第四讲

简介：

这一讲开始介绍 Generative model。从如何让一个有循环的网络记忆 pattern 开始，讨论了 Hopfield Network 这一最基本的模型。随后，为了记忆更多的 pattern 讨论了具有 hidden neuron 的网络并引入 probabilistic 的 Boltzmann Machine。最后，将前面的模型进行总结，对一般的 Energy-based model 进行论述。

目录：

Hopfield Network: 引入

考虑一个含有圈的 Neural network, 其中每一个 neuron 的数值为$y_i \in \{\pm 1\}$。在每一轮迭代，它们根据如下的方式演化：

$$y_i = \Theta\left(\sum_{j\neq i} w_{ji}y_j + b_j\right)$$

其中$\Theta(z) = \mathrm{sgn}(z)$，也就是$z$的符号函数。进一步，我们假设网络具有对称权重，也就是$w_{ij} = w_{ji}$。 这样的网络并不**稳定**，也就是说一般地给定初始值并固定w之后，y_i会不断变化。

我们进而自然地考虑一个问题：这样的过程会终止吗？答案是**肯定的**。其证明的思路是考察如下的量

$$D \equiv \sum_{i<j} w_{ij}y_iy_j + \sum_i y_ib_i$$

的演化。考虑每一个y_i，在某次演化之前的值是y_i^-，而演化之后变成了$y_i^+ = -y_i^-$。我们发现如果$y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_j\right) \geq 0$，那么演化不会发生；而如果$y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_j\right) < 0$，那么就会演化，并且其贡献的$D$值变化为

$$\Delta D = (y_i^+ - y_i^-)\left(\sum_{j\neq i} w_{ij}y_j + b_i\right) = -2y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_j\right) > 0$$

但是注意到因为y_i的取值是有限种，因此必定在有限次内达到收敛。这一模型有点类似于物理里磁性物质的 **Ising Model**：整个体系的演化使得最终它们的总**能量**

$$E = -D = -\sum_{i<j} w_{ij}y_iy_j - \sum_i y_ib_i$$

达到（局部的）最小。在此情况下，我们会发现任何一个 neuron 的扰动还是会让体系回到这个最小值点。

我们立刻发现，这一模型的重要意义在于它可以用来**储存一些 pattern**：每一个初始的y_i可以是任意的，但是它们最终都到达有限个局部的最小值。换句话说，这一个网络**记住了**这些最小值。这也被称为 Associated Memory。

Hopfield Network: 训练

既然这个网络可以进行一些储存，那么我们自然希望这里存下来的 patterns 是有意义的，也就是我们的目标 image。对于N个 neuron 的 network，我们有着$\frac{N(N-1)}{2}$个 weight 参数需要学习。我们的目标就是希望最后的 Energy function 对应的 local minimum 就是我们想要的图片。在继续开始之前，为了方便讨论，我们取$b_i = 0$。这样，energy function 就成为了

$$E = -\sum_{i<j} w_{ij} y_i y_j$$

我们不妨从最简单的情况开始：假设我们只想储存一个 pattern。在这一情况下，**Hebbian learning rule** 给出了一个必定可行的方式：假设我们的目标是y^*，那么我们就取

$$w_{ij} = y_i^* y_j^*$$

这样，在$y_i = y_i^*, y_j = y_j^*$的时候，我们发现$E = -\frac{N(N-1)}{2}$已经是可能的最小值了。换句话说，这里的目标对应着全局的最小值。理论上可以证明，这一思路也可以被推广到学习N_p个不同的 patterns $P = \{y^p\}$的情形，这时的 Hebbian learning rule 写为

$$w_{ij} = \frac{1}{N_p} \sum_p y_i^p y_j^p$$

但是这一方法也有一定的缺陷。最为关键的一点就是，当储存的 pattern 比较多的时候，直观上就会发现每个y^p贡献的权重$y_i^p y_j^p$互相干扰，导致**原先的一些极小值被模糊，而反而在预料不到的地方出现了极大值**。比如说，考虑一个极端的情况：我们想要存储全部2^N种不同的 pattern，这样的计算立刻给出$w_{ij} = 0$。十分明显，这里所有的y的确都对应着局部极小值，但是所有位置的 energy 都是 0，这也就失去了意义。因此，我们转而希望找到一个方法来产生**稳定的局部极小值**，而非简单的局部极小值。

让我们来明确一下现在的问题：对于N个 neurons 的网络，我们希望找到一个权重w_{ij}，使得 energy

$$E = -\frac{1}{2} y^T W y$$

对于所有K个想要储存的 patterns $P = \{y^p\}$都是稳定的局部极小值。这是一个前面提到过的 optimization problem，因此我们需要定义一个合适的 objective function。怎样的函数比较合适呢？可以发现，我们需要把y^p对应的E最小化；但是我们同时还需要**把我们不想要的$y' \notin P$对应的E最大化**。这样，我们就决定选取

$$L = \sum_{y \in P} E(y) - \sum_{y' \notin P} E(y')$$

从而我们可以进行 Gradient descent：

$$W \leftarrow W - \eta \nabla_W L = W - \eta \left(\sum_{y \in P} y y^T - \sum_{y' \notin P} y' y'^T \right)$$

在把这一方法作为解决储存 pattern 问题的最终优化方式之前，让我们静下来好好考虑一下这个表达式的意义：我们看到，$\nabla_W L$的第一项$\sum_{y \in P} y y^T$对应着对 desired patterns 的 energy 进行下降；而第二项对应着对所有的 non-desired patterns 进行上升。但这有些浪费：比如原先就对应着E的最大值的y'，再增加它就完全没有必要。因此，我们可以**只考察使得E去到谷值的那些y'**：

$$W \leftarrow W - \eta \left(\sum_{y \in P} y y^T - \sum_{y' \notin P, y' \text{ is Valley}} y' y'^T \right)$$

但是该如何找到 valleys 呢？一个巧妙的发现是，**Hopfield Network 本身的演化就会把一个任意的初始值y带到一个 valley 内**！因此，我们就可以得到W的 update rule：

1.计算P内部y的 outer products $y y^T$；

2. 随机选取y'，

（1）对y'进行多次 Hopfield Network 演化直至收敛；

（2）计算$y' y'^T$。

3. 按照上面的表达式对W进行 update。

但是这个算法仍然存在问题：我们发现即便是对于各个 valley，它们也不是等价重要的：只有接近于我们的目标的、烦人的 spurious valleys 才是我们需要的。这一问题很好解决，我们只需要把上面方法中的“随机选取y'”改为“用P中元素（即我们想要的 patterns）初始化y'”即可。但是即便如此，依然还有可以改进的空间：我们发现如果某个目标y本身就在 valley 附近，那么这样的操作对这一 valley 的影响很大，可能会导致整体的变化。因此，我们希望不再影响整个 valley，而是**只把在我们的 patterns 附近的少数y'对应的 energy 抬升**。这对应着我们不再“对y'进行多次 Hopfield Network 演化直至收敛”，而是只“对y'进行 2~4 次 Hopfield Network 演化”。再把原先的 GD 改为 SGD，我们就可写出最后的 update 方式如下：

W的 SGD update rule：

1.计算P内部y的 outer products $y y^T$；

2. 用P中元素（即我们想要的 patterns）初始化y'

（1）对y'进行 2~4 次 Hopfield Network 演化；

（2）计算$y' y'^T$。

3. 用

$$W \leftarrow W - \eta \left(E_{y \in P}[y y^T] - E_{y'}[y' y'^T] \right)$$

对W进行 update。

Hopfield Network: The expanded network

前面的 Hopfield Network 已经可以储存 patterns，并且有了一定的训练方式。但是理论上证明了它的储存能力大概是$O(N)$的。注意到这里N并不是想多大就多大：N是 neurons 的数目，也就等于我们想要获得的数据y的维度。换句话说，对于128×128的黑白图片我们最多能存下128^2量级的 patterns，这并不满足实际应用的需求。为了增加可能储存 patterns 的数目，一个自然的想法就是引入 **hidden neurons**。对于数据维度为N的情况，我们也就设置N个 visible neurons 作为输出；而我们还加入K个 hidden neurons. 这样，它们之间的相互作用就可以使得我们的模型记忆更多的 patterns。

但是这样的结构面临着一些问题：这多余的K个 neurons 的数值该如何设置？同时，当模型训练好投入使用的时候，每一次输出都只用到N个 neurons 的结果，但是我们为了得到这个答案必须对全部的$K+N$个 neurons 进行演化，这带来很大的开销。因此，我们希望把 hidden 和 visible 进行**解耦**，而实现这个操作的方式就是使用一个 **probabilistic framework**。具体地，我们学习一个和参数W相关的概率分布$P_W(v,h)$，其中v代表 visible neurons 的数值，而h代表 hidden neurons 的数值。最后输出的时候我们只需要对 hidden neurons 采样，并得到

$$P_W(v) = \sum_h P_W(v,h)$$

就可以了。

我们解决了前面所述的问题，但又遇到了困难：我们知道 Hopfield network 是 deterministic 的，该如何把它变成 probabilistic 的呢？这时，来自物理中热力学理论的 **Free Energy** F起到了重要的作用。该理论表明，对于一个热力学系统的不同状态S，可以定义

$$F_T = \sum_S P_T(S)E_T(S) + kT\sum_S P_T(S)\log P_T(S)$$

其中$P_T(S)$代表给定温度T之下S状态的概率，而$E_T(S)$代表状态S的能量。一个物理系统的演化达到稳定状态时，总是使得F_T变成最小值。这一理论给出在稳定的状态下，这些概率值满足 **Boltzmann Distribution**：

$$P_T(S) = \frac{1}{Z}\exp\left(-\frac{E_T(S)}{kT}\right)$$

这里的Z不依赖于S，是概率的归一化常数，也被称为 **partition function**。按照这个公式，我们把前面得到的 Hopfield Network 的 energy 代入，并取$k=T=1$，可以给出

$$P(y) = \frac{1}{Z}\exp\left(\sum_{i<j} w_{ij}y_iy_j + b_iy_i\right)$$

这就是“热力学稳态”情况下输出 neuron 对应y值的概率分布。那么，该如何达到这个“热力学稳态”呢？类似于原来 deterministic 的网络的 flip 可以使得能量达到最小，现在我们也可以**给出一个 probabilistic 的“flip”操作**使得F最小（进而 neuron 的概率分布就

是上面的 Boltzmann Distribution)。为了得到这一操作，我们计算在稳态的情况下，固定$y_{j\neq i}$的值时$y_i = \pm 1$的条件概率，得到

$$\log\frac{P(y_i=1|y_{j\neq i})}{P(y_i=-1|y_{j\neq i})} = \log\frac{P(y_i=1,y_{j\neq i})}{P(y_i=-1,y_{j\neq i})} = -2\sum_{j\neq i} w_{ij}y_j - 2b_i$$

再改写$P(y_i=-1|y_{j\neq i}) = 1 - P(y_i=1|y_{j\neq i})$，我们可以得到

$$P(y_i=1|y_{j\neq i}) = \frac{1}{1+\exp(-2\sum_{j\neq i} w_{ij}y_j - 2b_i)} = \sigma\left(-2\sum_{j\neq i} w_{ij}y_j - 2b_i\right)$$

可以看到，这里的条件概率分布恰好是 sigmoid 函数！从这样的概率分布中取样也叫做 **Gibbs sampling**。我们由此给出这样的 **Stochastic Hopfield Network** 的 update rule：

1. 计算z_i（也叫做y_i处的 Field)：

$$z_i = 2\sum_j w_{ij}y_j + 2b_i$$

2. 按照概率

$$P(y_i=1|y_{j\neq i}) = \sigma(z_i)$$

进行对y_i的翻转。

对于一个已经训练完成的网络，我们只需要对于若输入y_0按照上面的 Gibbs Sampling 进行取样，再对最后几次的$y_{L-M+1}, y_{L-M+2}, \dots, y_L$求平均即可。当然，在很多实际的应用中，我们也可以直接输出经过L轮演化之后得到的y_L。

最后，如果我们希望得到一个精确的 pattern 而非概率分布的话，我们也可以使用所谓的 Annealing 方法，也就是逐步降低温度，像给铁器“退火”一样：

1. 初始化y_0和温度T；

2. 重复：

（1）按照

$$P(y_i=1|y_{j\neq i}) = \sigma\left(\frac{z_i}{T}\right)$$

进行演化若干轮；

（2）将温度T变为αT，其中$\alpha < 1$是常数。

3. 输出最终的y。

直观上来说，随着不断的“退火”，概率分布$\exp\left(-\frac{E(y)}{T}\right)$会向使得能量最小的$y$倾斜。的确，在理论上可以证明，这个系统会最终达到“最概然”的 pattern y。

Boltzmann Machine: 引入

我们可以把前面的 Stochastic Hopfield Network 进行一个一般化的阐述，这就是所谓 Boltzmann Machine。给定一个能量函数

$$E(y) = -\frac{1}{2}y^T W y$$

它给出输出y的一个概率分布

$$P(y) = \frac{1}{Z}\exp\left(-\frac{E(y)}{T}\right)$$

我们略微做一个小的更改，取$y_i \in \{0,1\}$，那么 Gibbs Sampling 就变成了

$$P\left(y_i = 1 \middle| y_{j\neq i}\right) = \sigma(z_i), z_i = \frac{1}{T}\sum_j w_{ji} y_j$$

我们现在的目标就是根据需要的 patterns 来学习W。类似之前决定性的 Hopfield Network 一样，我们需要给定一个 objective function 来进行 optimization。我们此时就希望将所有需要的 pattern 的概率最大化（这也就意味着削弱其他不想要的 pattern，因此这里 objective function 的设计比之前 Hopfield Network 的直接一些）。对于某一个特定的 pattern y，我们希望最大化

$$P(y) = \frac{1}{Z}\exp\left(\frac{1}{2}y^T W y\right) = \frac{\exp\left(\frac{1}{2}y^T W y\right)}{\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right)}$$

而对于多个 pattern，我们并不希望直接把概率相加；相反，我们希望把概率的**对数**相加（也就是 **log-likelihood**）。后面会看到这会带来计算上的很多方便之处。我们就有

$$L(W) = \frac{1}{N_p}\sum_{y\in P}\log P(y) = \frac{1}{N_p}\sum_{y\in P}\frac{1}{2}y^T W y - \log\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right)$$

也就是希望把它最大化。为此，我们应该进行 Gradient Ascent，并计算梯度

$$\nabla_{W_{ij}} L = \frac{1}{N_p}\sum_{y\in P} y_i y_j - \frac{1}{Z}\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right) y'_i y'_j$$

第一项很好计算；但是我们发现第二项中含有一个Z，而这个Z必须计算所有的y'对应的概率求和才可以得到。这将带来指数量级的计算量。但是这一理论的巧妙之处在于，我们发现刚好有

$$P(y') = \frac{1}{Z}\exp\left(\frac{1}{2}y'^T W y'\right)$$

因此，我们实际上可以写出

$$\nabla_{W_{ij}} L = \frac{1}{N_p}\sum_{y\in P} y_i y_j - \sum_{y'} P(y') y'_i y'_j = \frac{1}{N_p}\sum_{y\in P} y_i y_j - E_{y'}\left[y'_i y'_j\right]$$

此时，我们就不再需要计算全部的y'，而是只是按照概率分布$P(y')$随机的取样本集S，

并近似地计算出第二项：

$$\nabla_{W_{ij}} L \approx \frac{1}{N_p} \sum_{y \in P} y_i y_j - \frac{1}{|S|} \sum_{y' \in S} y'_i y'_j$$

这就是 **Monte-Carlo Approximation**。我们还可以立刻注意到，为了取样出S，我们只需要对于若干样本不断在 Stochastic Hopfield Network 中演化（也就是做 Gibbs Sampling）就可以了。

按照这个方法，我们就可以总结出 Boltzmann Machine 的完整训练方式：

Boltzmann Machine 的训练

1. 初始化W；

2. 重复：

（1）随机选取$y(0)$；

（2）不断对于$y_i(t)$进行循环，按照之前计算出来的$P\left(y_i(t+1)=1 \middle| y_{j \neq i}(t)\right)$进行 Gibbs Sampling，得到一系列样本$y(0), y(1), \dots, y(L)$直到收敛；

（3）取最后的M个 state 作为S：$S = \{y(L-M+1), y(L-M+2), \dots, y(L)\}$。

（4）用

$$\nabla_{W_{ij}} L \approx \frac{1}{N_p} \sum_{y \in P} y_i y_j - \frac{1}{|S|} \sum_{y' \in S} y'_i y'_j$$

估算梯度，并使用 Gradient Ascent

$$W_{ij} \leftarrow W_{ij} + \eta \nabla_{W_{ij}} L(W)$$

进行对W的更新。

Boltzmann Machine with Hidden Neurons

我们重新回到前面讨论的 hidden neurons：现在假设$y=(v,h)$，其中v代表 visible neurons，也就是我们的输出；而h是 hidden neurons。我们希望的输出是

$$P(v)=\sum_h P(v,h)$$

这个 **marginal distribution**。这时，注意到我们的目标不再是$P(y)$，而是

$$P(v)=\sum_h P(v,h)=\frac{\sum_h \exp\left(\frac{1}{2}y^T W y\right)}{\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right)}$$

因此我们的 objective function 也应该相应地改为

$$L(W)=\frac{1}{N_p}\sum_{v\in P}\log\left(\sum_h \exp\left(\frac{1}{2}y^T W y\right)\right)-\log\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right)$$

这个梯度就成为了

$$\nabla_{W_{ij}}L=\frac{1}{N_p}\sum_{v\in P}\frac{1}{\sum_h \exp\left(\frac{1}{2}y^T W y\right)}\sum_h \exp\left(\frac{1}{2}y^T W y\right)y_i y_j-\frac{1}{Z}\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right)y'_i y'_j$$

第二项我们已经知道该如何处理；而我们刚好发现第一项也可以变成概率（对于同样的v不同h的条件概率），并使用 Monte Carlo Approximation 进行估计。由此可以写出

$$\nabla_{W_{ij}}L\approx\frac{1}{N_p}\sum_{v\in P}E_h[y_i y_j]-E_{y'}[y'_i y'_j]$$

这样我们前面的训练方法也就可以直接推广：

Boltzmann Machine（with hidden neurons）的训练

1. 初始化W；

2. 重复：

   （1）对于每一个$v\in P$：

      （1.1）固定 visible neurons，随机选取 hidden neurons；

      （1.2）只对 hidden neurons 进行 Gibbs Sampling，得到样本S_c；

   （2）随机选取$y(0)$；

   （3）对所有 neurons 进行 Gibbs Sampling，得到样本S；

   （4）用

$$\nabla_{W_{ij}}L\approx\frac{1}{N_p}\sum_{v\in P}\frac{1}{|S_c|}\sum_{y\in S_c}y_i y_j-\frac{1}{|S|}\sum_{y'\in S}y'_i y'_j$$

估算梯度，并使用 Gradient Ascent

$$W_{ij} \leftarrow W_{ij} + \eta \nabla_{W_{ij}} L(W)$$

进行对W的更新。

有了这个方法之后，我们终于解决了之前在 deterministic 的 Hopfield Network 中无法解决的问题，也就是储存更多的 pattern。但是我们得到的远不止这些：Boltzmann Machine 使用概率的思路处理问题，这使得它可以对于不同的条件概率方式给出不同的工作模式。我们已经发现给定一个初始化，通过 Gibbs Sampling 我们可以实现 **pattern generation**；同时，如果固定一些 visible neurons，我们再做 Conditioned Gibbs Sampling 就可以实现 **pattern completion**。我们甚至可以把它用来做 **classification**：只需要训练的时候加入一个标记 label 的变量c，对完整的(y, h, c)进行上面的训练，就可以给出条件分布$P(c|v)$来确定给定 visible neuron 时候各个 label 的概率。如果加入了c，我们还可以进行 **conditional generation**：给定 label c，输出 visible neuron y。

但是，Boltzmann Machine 也因为它的全能性付出了一些代价：作为一个全连接的网络，训练需要很大的计算量；同时，Gibbs sampling 模拟的是物理学中的热力学系统，因此需要很长的时间才能收敛。这使得它很难真正应用在很大的数据上。更好的结构需要被引入，以用少部分能力的牺牲换取更优秀的计算效率。

Restricted Boltzmann Machine

前面 Boltzmann Machine 的耗时主要原因来自于其 neurons 之前完全连接。而在 **Restricted Boltzmann Machine** 中，neurons 的连接形成一个二部图：hidden neurons 只和 visible neurons 连接，反之亦然。可以发现，这样的图结构使得我们可以从原来逐个 neuron 的 Gibbs Sampling 中摆脱，而是进行 **Iterative Sampling**：

$$P(h_i = 1|v_j) = \sigma\left(z_i^{(h)}\right), z_i^{(h)} = \sum_j w_{ji} v_j$$

$$P(v_i = 1|h_j) = \sigma\left(z_i^{(v)}\right), z_i^{(v)} = \sum_j w_{ij} h_j$$

也就是说，我们可以一次批量对 visible neurons 进行 sample，一次对 hidden neurons 进行 sample，不断重复。同时，前面计算梯度时需要的 conditional probability $P(y|v_0)$ 可以用**一次 sample** 就完成，也就是只需要输出用v_0直接从上面公式计算出的h_0分布就可以了。这样，我们就可以给出梯度的新表达：

$$\nabla_{W_{ij}} L \approx \frac{1}{N_p} \sum_{v_0 \in P} (v_0)_i \frac{1}{|S_c|} \sum_{h_0 \in S_c} (h_0)_j - \frac{1}{S} \sum_{v_0' \in S} (v_\infty)_i (h_\infty)_j$$

我们发现第一项已经基本不需要计算，但第二项仍然需要很多次演化。这时，我们可以回忆起之前在 Hopfield Network 的分析时发现，只需要做 2~4 次演化就可以覆盖到 pattern 附近的 valley，进而将它们提升；而更远的 valley 无关紧要。因此，我们这里只做一完整轮 Gibbs Sampling 即可。这样公式就成为

$$\nabla_{W_{ij}} L \approx \frac{1}{N_p} \sum_{v_0 \in P} (v_0)_i \frac{1}{|S_c|} \sum_{h_0 \in S_c} (h_0)_j - \frac{1}{S} \sum_{v_0' \in S} (v_1)_i (h_1)_j$$

这里v_1, h_1的计算方式是$v_0 \to h_0 \to v_1 \to h_1$。这样我们就可把 Gibbs sampling 的计算次数减少到 3 次。

Restricted Boltzmann Machine 克服了普通 Boltzmann Machine 的收敛困难。在其之后，人们的研究方向变为试着通过更加 **deep** 的网络结构实现更强的能力。从而有了 Deep Belief Net、Deep Boltzmann Machine 等结构，它们在实际应用中都有着很好的效果。

General Energy-based Model

我们可以把前面的众多模型进行一个总结：它们都是希望得到一个关于输出x的概率分布$P(x)$，而这个分布都是通过一个 Energy function $E(x;\theta)$实现的：

$$P(x)=\frac{1}{Z}\exp\big(-E(x;\theta)\big)$$

其中

$$Z=\int \exp\big(-E(x;\theta)\big)\,dx$$

是 partition function。这里 exponential 的形式的好处我们也已经看到：它可以减少一定的计算。这样的"广义"Energy function 更加普适，我们可以把其中的$E(x;\theta)$换成任何想要的函数，甚至 CNN。

我们也可以总结出通用的 objective function：

$$L=\frac{1}{N_p}\sum_{x\in P}-E(x;\theta)-\log\sum_{x'\notin P}\exp\big(-E(x';\theta)\big)$$

和为了避免 partition function Z的出现的梯度近似方式：

$$\nabla_\theta L\approx\frac{1}{N_p}\sum_{x\in P}-\nabla_\theta E(x;\theta)+\frac{1}{|S|}\sum_{x'\in S}\nabla_\theta E(x';\theta)$$

但是对于一般的 Energy-based model，E不一定对于x的各个分量有着之前的简单形式，因此从中取 sample 会略微困难。但是可以证明，下面的算法

从 Generic Energy-based Model 中取样

1. 随机初始化x^0

2. 重复：

（1）为x^t加一些噪声，得到x'；

（2）（i）如果$E(x')<E(x^t)$，那么取$x^{t+1}=x'$；

（ii）如果$E(x')\geq E(x^t)$，那么以$\exp\big(E(x^t)-E(x')\big)$的概率取$x^{t+1}=x'$；

在很多轮后的输出就是按照$\frac{1}{Z}\exp(-E(x';\theta))$ 分布的x。具体的内容会在下一讲提及。