#DL

- *The brain is not a container needs to be filled with knowledge, but a torch needs to be lighted*
  现在，让我们开始深度学习的旅程！

ANN 是什么?

ANN 是由独立的单元通过某种方式连接而形成的图，是一种自下而上地形成的人工智能——也就是说，它先模拟了人脑的结构，再实现人脑的功能
三个重要元素：神经元、结构、学习算法
两个重要函数：组合与激活函数
两种重要的网络结构：带有反馈和不带反馈
一些人脑特有的学习方式：Hebbrian, Competitive
基本的学习方式：监督、非监督、强化

神经元的模拟

对于每个神经元，我们设 y 为其输出，向量 x 为其输入，那么一个神经元被按照如下的方式模拟：

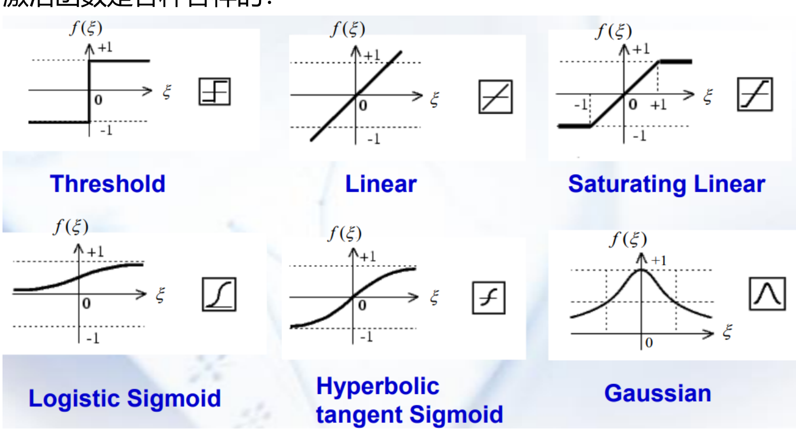
$$y = f(g(x))$$

其中，g 被称为整合 (Combination)函数，而 f 被称为激活 (Activation)函数
一般来说，

$$g(x) = \sum_i w_i x_i - \theta_i$$

其中，θ_i 被称为偏置项

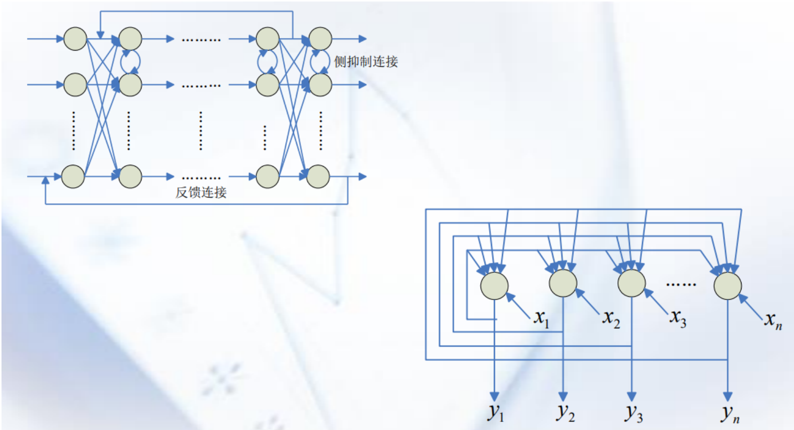
激活函数是各种各样的：



现在很火的 ReLu 函数是 Saturating 函数的一个特例

网络结构优化：NAS

两种网络的显著区别：不带反馈、带有反馈



如图是两种带有反馈的网络，它们分别带有层内的传递（侧抑制）和层间的传递。

一些常见的学习策略

- 正常的策略：最小化误差，梯度下降
- Hebbrian Learning : 一个神经的两端的神经元同时激活时，神经的权值将上升
- 竞争学习：

多层感知机（前馈网络）

单层感知机

最初的多层感知机使用一种 0-1 激活函数，一旦整合函数的结果超过阈值，那么就输出为 1，否则反之。最初的学习方式是

$$\omega_{ij} \leftarrow \omega_{ij} + \eta x_i(d_i - y_i)$$

这实际是一个平方损失求导，进行梯度下降。
实际上，这种最初的前馈感知机就是一种函数的表示，但是，它只能表示线性的函数！
（解决办法可能是：多条直线拼接，或者使用非线性的函数，实际上最有效的是增加模型的复杂度）
（我们增加复杂度的时候一般是增加深度，这是因为深度增加带来的复杂度开销小于增加宽度）

- 实际上，3 层的网络可以表达所有连续函数，而 4 层网络可以拟合所有函数（包括离散）

BP 神经网络

注意：需要推导反向传播

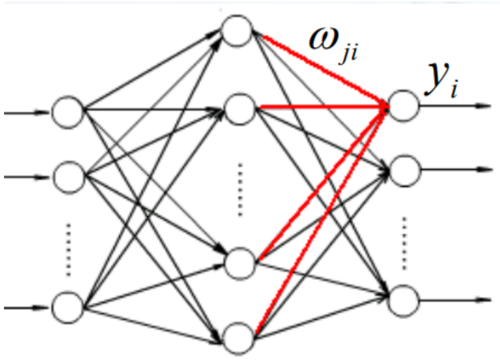
- 在 BP 神经网络上，激活函数被换成了 Sigmoid 函数（这样的性质利于反向传播的建立）
  我们这里仅仅考虑最简单的神经网络，让其误差为 MSE 损失：

$$e(\omega) = \frac{1}{2}\sum_i (d_i - y_i)^2$$

  其中，d_i 为真实值，而 y_i 为预测值。这个函数的好处是：它很有可能是凸函数。
  我们的优化方法通常是基于梯度的：

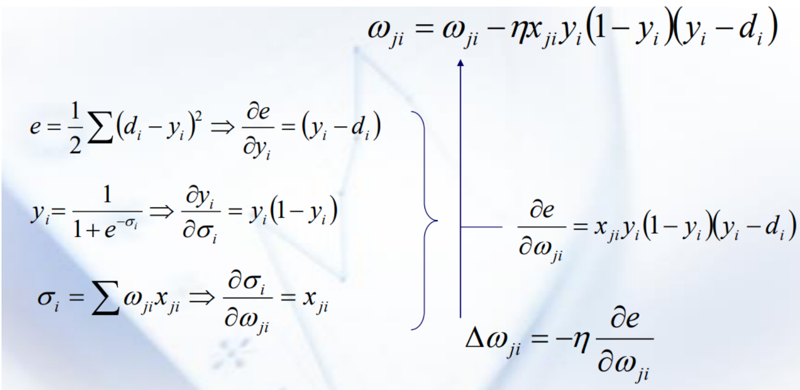
$$\omega \leftarrow \omega - \eta \frac{\partial e}{\partial \omega}$$

接下来，我们推导反向传播算法：

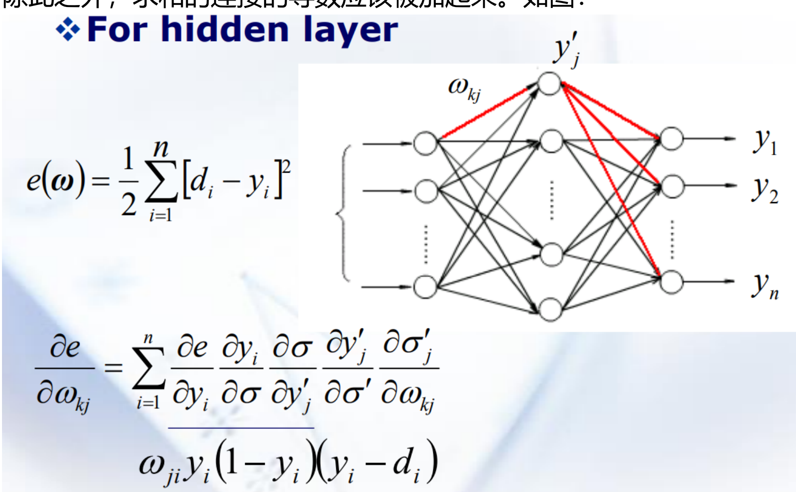


$$\frac{\partial e}{\partial \omega_{ji}} = \frac{\partial e}{\partial y_i}\frac{\partial y_i}{\partial \sigma_i}\frac{\partial \sigma_i}{\partial \omega_{ji}}$$

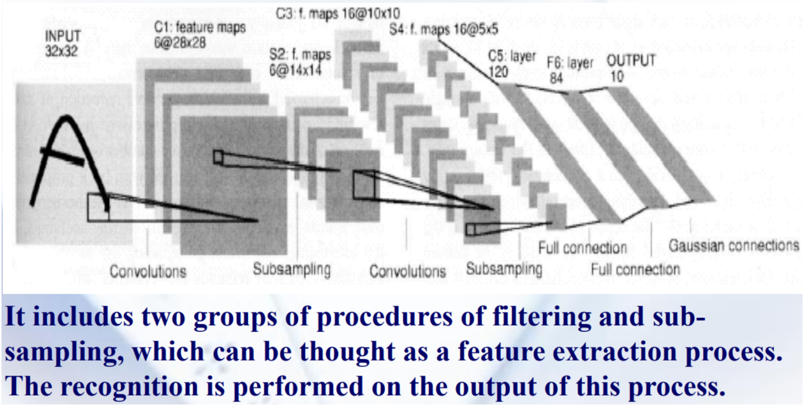
其中，σ_i 是激活函数。因此，我们将三个导数加起来



对于多层的情况：第一层的激活函数对着第二层的输出求导；第二层的输出再对着激活函数求导，激活函数在对着第三层的输出求导；，，；最后一层的激活函数对着权重求导。
除此之外，求和的连接的导数应该被加起来。如图：



Example 利用卷积神经网络从局部上获取特征



卷积提取到的是局部特征，这相当于一个滤波的运算
通道数：由于我们使用了多个卷积核，因此在一个点上会计算出许多特征，特征的数目就是通道数
池化（下采样）：每一次池化其实都是在增大感受野，使得我们可以将卷积提取到的特征进行组合；除此之外，可以降低数据的规模，降低模型复杂度
后期使用了 ReLu 函数

从这里，深度学习开始了。这里主要是结构上的革命，学习方法上仍然是后向传播。

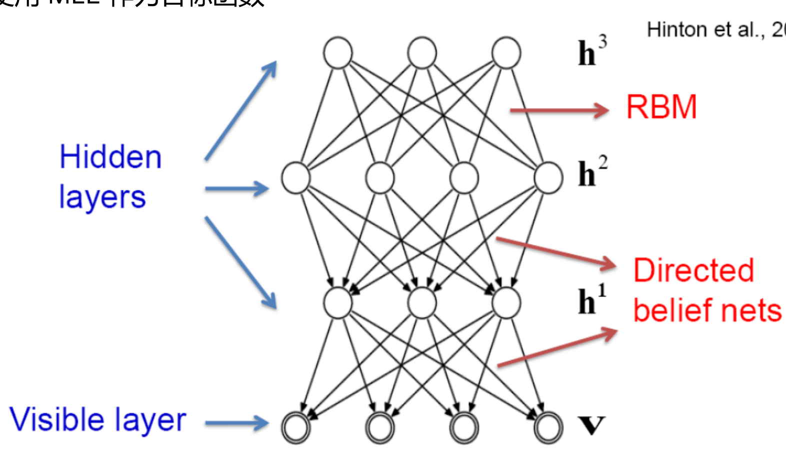
为什么传统的 BP 在深度增加后仍有问题?

- 梯度消失
- 训练太慢
  - Exacerbated since top couple layers can usually learn any task "pretty well" and thus the error to earlier layers drops quickly as the top layers "mostly" solve the task– lower layers never get the opportunity to use their capacity to improve results, they just do a random feature map
- 需要更多的数据
  - Can we use unsupervised/semi-supervised approaches to take advantage of the unlabeled data
- 有更多的局部极小值

为了解决这些问题，我们试图使得网络的前几层得到充分的训练；并且使得未标注的数据也可以被用于训练。

两个提升：深度信念网络和自动编码器

深度信念网络

- 使用了概率生成模型
- 使用 MLE 作为目标函数
- 

  这个网络类似于贝叶斯信念网络，那么，写出联合概率：

$$P(v, h_1, h_2, \cdots, h_l) = P(v|h_1) \prod P(h_i|h_{i+1}) P(h_{l-1}, h_l)$$

在同一层内，显然有：（类似于朴素贝叶斯）

$$P(h_i|h_{i+1}) = \prod_{j=1}^{n^i} P(h_j^i|h_{i+1})$$

使用如下性质的激活函数：

$$P(h_i^j = 1|h_{i+1}) = \frac{1}{1 + \exp(-b_i^j - \sum w_i^{kj} h_{i+1}^k)}$$

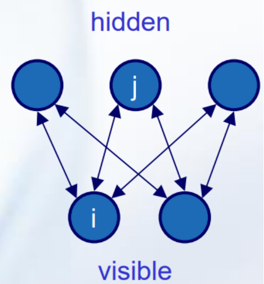
An RBM has a single layer of hidden units which are not connected to each other and have undirected, symmetrical connections to a layer of visible units.

$$P(v,h) = \frac{1}{z} e^{\mathbf{h'Wv + b'v + c'h}}$$

$$energy(v,h) = -\mathbf{h'Wv} - \mathbf{b'v} - \mathbf{c'h}$$

$$P(v_i = 1|\mathbf{h}) = sigm\left(b_i + \sum_k W_{ki} h_k\right)$$

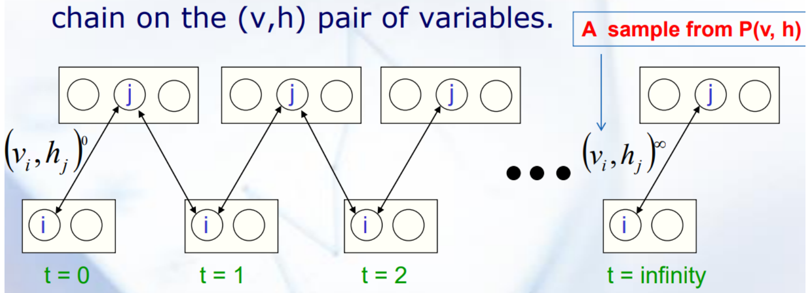
$$Q(h_j = 1|\mathbf{v}) = sigm\left(c_j + \sum_k W_{jk} v_k\right)$$



学习过程

想要获知网络输出了什么，你需要进行采样

- To obtain an estimator of the gradient on the log-likelihood of an RBM, we consider a Gibbs Markov chain on the (v,h) pair of variables.



Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

通过这样的方式将 $P(v, h)$ 采样出来。DBN 旨在学习输入数据的概率分布，以便生成与输入数据类似的新数据。

$$\log P(\mathbf{v}_0) = \log \sum_{\mathbf{h}} P(\mathbf{v}_0, \mathbf{h}) = \log \sum_{\mathbf{h}} e^{-energy(\mathbf{v}_0, \mathbf{h})} - \log \sum_{\mathbf{v},\mathbf{h}} e^{-energy(\mathbf{v}, \mathbf{h})}$$

$$\frac{\partial \log P(\mathbf{v}_0)}{\partial \boldsymbol{\theta}} = -\sum_{\mathbf{h}_0} Q(\mathbf{h}_0|\mathbf{v}_0) \frac{\partial energy(\mathbf{v}_0, \mathbf{h}_0)}{\partial \boldsymbol{\theta}} + \sum_{\mathbf{v}_k, \mathbf{h}_k} P(\mathbf{v}_k, \mathbf{h}_k) \frac{\partial energy(\mathbf{v}_k, \mathbf{h}_k)}{\partial \boldsymbol{\theta}}$$

$$k \to \infty$$

DBN 是使用一种贪心策略，自下而上地逐层叠加 RBM（受限玻尔兹曼机）的。对于每一层玻尔兹曼机，你都试图使其拟合真实的数据分布。每层玻尔兹曼机包含一层可见节点和一层不可见节点，每个节点的数值只能取 0 或 1，在每一轮

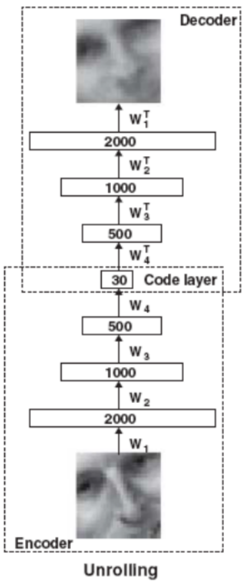
训练中，RBM 经历一次正向传播（调整隐变量的节点数值）和反向传播（调整连边权重）的过程。采样是指根据隐变量的值来依照概率生成可见变量的过程，因为真实变量的概率分布是难以计算的，因此只能通过采样的方式看到网络现在是什么样子。

在完成自下而上的训练后，DBN 会采样生成最上层的数据，再进行自上而下的传播，使得上层对下层有影响。

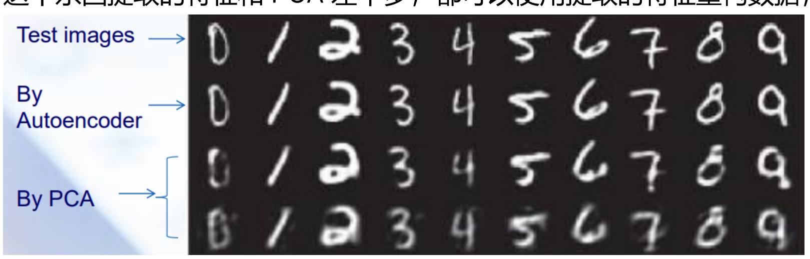
泛泛地来说，DBN 还是很好的，因为这是对学习方式的变革，只不过现在用的不太多

自动编码器

AutoEncoder 将 DBN 网络视作一种"编码器"，在第一层上，真实可见的变量被编码成一系列的隐变量；再向上一层，隐变量被进一步编码。如果你的编码是正确的，那么如果将"编码器"倒扣过来，图像可以被还原，如图：



AutoEncoder 试图计算编码-解码后的图像与原始图像的关系。并通过这个距离来反向传播进行训练。
这个东西提取的特征和 PCA 差不多，都可以使用提取的特征重构数据，这是很多东西 (CNN)得不到的



循环神经网络 (Recurrent Network)

Hopfield Network

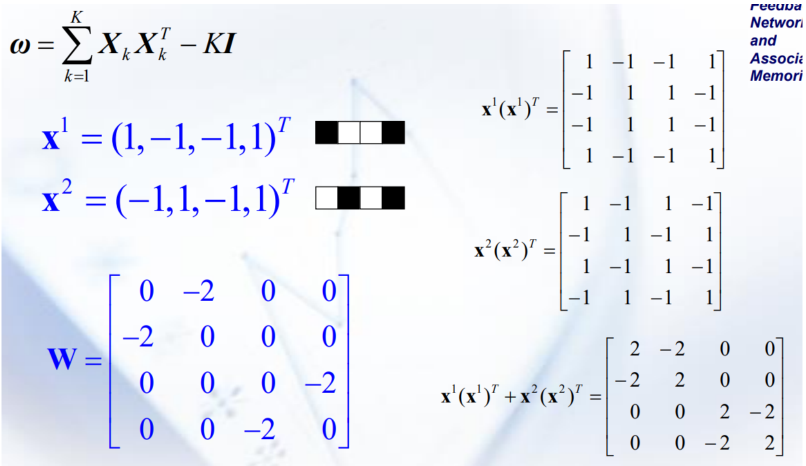
这个网络的特征是：每个神经元的输出直接回到了每个神经元的输入。（这与目前的 RNN 的区别是，RNN 将隐变量回送并且加入新的输入）
我们试图使得网络参数走向稳定，那么 Hopfield 使用 Lyapunov 稳定性来衡量网络的稳定性。简单地，假设每个神经元只能取 1 或 0，w_{ij} 为两个神经元之间的连边权重，那么可以证明，如果

$$w_{ij} = w_{ji} \quad w_{ii} = 0$$

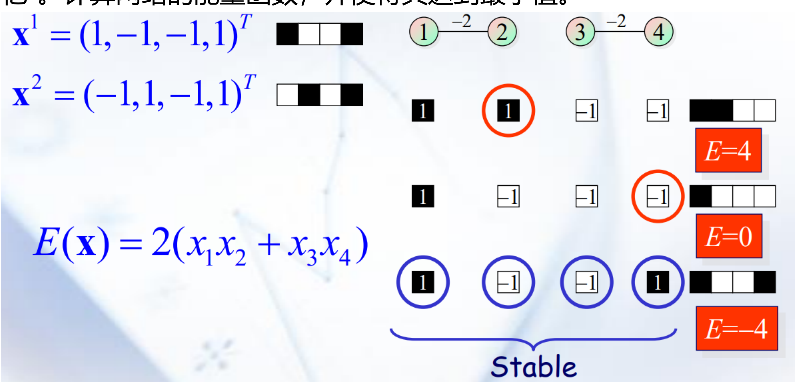
网络就是稳定的。
这种稳定的网络有两种应用：

Associative Memories（联想记忆）

我们已经提到过，Lyapunov 稳定性中使用一个能量函数来刻画系统的稳定性，因此，Hopfield 网络试图找到这个能量函数的最小值。比如，存储图像时，每个点对应一个神经元。那么你可以计算一种符合要求的网络结构：



网络的权重中就存入了它记住的模式，现在，我们要做的是输入一个信息，使得网络进行"联想"，从而激发其之前的"记忆"。计算网络的能量函数，并使得其达到最小值。

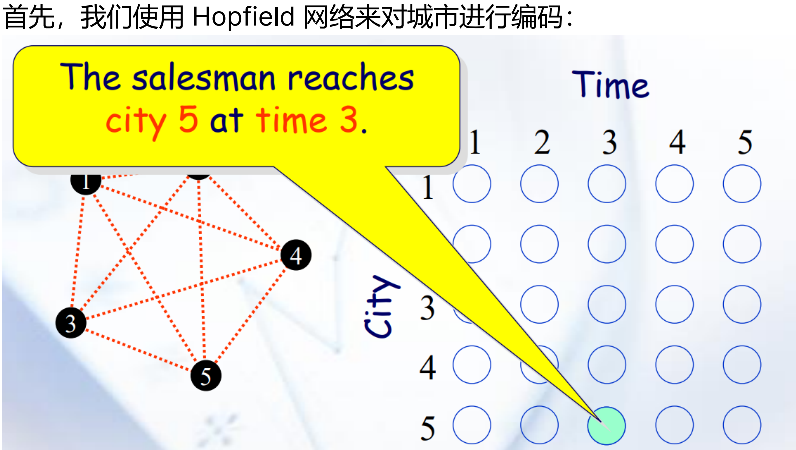


不同的输入将使得网络联想到不同的"记忆"，不同的变化方式也会将网络引向不同的"记忆"。一个网络能储存多少个模式，是由网络能量函数的局部极小值的数目决定的。

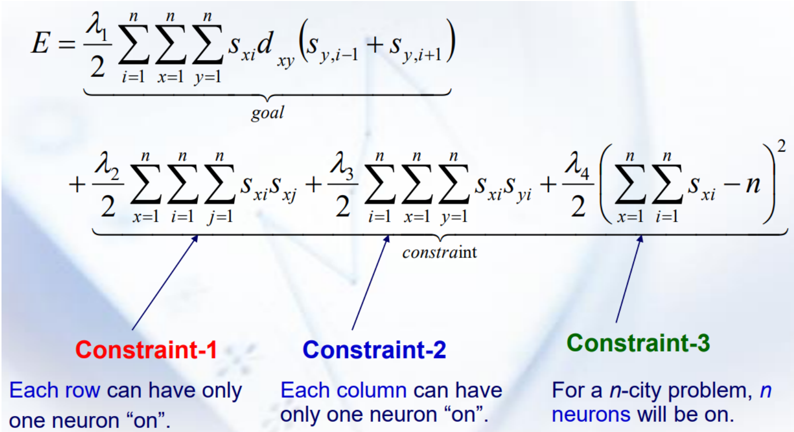
Combinational Optimization（组合优化）

例子：TSP 问题

首先，我们使用 Hopfield 网络来对城市进行编码：



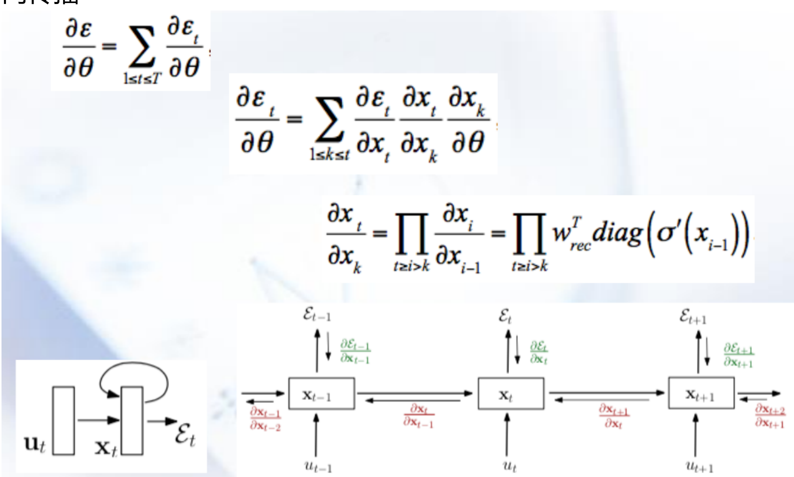
在第 i 行，第 j 列的神经元激活意味着将在第 j 个时间步访问第 i 个城市。根据合情合理的约束条件，你可以建模一个目标函数（使用了乘子法）：



此后，你需要将这个目标函数转换成 Hopfield 网络的能量函数的形式，然后就可以迭代求解了。

Long-Short Term Memory Network

这个网络是参数时变的网络，每轮预测中，一个隐变量被回送给神经元，和新的输入一起给出一个新的输出。这种网络的基本训练方式是 Back propagation Through Time (BPTT)，神经网络在几个时间步上被展开，梯度沿着神经网络反向传播



这样做的问题是，会导致梯度的爆炸 (Exploding)和消失 (Vanishing)

LSTM 通过某些强制性的设置，使得导数中的某些项强制为 1，从而避免这些现象。此外，LSTM 使用了遗忘门来控制网络如何记忆过去的信息，使用输出们控制神经元到底要把哪些东西吐出去给下一个神经元

Meta-Learning：Learning to Learn !