\
&= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y_i - h_\theta(x_i))^2}{2\sigma^2}\right)
\end{aligned}
$$

基于上述原则推导 MSE 的合理性

对上式进行最大化求取，等价于：

$$
\begin{aligned}
\log\left(L(\theta)\right) &= log\prod_{i=1}^{m} p(y_i|x_i;\theta) \\
&= \sum_{i=1}^{m}\log\frac{1}{\sqrt{2\pi}\sigma}exp\left(-\frac{(y_i-h_\theta(x_i))^2}{2\sigma^2}\right) \\
&= m\log\frac{1}{\sqrt{2\pi}\sigma}-\frac{1}{\sigma^2}\cdot\frac{1}{2}\sum_{i=1}^{m}(y_i-h_\theta(x_i))^2
\end{aligned}
$$

要通过调整 θ，使上式最大化，只需要考虑使最后的二次项最小化即可，即最小化：

$$
\underset{\theta}{\operatorname{argmin}}\frac{1}{2}\sum_{i=1}^{m}(y_i-h_\theta(x_i))^2 \quad \text{即：} \underset{\theta}{\operatorname{argmin}}\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x_i)-y_i)^2
$$

基于伯努利分布的 Loss Function

若可假设网络输出满足如下分布：

$$p(y=1|x;\theta)=h_\theta(x)$$
$$p(y=0|x;\theta)=1-h_\theta(x)$$

上式可写为：

$$p(y|x;\theta)=(h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

基于伯努利分布的 Loss Function

则，似然函数为：

$$
\begin{aligned}
L(\theta) &= p(Y|X;\theta) \\
&= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\
&= \prod_{i=1}^{m} \left((h_\theta(x))^{y^{(i)}} (1 - h_\theta(x))^{1-y^{(i)}} \right)
\end{aligned}
$$

进行 $\log$ 处理，得到需要最大化的 Loss Function 为：

$$
\begin{aligned}
l(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} \left(\log h(x^{(i)}) + (1 - y^{(i)}) \log \left(1 - h(x^{(i)})\right) \right)
\end{aligned}
$$

Logistic Regression

基于多项分布的 Loss Function

若可假设网络输出满足如下分布：

$$
\begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)} \begin{bmatrix} f(x^{(i)};\theta_1) \\ f(x^{(i)};\theta_2) \\ \vdots \\ f(x^{(i)};\theta_k) \end{bmatrix}
$$

定义函数：

$$
1\{\text{表达式为真}\} = 1
$$

基于多项分布的 Loss Function

则，似然函数为：

$$\begin{aligned} L(\theta) &= p(Y|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m}\left(\sum_{j=1}^{k} 1\{y^{(i)}=j\}\frac{f(x^{(i)};\theta_j)}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)}\right) \end{aligned}$$

进行 $\log$ 处理，得到需要最大化的 Loss Function 为：

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m}\left(\sum_{j=1}^{k}\left(1\{y^{(i)}=j\}\right)\log\frac{f(x^{(i)};\theta_j)}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)}\right) \end{aligned}$$

基于多项分布的 Loss Function

等同于最小化：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{k}\left(1\{y^{(i)}=j\}\right)\log\frac{f(x^{(i)};\theta_j)}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)}\right)$$

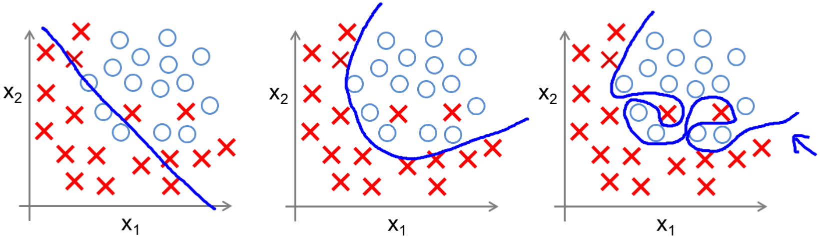
为便于计算，通常取 $f(x^{(i)};\theta_j) = exp(\theta_j^T x^{(i)})$，得：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{k}\left(1\{y^{(i)}=j\}\right)\log\frac{exp(\theta_j^T x^{(i)})}{\sum_{j=1}^{k} exp(\theta_j^T x^{(i)})}\right)$$

SoftMax

关于正则化方法

Regularization



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g= sigmoid function)

"Underfit"

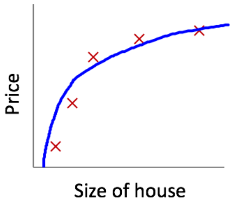
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

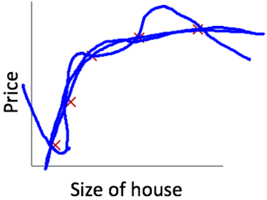
"Overfit"

From: Andrew Ng, Machine Learning Course.

Regularization



$\theta_0 + \theta_1 x + \theta_2 x^2$



$\underline{\theta_0 + \theta_1 x + \theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$

Suppose we penalize and make θ_3, θ_4 really small.

$$\rightarrow \quad \min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

$\theta_3 \approx 0 \qquad \theta_4 \approx 0$

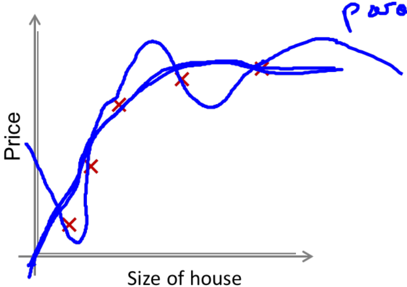
From: Andrew Ng, Machine Learning Course.

Regularization

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

regularization parameter

$$\min_\theta J(\theta)$$



From: Andrew Ng, Machine Learning Course.

L2 Regularization

设未经正则化的 Loss Function 为：$J(w,b)$；
正则化项为：$\Omega(\theta)=\frac{1}{2}\|w\|_2^2$；
正则化参数为 λ；
则，正则化后的 Loss Function 为：

$$J(w,b)+\frac{\lambda}{2}\|w\|_2^2=J(w,b)+\frac{\lambda}{2}w^Tw$$

在反向传播的过程中：

$$w=w-\alpha\frac{\partial J(w,b)}{\partial w} \qquad b=b-\alpha\frac{\partial J(w,b)}{\partial b}$$

得：

$$w=w-\alpha\frac{\partial(J(w,b)+\frac{\lambda}{2}w^Tw)}{\partial w} \qquad b=b-\alpha\frac{\partial(J(w,b)+\frac{\lambda}{2}w^Tw)}{\partial b}$$

L2 Regularization

可见：

$$w = w - \alpha\left(\frac{\partial(J(w,b)}{\partial w} + \lambda w\right)$$

$$\text{即}：w = (1-\alpha\lambda)w - \alpha\frac{\partial(J(w,b)}{\partial w}$$

$$\text{而}：b = b - \alpha\frac{\partial J(w,b)}{\partial b}$$

- 可见，正则化方法对于 b 的更新没有影响；
- 而对于 w 则起到了以 $1-\alpha\lambda$ 幅度减小 w 的作用，这被称为 Weight Decay.

L1 Regularization

对于 Loss Function：$J(w,b)$；
正则化项为：$\Omega(\theta) = |w|_1 = \sum_i |w_i|$；
正则化参数为 λ；
则，正则化后的 Loss Function 为：

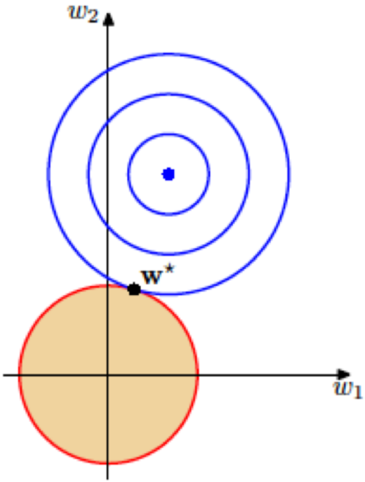
$$J(w,b) + \lambda |w|_1 = J(w,b) + \lambda \sum_i |w_i|$$

得：

$$w = w - \alpha \frac{\partial (J(w,b) + \lambda \sum_i |w_i|)}{\partial w}$$
$$w = w - \alpha \lambda sign(w) - \alpha \frac{\partial (J(w,b)}{\partial w}$$

其中，sign(w) 表示 w 的符号，当 w 为正时，更新后 w 变小；当 w 为负时，更新后 w 变大，可见其结果仍然是让 w 靠近 0；

Regularization



From: Christopher M. Bishop, Pattern Recognition and Machine Learning

关于学习率的优化

梯度下降过程中的权重更新：

$$\theta = \theta - \alpha \nabla_\theta J(\theta)$$

Momentum

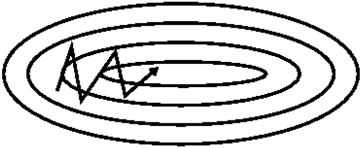
- Momentum based Gradient Descent:

$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta \left(\frac{1}{m} \sum_t^m J(x^{(i)}, y^{(i)}; \theta) \right)$$

$$\theta = \theta - v_t$$

- 在更新模型参数时，对于那些当前梯度方向与上一次梯度方向相同的参数，进行加强，即在这些方向上的参数更新更快了；
- 对于那些当前梯度方向与上一次梯度方向不同的参数，进行削减，即在这些方向的参数更新上减慢了。

一般而已，动量项参数 $\gamma < 0.9$



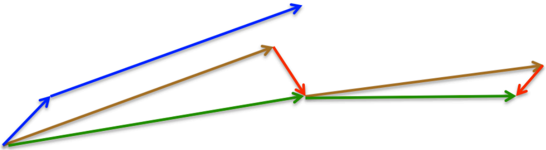
NAG-Nesterov Accelerated Gradient

- NAG:

$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta \left(\frac{1}{m} \sum_t^m J(\theta - \gamma v_{t-1}) \right)$$

$$\theta = \theta - v_t$$

- Computing $\theta - \gamma v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update). This is a rough idea where our parameters are going to be.
- We can now effectively look ahead by calculating the gradient not with regard to our current parameters θ but with regard to the approximate future position of our parameters.



Adagrad

- 在上述模型中，每个模型参数 θ_i 使用相同的学习速率 α，而 Adagrad 在每一个更新步骤中对于每一个模型参数 θ_i 使用不同的学习速率 α_i;
- 设第 t 次更新步骤中，目标函数的参数 θ_i 梯度为 $g_{t,i}$，即：

$$g_{t,i} = \nabla_\theta J(\theta_i)$$

- 则，传统的 SGD 的更新方程表示为：

$$\theta_{t+1,i} = \theta_{t,i} - \alpha \cdot g_{t,i}$$

- 而，Adagrad 对每一个参数使用不同的学习率，则其更新方程变为：

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

其中，$G_t \in \mathcal{R}^{d\times d}$ 是一个对角矩阵，其中第 i 行的对角元素 e_{ii} 为过去到当前第 i 个参数 θ_i 的梯度的平方和，ϵ 是一个平滑参数，为了使得分母不为 0（如可取 $\epsilon = 1e-8$）

Adagrad

写成矩阵形式：

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

- Adagrad 主要优势在于它能够为每个参数自适应不同的学习速率，而一般的人工都是设定为 0.01。其缺点在于需要计算参数的整个梯度序列的平方和，并且学习速率趋势是不断衰减最终达到一个非常小的值，开始很大，最后很小。

Adadelta

- Adadelta 提出的目的也是为了避免 Adagrad 对学习速率的调整过于"鲁莽"的问题；
- 同时，为了避免计算整个梯度序列的平方和，Adadelta 采用了"窗口"技术，即，仅对固定窗口内的 w 个梯度序列进行计算；
- 当前的梯度平方的平均值（$E[g^2]_t$）仅依赖于前一个时刻的平均值和当前的梯度；

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2$$

- 可以把 γ 设为一个类似于动量的值，如 0.9 附近.

Adadelta

- 下面给出 Adadelta 的表达式：从 $\Delta\theta_t$ 的表达式开始:

$$\Delta\theta_t = -\alpha \cdot g_{t,i}$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

对比 Adagrad 的公式：

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t$$

用 $E[g_t^2]$ 替换 G_t:

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{E[g_t^2] + \epsilon}} \odot g_t$$

Adadelta

- 可见，分母为梯度的均方根（Root Meam Square），简短表示为：$RMS[g]_t$ 得：

$$\Delta\theta_t = -\frac{\alpha}{RMS[g]_t} \cdot g_t$$

- 还注意到，梯度的更新中 α 并不平缓，做以下替换:

因为：$E[\Delta\theta^2]_t = \gamma E[\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$

- 于是，得到：

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Adadelta

- 又因为 t 时刻的 $RMS[\Delta\theta]_t$ 并不知道，于是用 t 时刻之前的参数更新的 RMS 来代替，即：用 $RMS[\Delta\theta]_{t-1}$ 代替 α，最终得到 Adadelta 的更新规则：

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

RMSprop

RMSprop 由 Geoff Hinton 在他的 Coursera 课程中提出。
RMSprop 与 Adadelta 几乎在同一时间提出，只是可以看做 Adadelta 的简化版本，对比如下：

- Adadelta:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

- RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1 g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g_t^2] + \epsilon}} \cdot g_t$$

- 可见，RMSprop 方法也是用"衰减的梯度均方根误差"去除学习率。

Adam-Adaptive Moment Estimation

- Adam（自适应的矩估计）也是一种不同参数自适应不同学习速率方法，与 Adadelta 与 RMSprop 区别在于，它计算历史梯度衰减方式不同，不使用历史平方衰减，其衰减方式类似动量：

$$
m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t
$$
$$
v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
$$

- m_t 与 v_t 分别是梯度的一阶矩和二阶矩的估计值，初始为 0 向量；

Adam-Adaptive Moment Estimation

- 然而，它们通常被偏置化为趋向于 0 的向量，特别是当衰减因子(衰减率)β_1, β_2 接近于 1 时；
- 为了改进这个问题，可以改进上式中的偏置项：利用经过偏置修正的一阶和二阶矩估计来计算 m_t 与 v_t：

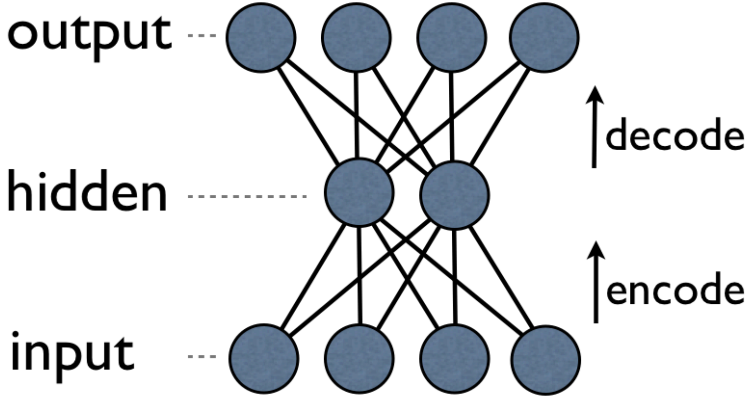
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- 类似 Adadelta 与 RMSprop 方法，可以得到 Adam 方法的更新规则：

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

- Adam 提出者建议 $\beta_1 = 0.9, \beta_2 = 0.9999, \epsilon = 10^{-8}$. 实验证实，Adam 方法较其他方法有更好的应用效果。



Regularized AutoEncoder

$$\mathcal{J}_{AE}(\theta) = \sum_{\mathrm{x} \in S} L(\mathrm{x}, g(f(\mathrm{x}))),$$

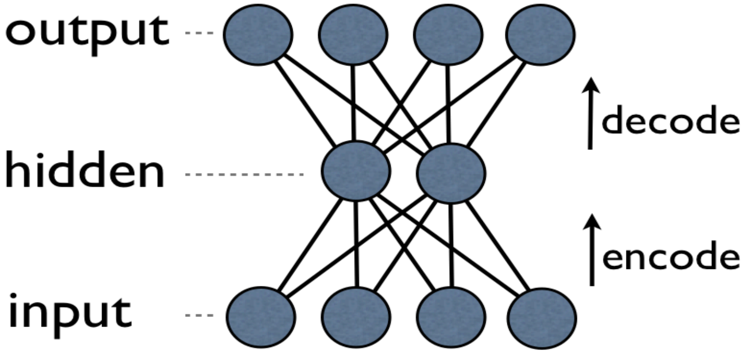
$$\mathcal{J}_{AE}(\theta) = \frac{1}{N} \sum_{\mathrm{x} \in S} L(\mathrm{x}, g(f(\mathrm{x}))),$$

- 增加权重衰减项的 损失函数：

$$\mathcal{J}_{AE+wd}(\theta) = \sum_{\mathrm{x} \in S} L(\mathrm{x}, g(f(\mathrm{x}))) + \lambda \sum_{i,j} W_{i,j}^2,$$

权重衰减项

Sparse AutoEncoder



n 输入输出层神经元个数
m 隐藏层神经元个数
x, y, h 各层神经元上的向量
p , q 各层神经元的偏置
w 输入层与隐藏层之间的权值
$\tilde{w}$ 隐藏层与输出层之间的权值

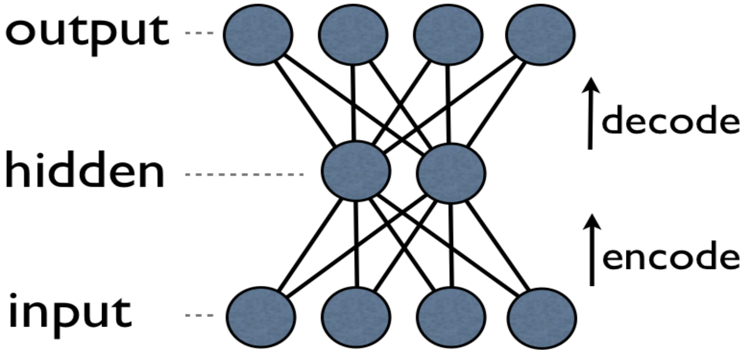
- 隐藏层上第 j 号神经元在训练集 $S=\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ 上的平均激活度.

$$\widehat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} h_j(\mathbf{x}^{(i)}),$$ 令 $$\widehat{\rho}_j = \rho,\ j = 1, 2, \cdots, m,$$

其中ρ为一个很小的数，例如$\rho < 0.05$

目的：当某隐藏层神经元的平均激活度超过ρ时，对其进行惩罚处理

Sparse AutoEncoder



$\boldsymbol{n}$ 输入输出层神经元个数
m 隐藏层神经元个数
x, y, h 各层神经元上的向量
p , q 各层神经元的偏置
w 输入层与隐藏层之间的权值
$\tilde{w}$ 隐藏层与输出层之间的权值

- 整体 损失函数 为：

$$\mathcal{J}_{AE+sp}(\theta)=\sum_{\mathrm{x}\in S} L(\mathrm{x}, g(f(\mathrm{x}))) + \beta\sum_{j=1}^{m} KL(\rho\|\hat{\rho}_j),$$

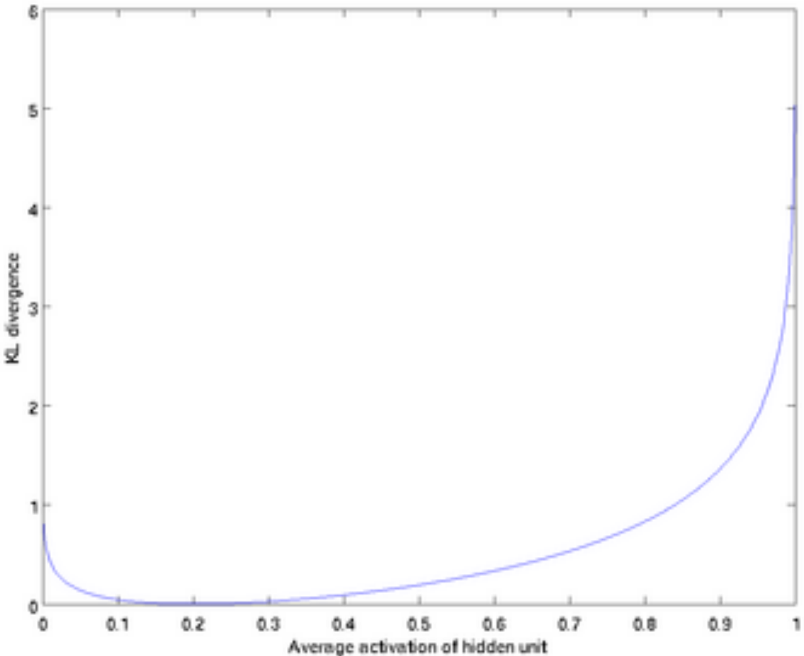
- 结合正则化的 损失函数 为：

$$\mathcal{J}_{AE+wd+sp}(\theta)=\sum_{\mathrm{x}\in S} L(\mathrm{x}, g(f(\mathrm{x}))) + \lambda\sum_{i,j} W_{i,j}^2 + \beta\sum_{j=1}^{m} KL(\rho\|\hat{\rho}_j).$$

如何进行惩罚处理？

- 引入 相对熵值 函数

$$KL(\rho||\hat{\rho}_j) = \rho * \log\frac{\rho}{\hat{\rho}_j} + (1-\rho) * \log\frac{1-\rho}{1-\hat{\rho}_j}.$$



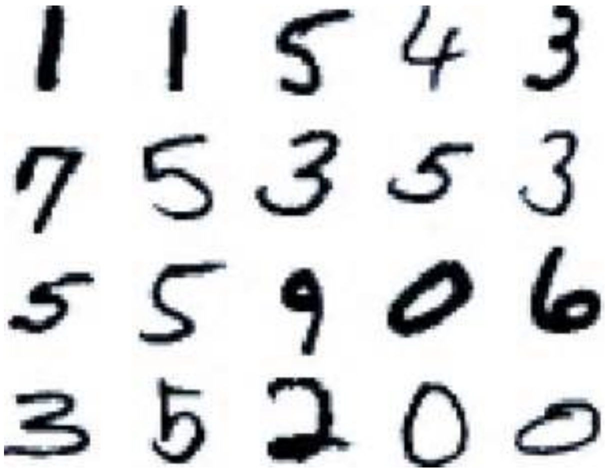
特点：

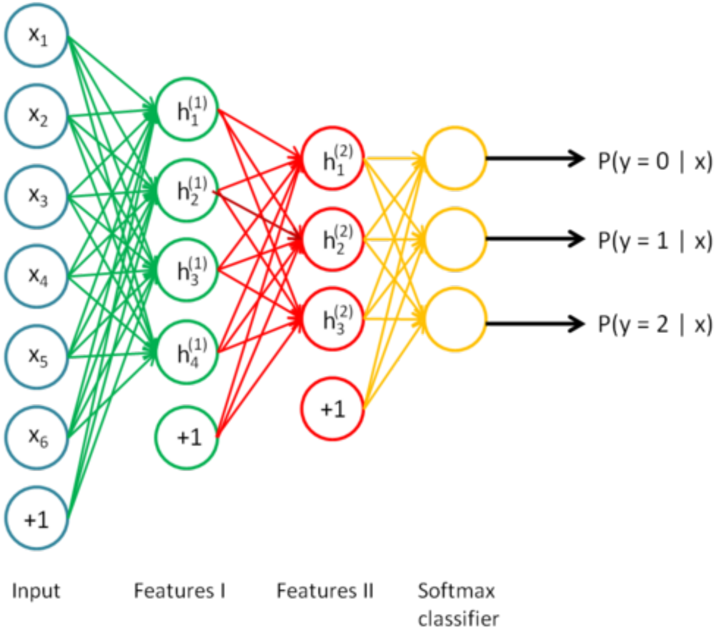
- 在 $\hat{\rho}_j = \rho$ 时达到最小值0
- 当 $\hat{\rho}_j$ 靠近0或者1的时候，相对熵则变得非常大；

Handwritten Digits Classifiers

■ 最终目标

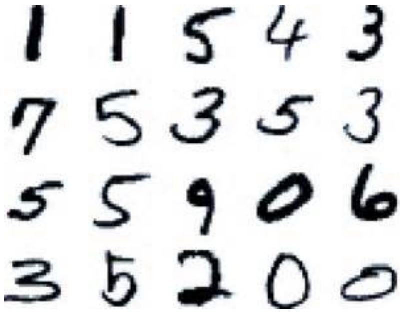
- ◆ 训练一个包含两个隐藏层的自编码器，用于MNIST手写数字识别。

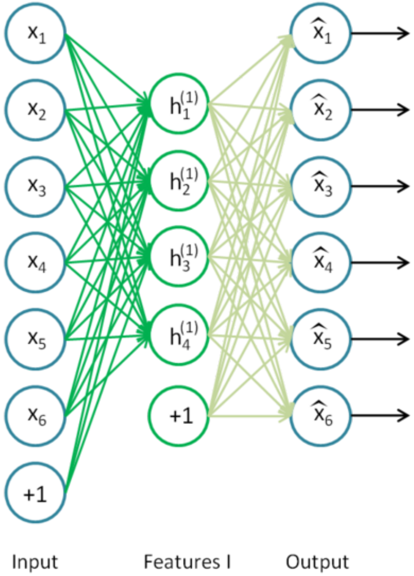




Handwritten Digits Classifiers

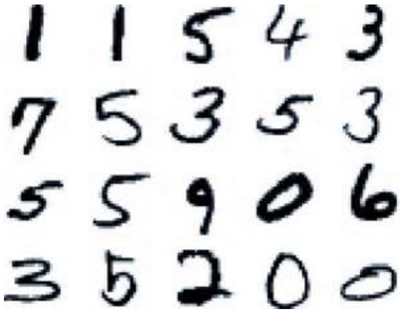
- Step 1：用原始输入$x^{(k)}$训练第一个自编码器，学习得到原始输入的一阶特征表示$h^{(1)(k)}$;
  - Step 1.5：把原始数据输入到Step1训练好的稀疏自编码器中，对于每一个原始输入$x^{(k)}$，都可以得到它对应的一阶特征表示$h^{(1)(k)}$

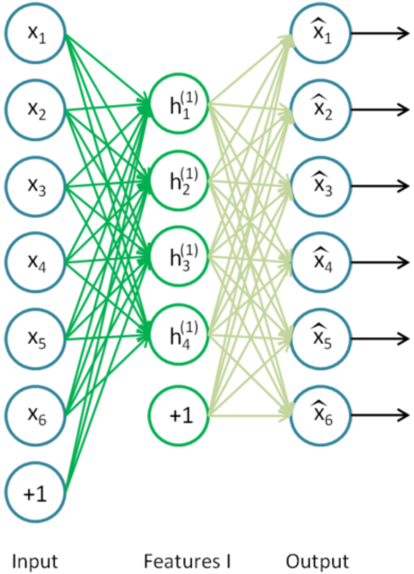


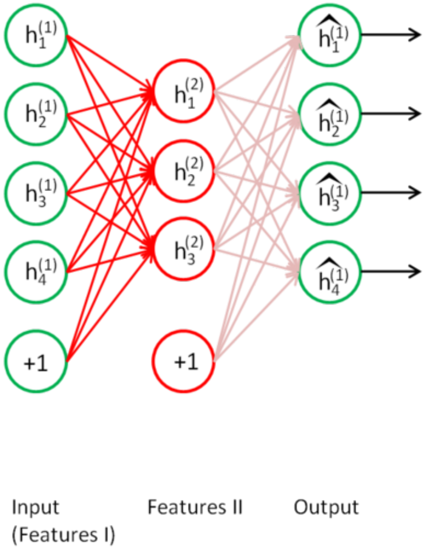


Handwritten Digits Classifiers

- Step 2：然后再用这些一阶特征作为另一个稀疏自编码器的输入，使用它们来学习二阶特征$h^{(2)(k)}$。
  - Step 2.5：把一阶特征输入到Step2训练好的第二层稀疏自编码器中，得到每个$h^{(1)(k)}$对应的二阶特征激活值$h^{(2)(k)}$。

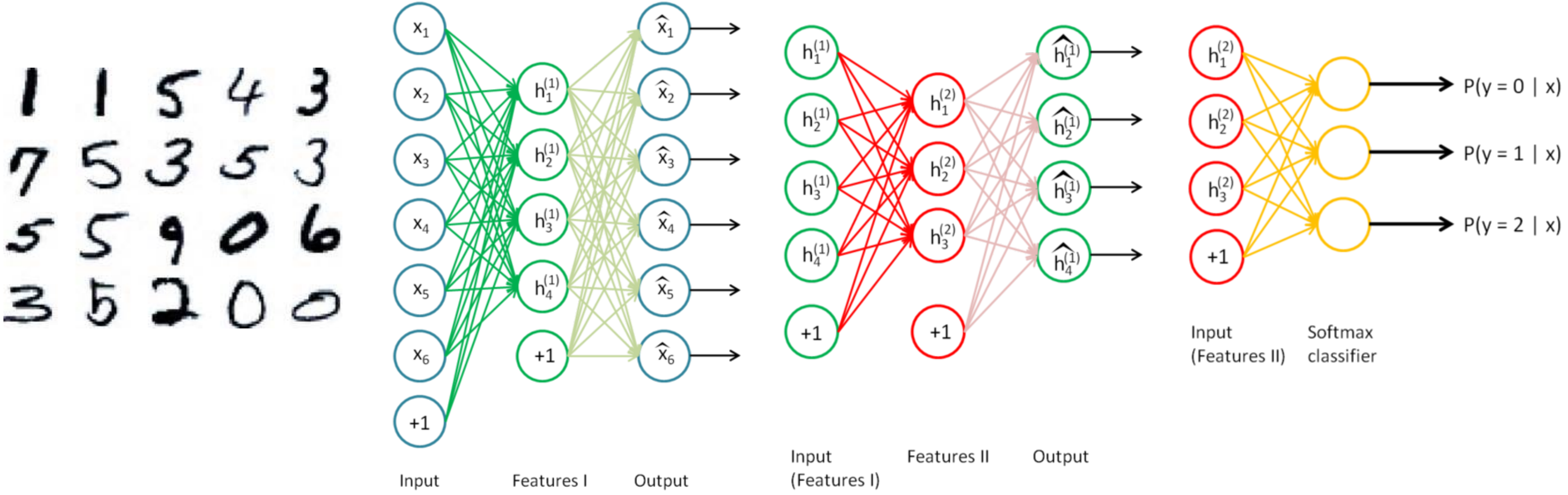






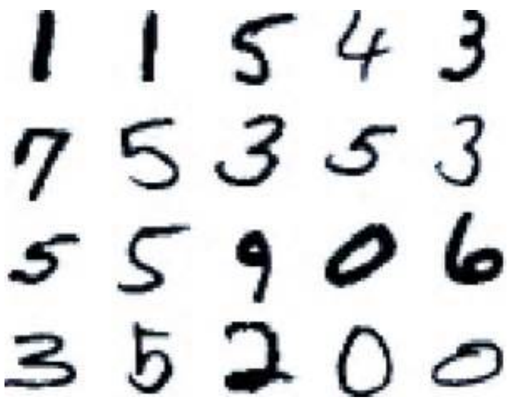
Handwritten Digits Classifiers

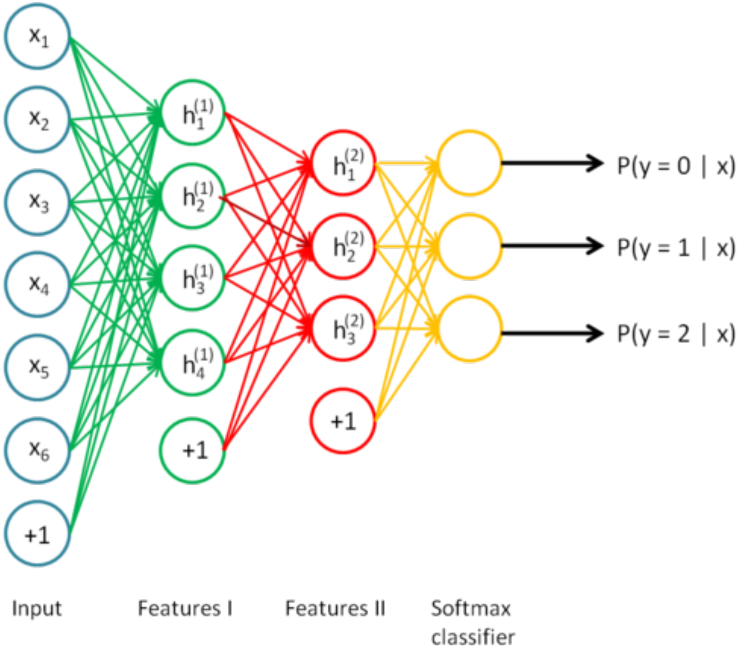
- Step 3：把这些二阶特征作为softmax分类器的输入，训练得到一个能将二阶特征映射到数字标签的模型。



Handwritten Digits Classifiers

- Step 4：将上述三层结合起来构建一个包含两个隐藏层和一个最终softmax分类器层的栈式自编码网络，对MNIST数字进行分类。





Thanks.