CA684: Deep Learning 04/Apr/2014

Tsuyoshi Okita

Dublin City University,
tokita@computing.dcu.ie

1 Overview

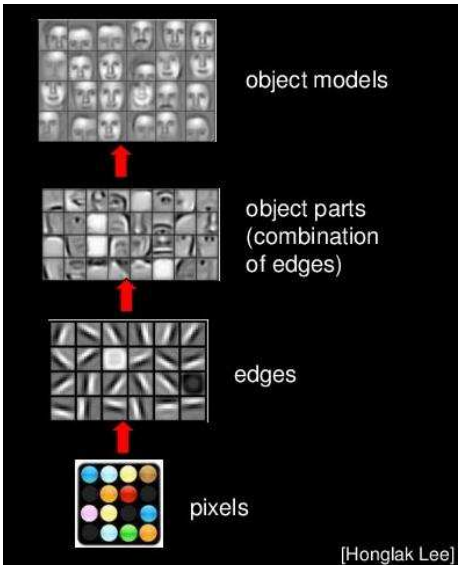


Fig. 1. Hierarchical feature representations of picture of faces [Lee]. For given training set of face images, deep learning aims at obtaining hierarchical feature representations in an automatic way.

This class aims at grasping the overview of deep learning as one hottest area of research (Figure 1 shows the idea of deep learning). However, as we haven't properly introduced neither feed-forward neural networks nor undirected graphical models, this class only covers the very introductory materials. The goal of this class is to understand three questions below.

1. What is deep learning?
2. Grasp the basic algorithm of Recurrent Neural Network (RNN) or Restricted Boltzmann Machine (RBM)?
3. Can you explain some toy application? (e.g. a summation of two binary numbers)

We restrict ourselves to introduce very basic characteristics of deep learning by two algorithms, RNN and RBM. Nevertheless, the understanding of these two algorithms will facilitate the further reading of the deep learning literatures although it will need at least one more lecture to complete the rapid overview of this field. Note that deep learning is a state-of-the-art in Machine Learning and is evolving (Hinton, 2007; Bengio, 2009; Murphy, 2012). A topic might be considerably updated in a few years time.

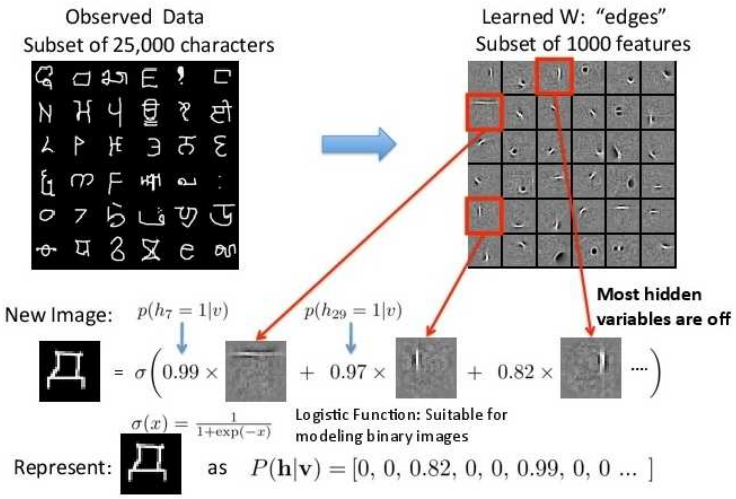


Fig. 2. Schematic illustration of sparse coding of Olshausen et al. using artificial data. Olshausen et al. observed that sparse coding of natural images produces wavelet-like oriented filters that resemble the receptive fields of simple cells in the visual cortex. Given a potentially large set of input patterns (left figure = observed data of 25,000 characters), sparse coding algorithms attempt to automatically find a small number of representative basis vectors (right figure of 1000 patterns). In test phase, these basis vectors reproduce the original input patterns. The human primary visual cortex is estimated to be overcomplete (Overcomplete codings smoothly interpolate between input vectors and are robust under input noise) by a factor of 500, so that, for example, a 14 x 14 patch of input (a 196-dimensional space) is coded by roughly 100,000 neurons.[Salakhutdinov, Tutorial]

Deep learning is to learn automatically the hierarchical feature representations, which we follow the definition of Yann LeCun (See ICML2013 Tutorial Video on techtalks.tv), Yoshua Bengio (Bengio, 2009), Geoffrey Hinton (Hinton, 2007), or others. Hence, the focus of this class is on "feature extraction" and "automatic Machine Learning algorithm" in order to realize this. One important question which was asked by Hinton (Hinton and Teh, 2001) in 2001 still in

the formation period of deep learning, "Is a linear causal model of Olshausen et al. (Olshausen and Field, 1996) plausible enough to explain the human vision?"

"Early research on the response properties of individual neurons in visual cortex typically assumed that neurons were rather specific feature detectors that only fired when they found a close match to the feature of interest. For the early stages of the visual cortex, this assumption has largely been replaced by the idea that the receptive fields of neurons represent basis functions and the neural activities represent coefficients on these basis functions. The sensory input is then represented as a weighted linear combination of the basis functions which is equivalent to assuming that the sensory input is generated by a causal linear model with one layer of latent variables and that low-level perception consists of inferring the most likely values of the latent variables given the sensory data. With the added assumption that the latent variables have heavy-tailed distributions, it is possible to learn biologically realistic receptive fields by fitting a linear, causal generative model to patches of natural images (Olshausen and Field, 1996)". (Text itself is from (Hinton and Teh, 2001). See Figure 2).

We study two algorithms, Recurrent Neural Network (RNN) and Restricted Boltzmann Machine (RBM): RNN is in the form of directed graphical model while RBM is undirected graphical model. These two can be thought of as an introduction to such algorithms, rather than these are the goals themselves.

2 Deep Learning

This section is based on mostly the excerpts from the following article.

- Yoshua Bengio. 2009. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Deep Learning Deep learning will be useful when we are trying to solve some complicated AI task such as computer vision, speech, and NLP. In these tasks, we may need to assume that the computational machinery necessary to express complex behaviors requires highly varying mathematical functions. Furthermore, depth of architecture, which refers to the number of levels of composition of non-linear operations in the function learned, may be large. In this case, our interest will be related to the complicated functions which represents high-level abstractions. In order to learn some kind of complicated functions that can represent high-level abstractions, one may need deep learning and deep architectures. Note that once a good representation has been found at each level, it can be used to initialize and successfully train a deep neural network by supervised gradient-based optimization. **Deep learning aims at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features ("hierarchical feature representations")**. Automatically learning features at multiple levels of abstraction allows a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features. This

is unsupervised learning. **Deep architectures** are composed of multiple levels of non-linear operations, such as in neural nets with many hidden layers or in complicated propositional formulae re-using many sub-formulae. Searching the parameter space of deep architectures is a difficult task. Recent deep learning algorithms such as Deep Belief Networks can tackle this problem with notable success, beating the state-of-the-art.

Automatic Discovery Ideally we would like learning algorithms that enable this discovery with as little human effort as possible. In general and for most factors of variation underlying natural images, we do not have an analytical understanding of these factors of variation. We do not have enough formalized prior knowledge about the world to explain the observed variety of images. The number of visual and semantic categories that we would like an "intelligent" machine to capture is rather large. The focus of deep architecture learning is to automatically discover such abstractions, from the lower level features to the highest level concepts.

Analogy of Humans The mammal brain is organized in a deep architecture with a given input percept[1] represented at **multiple levels of abstraction**, each level corresponding to a different area of cortex. The brain appears to process information through **multiple stages of transformation of abstraction**. This is particularly clear in the primate visual system: detection of edges, primitive shapes, and moving up to gradually more complex visual shapes. Each level of abstraction found in the brain consists of the "activation" (neural excitation) of a small subset of a large number of features, which is called a **distributed representation** (Hinton et al., 1986).

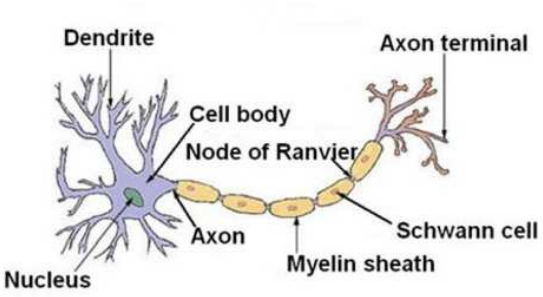


Fig. 3. Deep learning and neural networks are biologically motivated.

Technological Breakthrough in Neural Network Inspired by the architectural depth of the brain, neural network researchers had wanted for decades to train

[1] the mental result of perceiving

deep multi-layer neural networks, but no successful attempts were reported before 2006. **A breakthrough happened in 2006; Hinton and collaborators introduced Deep Belief Networks**, with a learning algorithm that greedily trains one layer at a time, exploiting an unsupervised learning algorithm for each layer, **a Restricted Boltzmann Machine**.

Statistical Strength Even though statistical efficiency is not necessarily poor when the number of tunable parameters is large, good generalization can be obtained only when adding some form of prior. In contrast to learning methods based on local generalization, the total number of patterns that can be distinguished using a distributed representation scales possibly exponentially with the dimension of representation. Deep learning algorithms are based on learning intermediate representations which can be shared across tasks. Many of these learned features are shared among m tasks provides sharing of statistical strength in proportion to m.

Benefits versus Shallow Learning Some functions may not be able to be efficiently represented (in terms of number of tunable elements) by architectures that are too shallow. In such a case, deep architectures may be able to compactly represent **highly-varying functions** [2] which would otherwise require a very large size to be represented with an inappropriate architecture. **The depth of architecture** can be very important from the point of view of statistical efficiency. Many shallow architectures associated with non-parametric learning algorithms has weakness in its locality in input space of the estimator. This is since functions that can be compactly represented with a depth k architecture could require a very large number of elements in order to be represented by a shallower architecture. Reorganizing the way in which computational units are composed can have a drastic effect on the efficiency of representation size. Theoretical results suggest that it is not the absolute number of levels that matters, but the number of levels relative to how many are required to represent efficiently the target function. We would expect that compact representations of the target function would yield better generalization.

3 Recurrent Neural Network

This section is based on mostly the excerpts from the following article.

- Alex Graves. 2012. Supervised sequence labeling with recurrent neural networks (textbook, studies in computational intelligence). *Springer*.
- Tomas Mikolov. 2012. Statistical language models based on neural networks. *PhD thesis at Brno University of Technology*.

[2]

Definition 1 (highly-varying function). *We say that a function is highly-varying when a piecewise approximation of that function would require a large number of pieces.*

– Deep learning page of Jurgen Schmidhuber. http://www.idsia.ch/ juergen/deeplearning.html and http://www.idsia.ch/ juergen/DeepLearning17April2014.pdf
– Feed-forward Neural Network. (Andrew Ng's coursera lecture, Bishop's books (Bishop, 1995; Bishop, 2006)).

From earlier days, it is known that the computational power of (fully connected) RNN (with rational valued weights) is theoretically equivalent to the universal Turing machine (finite state automata) (Siegelmann and Sontag, 1991). However, due to the practical difficulty of training the networks of many layers correctly it took 15-20 years until neural networks regain popularity. First, one of the biggest problems has been the **vanishing gradient problem** (Hochreiter, 1991). Among numerous attempts to overcome this **vanishing gradient problem**, one important architecture is Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). It is tempting to focus on this but in order to understand this we need to understand feed-forward neural networks and RNN. Due to the timing constraints, we only study feed-forward neural networks and RNN. The details of LSTM can be found, for example, in the paper of Alex Graves. Second, RNN fits for handling sequential data which appears in NLP, or time-series data in financial mathematics. In NLP, language modeling is one important research area. It is noted that the abbreviation of Recursive Neural Network is also RNN (Socher et al., 2012) which is different.

3.1 Feed-forward Neural Network

Suppose we solve a multiclass classification problem by a feed-forward neural network classifier. Let $\{(X_1, Y_1), \ldots, (X_m, Y_m)\} \in \mathcal{R}^d \times \{1, \ldots, K\}$ be a training set and $g(x) : \mathcal{R}^d \to \{1, \ldots, K\}$. We use a cross-entropy loss function $\ell(y, \hat{y})(= -\sum_{k=1}^{K} \hat{y}_k \log y_k)$ where $\hat{y}(= g(x))$.

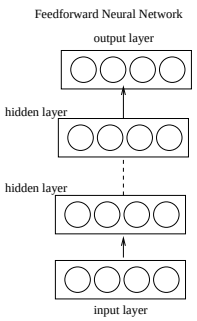


Fig. 4. Left figure shows a feed-forward neural network which has two (or more) hidden layers.

Figure 4 shows a feed-forward neural network. Let $w_{ij}^{(k)}$ be the weight from unit i in the $(k-1)$-th layer to unit j in the k-th layer, a denote the weighted sum of the previous units, and b denote the activation of a, and $\sigma(x)(=\frac{1}{1+e^{-x}})$ denote a logistic activation function. We denote by superfix $^{(n)}$ a n-th layer, which we will use for w_{ij}, a, b: hence $w_{ij}^{(n)}$, $a^{(n)}$, and $b^{(n)}$ means n-th layer of weights, a, and b. (For simplicity, we omit bias terms.)

Gradient Descent Optimization A gradient descent method shows a general method to update weights systematically by gradient information. There are several variants which improve some aspect of this method. One method is called a stochastic gradient descent method. In order to find a sufficiently good minimum, it will be required to run a gradient-based algorithm multiple times, each time using a different starting point which is randomly chosen, and comparing the performance on an independent validation set. The overall algorithm for feed-forward neural network is shown in Algorithm 1.

Algorithm 1 Gradient Descent Algorithm.

```
1: while stopping criteria not met do
2:     for each example in the training set do
3:         Run forward path and backward path to calculate the gradient.
4:             Δw_i ← Δw_i + ηℓ(w^[τ]) (gradient descent)
5:         Update weights with gradient descent algorithm
6:             w_i ← w_i + Δw_i
```

Let ℓ denote a loss function, $\nabla\ell$ denote a gradient of loss function, η denote learning rate, and $w^{[\tau]}$ is the weights at an iteration $[\tau]$. Gradient descent is $w^{[\tau+1]} = w^{[\tau]} - \eta\nabla\ell(w^{[\tau]})$, while stochastic gradient descent $w^{[\tau+1]} = w^{[\tau]} - \eta\nabla\ell_n(w^{[\tau]})$ where $\ell(w) = \sum_{n=1}^{N}\ell_n(w)$.[3]

Forward Path and Backward Path Line 3 and 4 of Algorithm 1 needs to calculate the gradient information, which is done by the forward and backward path.

Forward path is straight forward in which we need to calculate the following.

$$\begin{cases} b_j^{(1)} \leftarrow x_j & \text{input layer} \\ a_j^{(2)} = \sum_{i=1}^{m} w_{ij}^{(1)} b_i^{(1)} \\ b_j^{(2)} = \sigma(a_j^{(2)}) & \text{1-st hidden layer} \end{cases}$$

$$\cdots$$

$$\begin{cases} a_j^{(n-1)} = \sum_{i=1}^{m} w_{ij}^{(n-2)} b_i^{(n-2)} \\ b_j^{(n-1)} = \sigma(a_j^{(n-1)}) & (n-2)\text{-th hidden layer} \end{cases}$$

[3] Other than these, there are several methods: conjugate gradients, BFGS, and L-BFGS.

$$\begin{cases} a_j^{(n)} &= \sum_{i=1}^{K} w_{ij}^{(n-1)} b_i^{(n-1)} \\ b_j^{(n)} &= \sigma(a_j^{(n)}) \\ y_j &\leftrightarrow b_j^{(n)} \end{cases} \quad \text{output layer}$$

Backward path needs to propagate errors. First, a cost function $J(\theta)$ becomes as follows:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((\sigma(x^{(i)}))_k) + (1 - y_k^{(i)}) \log(1 - (\sigma(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} (w_{ij}^{(l)})^2$$

where $\sigma(x^{(i)})_k$ denote $a_k^{(n)}$ and the second term is called a regularization term. Second, we want to minimize this cost function. This needs to calculate $J(\theta)$ and $\frac{\partial}{\partial w_{ij}^{(n)}} j(\theta)$. In order to obtain the latter quantity, let us define $\delta^{(n)} \equiv \frac{\partial J(\theta)}{\partial a^{(n)}}$. Then, this quantity can be propagated in the following way.

$$\begin{cases} \delta_j^{(n)} &= b_j^{(n)} - y_j & \text{output layer} \\ \delta_j^{(n-1)} &= \sigma'(a^{(n-1)}) \sum_p w_{pj}^{(n-1)} \delta_k^{(n)} & (n-2)\text{-th hidden layer} \\ \cdots & & \\ \delta_j^{(2)} &= \sigma'(a^{(2)}) \sum_p w_{pj}^{(2)} \delta_p^{(3)} & \text{1-st hidden layer} \end{cases}$$

Algorithm 2 shows the overall algorithm how to compute $\frac{\partial}{\partial w_{ij}^{(n)}} J(\theta)$.

Algorithm 2 Calculation of gradient information for feed-forward neural network.

Training set $\{(X_1, Y_1), \ldots, (X_m, Y_m)\}$
Feed-forward neural network of N layers, input m units.
Set $\Delta_{ij}^{(n)} = 0$ (**for all** n, i, j)
For i=1 **to** m
    Set $b^{(1)} = x^{(i)}$
    Perform forward propagations to compute $b^{(n)}$ **for** n=2, 3,..., N.
    Using $y^{(i)}$, compute $\delta^{(N)} = b^{(N)} - y^{(N)}$
    Compute $\delta^{(N-1)}$, $\delta^{(N-2)}$, ..., $\delta^{(2)}$.
    $\delta_{ij}^{(n)} = \delta_{ij}^{(n)} + b_j^{(n)} \delta_i^{(n+1)}$
$D_{ij}^{(n)} = \frac{1}{m}\delta_{ij}^{(n)} + \lambda w_{ij}^{(n)}$ **if** $j \neq 0$
$D_{ij}^{(n)} = \frac{1}{m}\delta_{ij}^{(n)}$ **if** $j = 0$
$\frac{\partial}{\partial w_{ij}^{(n)}} J(\theta) = D_{ij}^{(n)}$

3.2 Recurrent Neural Network

The left two figures in Figure 5 shows recurrent neural networks (RNNs): the left most figure shows the folded form and the second left figure shows the unfolded form. The unfolded form is often called backpropagation through time (BPTT) (Robinson and Fallside, 1987). RNNs allow cycles between hidden layers, which is the difference from the feed-forward neural networks. However, this simple difference is considered to be a key characteristic of RNNs that (fully connected) RNNs (with rational valued weights) is equivalent to the universal Turing machine (finite state automata) (Siegelmann and Sontag, 1991) despite of its difficulty of training and its set up.

There are various models of RNNs. Figure 5 show some of these: from left, RNN, Backpropagation through time (BPTT) (Robinson and Fallside, 1987), bidirectional RNNs (Schuster and Paliwal, 1997),[4] and Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997).[5]

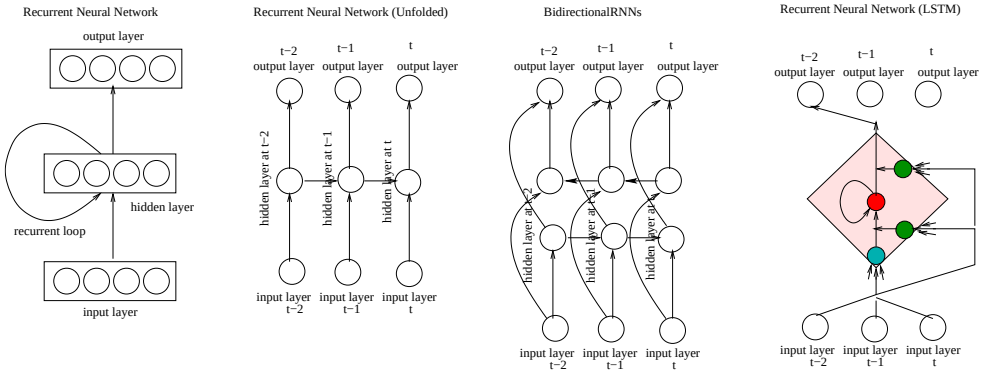


Fig. 5. From left to right, RNN, RNN with BPTT (unfolded), bidirectional RNN, RNN with LSTM.

As the representative model of RNNs we focus on RNN with backpropagation through time (BPTT) (See the second figure from the left in Figure 5). Consider the network with I input units, H hidden units, and K output units. Let $x_i^{(t)}$ denote the value of input i at time t, $a_j^{(t)}$ be the network input to unit j at time t, $b_j^{(t)}$ be the activation of unit j at time t. Similarly with feed-foward network,

[4] Bidirectional RNNs intend to access not only the past context but also the future context, which has advantage in sequence labeling task.

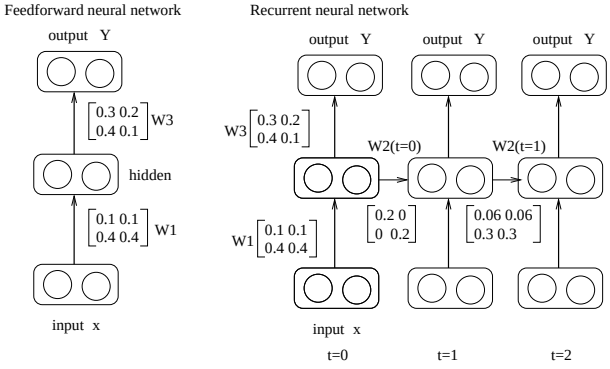
[5] Other than these, there are Elman networks, Jordan networks, time delay neural networks, and echo state networks.

let us define $\delta_j^{(t)} \equiv \frac{\partial J(\theta)}{\partial a_j^{(t)}}$.

$$
\begin{aligned}
\text{Forward Path} \quad & \begin{cases} a^{(t)} = \sum_{i=1}^{I} w_{ih} x_i^{(t)} + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{(t-1)} \\ b^{(t)} = \sigma(a^{(t)}) \end{cases} \\
\text{Backward Path} \quad & \begin{cases} \delta^{(t)} = \sigma'(a^{(t)})(\sum_{k=1}^{K} \delta_k^{(t)} w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{(t+1)} w_{hh'}) \\ \delta_j^{(t)} = \frac{\partial J(\theta)}{\partial a_j^{(t)}} \end{cases}
\end{aligned} \tag{1}
$$

The key difference is that the recurrent path is executed along with time elapse, which are appeared in the equations of forward and backward paths. For the forward path, the contribution of first term of input remains. For the backward path, the contribution of first term of output remains. It would be easy to understand intuitively if you do Example 1 which calculates forward path. (Although the backward path is more elaborate, it is similar if once understanding the backpropagation of feed-forward neural network).

Example 1. For given weights shown in Figure below (Ignore biases), run the forward path to obtain y. Consider a logistic sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ in output layer (Ignore activation function in hidden layer). Consider the recurrent loop is calculated according to time (BPTT) from $t = 0$ to $t = 2$. Note that the table below is the answer.



Feed-forward NN

x_1	x_2	h_1	h_2	y_1	y_2	$h(y_1)$	$h(y_2)$
0	0	0	0	0	0	1.0	1.0
0	1	0.1	0.4	0.11	0.08	1.12	1.08
1	0	0.1	0.4	0.11	0.08	1.12	1.08
1	1	0.2	0.8	0.22	0.16	1.24	1.17

Recurrent NN (t=0)

x_1	x_2	h_1	h_2	y_1	y_2	$h(y_1)$	$h(y_2)$
0	0	0	0	0	0	1.0	1.0
0	1	0.1	0.4	0.11	0.08	1.12	1.08
1	0	0.1	0.4	0.11	0.08	1.12	1.08
1	1	0.2	0.8	0.22	0.16	1.24	1.17

Recurrent NN (t=1)

x_1	x_2	h_1	h_2	y_1	y_2	$h(y_1)$	$h(y_2)$
0	0	0	0	0	0	1.0	1.0
0	1	0.48	0.13	0.17	0.21	1.19	1.23
1	0	0.48	0.13	0.17	0.21	1.19	1.23
1	1	0.24	0.96	0.26	0.19	1.30	1.21

Recurrent NN (t=2)

x_1	x_2	h_1	h_2	y_1	y_2	$h(y_1)$	$h(y_2)$
0	0	0	0	0	0	1.0	1.0
0	1	0.48	0.13	0.03	0.18	1.03	1.19
1	0	0.48	0.13	0.03	0.18	1.03	1.19
1	1	0.24	0.96	0.07	0.36	1.07	1.43

4 Restricted Boltzmann Machine

This section is based on mostly the excerpts from the following article.

- Geoffrey E. Hinton. 2007. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11.
- Geoffrey E. Hinton. 2010. A practical guide to training restricted Boltzmann machines. *UTML TR2010-003*.
- Yoshua Bengio. 2009. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127.
- Deep learning tutorial on the net provides detailed explanation about RBM including software implementation using Theano. http://deeplearning.net/tutorial/rbm.html.

Restricted Boltzmann Machine is an energy-based machine (Markov random field). This is undirected graphical model in which the topological ordering is not defined (as in directed graphical model). Instead, potential functions with each maximal clique is defined. An energy-based model is popular in Physics / statistical mechanics: the characteristic of "memory" is often mentioned in ising model or Hopfield network. This class of algorithms include Hopfield network, Boltzmann machine, Restricted Boltzmann machine, and Ising model. It is noted that some basic materials of undirected graphical model may be useful such as Hammersley-Clifford theorem and potential functions (Refer Appendix).

4.1 Predictive Samples

Definition 2 (Energy models; Hopfield Net, Boltzmann Machine, RBM). *The energy of these three models are defined as follows.*

- *Hopfield Net*

$$E(s) = -\sum_{j} s_i b_i - \sum_{i<j} s_i s_j w_{ij}$$

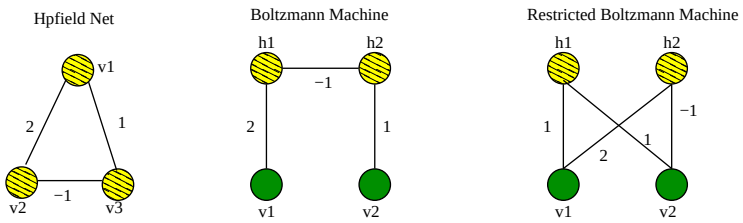


Fig. 6. From left to right three figures show Hopfield network, Boltzmann machine and RBM. The number attached near the arc shows weights which are used for a small excercise in Example 2.

Hopfield Net					
s_1	s_2	s_3	-E	e^{-E}	$p(v)$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Boltzmann Machine							
v_1	v_2	h_1	h_2	-E	e^{-E}	$p(v,h)$	$p(v)$
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

RBM			
-E	e^{-E}	$p(v,h)$	$p(v)$

Table 1. Question is to fill in the yellow column in the above three tables. There are three tables where each table corresponds to the networks written in Figure 6: the left most corresponds to Hopfield net, the middle is Boltzmann Machine, and the right most is Restricted Boltzmann Machine.

– *Boltzmann Machine*

$$E(v,h) = -\sum_{i \in visible} v_i b_i - \sum_{k \in hidden} h_k b_k - \sum v_i v_j w_{ij} - \sum v_i h_j w_{ik} - \sum h_k h_l w_{kl}$$

– *Restricted Boltzmann Machine*

$$E(v,h) = -\sum_{i \in visible} v_i b_i - \sum_{k \in hidden} h_k b_k - \sum v_i h_j w_{ik}$$

Example 2 (Energy-based Model). By this example we are going to infer the posterior samples (to generate samples) by hand when the model is specified. We consider three kinds of energy-based models: Hopfield net, Boltzmann machine, and Restricted Boltzmann Machine. For simplicity, we are given the model parameters of weights (we ignore bias) as in Figure 6.

1. For simplicity we suppose that all the bias $b_i = 0$. For all configurations, calculate energy E. (Hint. In Hopfield net, suppose that $v1, v2, v3 = [1, 1, 1]$. The energy can be calculated by $E(s) = -\sum_{i<j} s_i s_j w_{ij} = 1 \times 1 \times 2 + 1 \times 1 \times (-1) + 1 \times 1 \times 1 = 0$.)
2. Then for Hopfield net, calculate e^{-E}, $p(v)$. (Hint. In Hopfield net, calculate it using a calculator. If you use *python*, use numpy.exp(1).) In order to calculate $p(v)$, use $p(v) = \frac{e^{-E}}{\sum e^{-E}}$.

3. For Boltzmann Machine and RBM, calculate e^{-E}, $p(v,h)$, then $p(v)$. For the calculation of $p(v)$, you need to marginalize out h_1 and h_2 from the tables.

The answer of the above example is in Table 2.

Hopfield Net					
s_1	s_2	s_3	-E	e^{-E}	$p(v)$
0	0	0	0	1	0.045
0	0	1	0	1	0.045
0	1	0	0	1	0.045
0	1	1	-1	0.37	0.017
1	0	0	0	1	0.045
1	0	1	1	2.72	0.124
1	1	0	2	7.39	0.337
1	1	1	2	7.39	0.337
-	-	-	-	21.87	-

Boltzmann Machine							
v_1	v_2	h_1	h_2	-E	e^{-E}	$p(v,h)$	$p(v)$
0	0	0	0	0	1	0.025	0.084
0	0	0	1	0	1	0.025	
0	0	1	0	0	1	0.025	
0	0	1	1	-1	0.37	0.009	
0	1	0	0	0	1	0.025	0.144
0	1	0	1	1	2.72	0.069	
0	1	1	0	0	1	0.025	
0	1	1	1	0	1	0.025	
1	0	0	0	0	1	0.025	0.306
1	0	0	1	0	1	0.025	
1	0	1	0	2	7.39	0.187	
1	0	1	1	1	2.72	0.069	
1	1	0	0	0	1	0.025	0.468
1	1	0	1	1	2.72	0.069	
1	1	1	0	2	7.39	0.187	
1	1	1	1	2	7.39	0.187	
-	-	-	-	-	39.60	-	-

RBM			
-E	e^{-E}	$p(v,h)$	$p(v)$
0	1	0.0140	0.056
0	1	0.0140	
0	1	0.0140	
0	1	0.0140	
0	1	0.0140	0.0712
-1	0.37	0.0052	
1	2.72	0.038	
0	1	0.0140	
0	1	0.0140	0.436
2	7.39	0.103	
1	2.72	0.038	
3	20.1	0.281	
0	1	0.0140	0.436
1	2.72	0.038	
2	7.39	0.103	
3	20.1	0.281	
-	71.5		

Table 2. This table shows an answer for Example 2.

4.2 Negative Log-Likelihood

In Example 2 we inferred the predictive samples when the model is specified. We understand that the global energy is the sum of many contributions and that each contribution depends on one connection weight and the binary states of two neurons.

For an energy-based model without hidden variables (e.g. Hopfield networks), we consider to evaluate the negative log-likelihood (Refer Figure 3 for various loss functions) and learn the parameters.

$$p(x) = \frac{e^{-E(x)}}{Z} \quad \text{where} \quad Z = \sum_x e^{-E(x)} \tag{2}$$

Using the (stochastic) gradient descent which seeks the direction of the gradient of the negative log-likelihood of the training data, we learn the optimal model parameters by minimizing the negative log-likelihood.

$$\ell(\theta, \mathcal{D}) = -\mathcal{L}(\theta, \mathcal{D}) \quad \text{where} \quad \mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log p(x^{(i)}) \tag{3}$$

For an energy-based model with hidden variables (e.g. Boltzmann Machine and RBM) we use the free energy where free energy is defined as the negative log of the probability with the state of the binary hidden variable integrated out (Hinton and Teh, 2001; Bengio, 2009):

$$\mathcal{F}(x) = -\log \sum_{h} e^{-E(x,h)} \tag{4}$$

By this, we can write the following.

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \quad \text{where} \quad Z = \sum_{x} e^{-\mathcal{F}(x)} \tag{5}$$

The gradient of the negative log-likelihood, in this case, is as follows:

$$-\frac{\partial \log p(x)}{\partial \theta} = \frac{\partial \mathcal{F}(x)}{\partial \theta} - \sum_{\tilde{x}} p(\tilde{x}) \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta} \tag{6}$$

$$= \frac{\partial \mathcal{F}(x)}{\partial \theta} - \frac{1}{|N|} \sum_{\tilde{x} \in N} \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}. \tag{7}$$

where the second line means that we are doing sampling. The first term $\frac{\partial \mathcal{F}(x)}{\partial \theta}$ increases the probability of training data by reducing the corresponding free energy (positive phase), while the second term $\frac{1}{|N|} \sum_{\tilde{x} \in N} \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}$ decreases the probability of samples generated by the model (negative phase). Figure 7 illustrates the positive phase as "push down" and the negative phase as "pull up".

$$\mathcal{F}(v) = -\sum_{i \in visible} v_i b_i - \sum_{i} \log \sum_{h_i} e^{h_i (c_i + W_i v)} \tag{8}$$

4.3 RBM

In the previous two subsections we briefly overview the energy-based machine.

1. RBM does not have hidden-hidden and visible-visible interactions. (Compare the network architecture of RBM with Boltzmann Machine in Figure 6).
2. The gradient of negative log-likelihood in RBM has two terms. (See Equation (7). Compare the model without hidden variables and the model with hidden variables)

The network architecture of RBM is depicted in Figure 8. There is no connection between nodes in the same layers: all the links are between hidden and visible variables. Hence, the hidden variables in RBM are conditionally independent given the visible variables. We say this using "explaining away" (See the explanation of "explaining away" in Figure 9). With this restriction, the

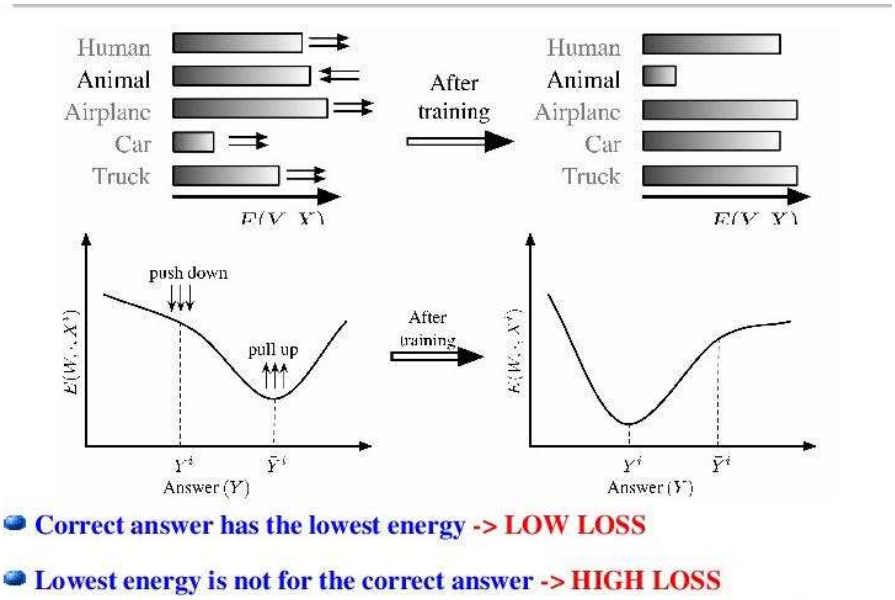


Fig. 7. The first term needs "push down" the energy while the second term needs "pull up" the energy in the left side of figure. [LeCun]

RBM possesses the useful properties. First, the conditional distribution over the hidden units factorizes given the visibles as in (9):

$$P(h|v) = \prod_i P(h_i|v)$$
$$P(h_i = 1|v) = \sigma(\sum_j W_{ji} v_j + d_i) \tag{9}$$

Second, the conditional distribution over the visible units given the hidden units factorizes as well as in (10):

$$P(v|h) = \prod_j P(v_j|h)$$
$$P(v_j = 1|h) = \sigma(\sum_j W_{ji} h_i + b_j) \tag{10}$$

These conditional factorization properties imply that most inferences of our interests are readily tractable. Given the conditional independence in Eq. (9) the RBM feature representation, which is the set of posterior marginals $P(h_i|v)$, becomes instantly available. Hence the positive phase of Eq. 7 is readily tractable.

For the negative phase of Eq. 7, the partition function, however, still involves summing an exponential number of terms. However, due to the conditional in-

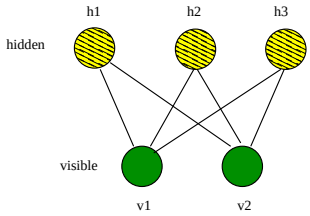


Fig. 8. Network architecture of RBM.

dependence property of RBM we can sample $(\tilde{v}^{(l)}, \tilde{h}^{(l)})$ using a block Gibbs sampling as in (11):

$$\begin{aligned} \tilde{v}^{(l)} &\sim P(v|\tilde{h}^{(l-1)}) \\ \tilde{h}^{(l)} &\sim P(h|\tilde{v}^{(l)}) \end{aligned} \tag{11}$$

The most naive approach is to start a Gibbs sampling chain until the chain converges to the equilibrium distribution and then draw a sufficient number of samples to approximate the expected gradient with respect to the model (joint) distribution. Then, restart the process for the next step of approximate gradient ascent on the log-likelihood. However, this is prohibitively expensive.

The contrastive divergence (CD) method, instead, initializes the Markov chain with a training data used in the positive phase. The training data in the positive phase is likely to be already close to the (final) true distribution. Hence, even with a very short (one step) Gibbs chain, this method is believed to approximate the negative phase expectation. Although the samples drawn from very short Gibbs chain may be a poor representation of the model distribution, they are at least moving in the direction of the model distribution relative to the data distribution represented by the positive phase training data. Eventually, they may combine to produce a good estimate of the gradient. Note that CD with only 1 step of Gibbs sampling is practically used.

5 Materials for Further Study

This lecture note only covers very rapid way until 38 page of the following article.

– Yoshua Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

In order to obtain hierarchical feature representation, RBM needs to be stacked in layers via Deep Belief Network or Deep Boltzmann Machine. We skip all the description about auto-encoder and its variants.

The deep learning portal is on http://deeplearning.net, which provides software, tutorials, papers, and so forth. There are many excellent videos available on internet in which the researchers who contributed the progress in this area explain this area by their words.

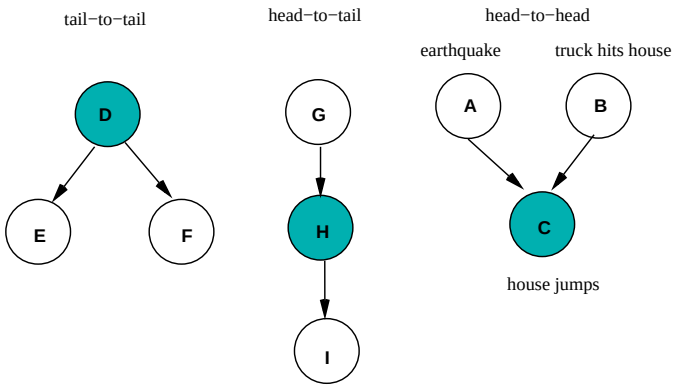


Fig. 9. Explain what is "explaining away"? Node D is tail-to-tail w.r.t the path from E to F (left). Node H is head-to-tail w.r.t. the path from G to I (middle). Node C is head-to-head w.r.t. the path from A to B (right). A tail-to-tail node or a head-to-tail node leaves a path unblocked unless it is observed in which case it blocks the path. In contrast, a head-to-head node blocks a path if it is unobserved, but once the node is observed the path becomes unblocked (This phenomenon is called "explaining away"). In other words, as a result of (C) (=observing the house jumps), event (A) (=earthquake) and (B) (=truck hits house) becomes dependent on each other.
.

- Yann LeCun (ICML 2013 tutorial) techtalks.tv
- Geoffrey Hinton (Google tech talk, coursera)
- Andrew Ng (UCLA IPAM summer school, coursera) / Christopher Manning (NAACL tutorial) techtalks.tv
- Yoshua Bengio youtube.com

Many video lectures are available in the following sites.

- youtube.com
- videolectures.net
- techtalks.tv
- UCLA IPAM summer school.
- googletechtalk
- coursera
- NIPS tutorials

There are many software for deep learning. Among these, Theano and deep learning tutorials are good start point to do excercise.

- RNN related software
  - rnnlib http://sourceforge.net/projects/rnnl/.
  - rnnlm toolkit http://rnnlm.org, word2vec https://code.google.com/p/word2vec/.
  - PyBrain http://pybrain.org.
- RBM related software

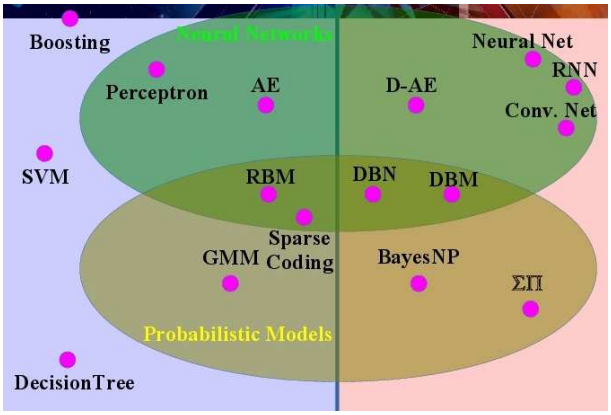


Fig. 10. Landscape of deep learning algorithms [LeCun]

- **Theano** http://deeplearning.net/software/theano.
- **Deep Learning Tutorials** http://deeplearning.net/tutorial.
- PyLearn2 https://github.com/lisa-lab/pylearn2.
- EBlearn http://koray.kavukcuoglu.org/.
- Torch http://torch.ch.

– Other software (auto-encoder, etc)
- Look at http://deeplearning.net/software_links.

References

Yoshua Bengio. 2009. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Christopher M. Bishop. 1995. Neural networks for patttern recognition. *Oxford University Press*.

Christopher M. Bishop. 2006. Pattern recognition and machine learning. *Springer*.

Geoffrey Hinton and Yee-Whye Teh. 2001. Discovering multiple constraints that are frequently approximately satisfied. *In Proceedings of Uncertainty in Artificial Intelligence (UAI 2001)*, pages 227–234.

Geoffry E. Hinton, J. L. McClelland, and D.E. Rumelhart. 1986. Distributed representations. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition(Edited by D.E. Rumelhart and J.L. McClelland) MIT Press*, 1.

Geoffrey E. Hinton. 2007. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8).

Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen netzen. *Diploma thesis (Institut fur Informatik Technische Universitat Munchen).*

Kevin P. Murphy. 2012. Machine learning: A probabilistic perspective. *MIT Press.*

B.A. Olshausen and D.J. Field. 1996. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609.

A.J. Robinson and F. Fallside. 1987. The utility driven dynamic error propagation network. *Technical report (Cambridge University)*, CUED/F-INFENG/TR1.

Schuster and Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–81.

Hava T. Siegelmann and Eduardo D. Sontag. 1991. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50:132–150.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. *Conference on Empirical Methods in Natural Language Processing.*

Background Knowledge of Neural Networks

1-of-K coding scheme This coding scheme encodes the original encoding into the binary vector with all elements equal to zero except one element. For example in multiclass classification problem of 5 class (=$\{a, b, c, d, e\}$, the class a is encoded as $a = [1, 0, 0, 0, 0]$, the class b as $[0, 1, 0, 0, 0]$, and so forth.

Activation Function

- linear activation function $\sigma(x) = x$
- binary threshold function $\sigma(x) = 1$ if $x > \theta$ (0 otherwise).
- logistic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$
- tanh function $\sigma(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- rectified linear function (linear threshold function) $\sigma(x) = x$ if $x > 0$ (0 otherwise).
- softmax output function $\sigma(x_i) = \frac{e^{x_i}}{\sum_{k'=1}^{K} e^{x_{k'}}}$

Nonlinearity It is essential to use nonlinear activation functions, such as tanh and logistic sigmoid, if the network as a whole need to capture nonlinearity.

Differentiability It is required to use differentiable activation function such as tanh and logistic sigmoid when you train the network with gradient descent methods.

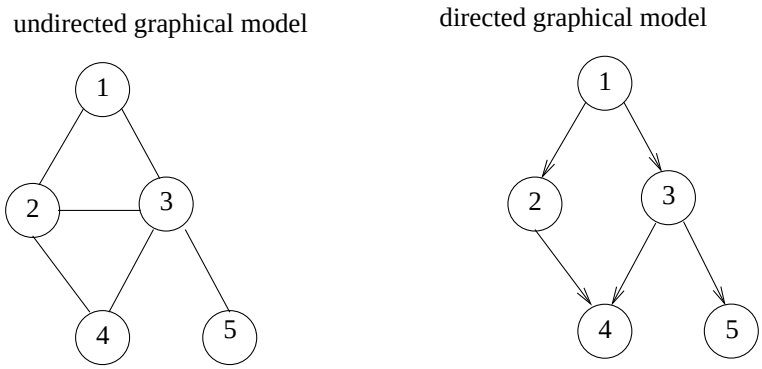


Fig. 11. The figure on the left shows undirected graphical model: this model has three maximal cliques $\{1,2,3\}$,$\{2,3,4\}$,$\{3,5\}$. The figure on the right shows directed graphical model where the number is in topological order where node 1 is the root.

Background Knowledge of Undirected Graphical Model

A clique is a subset of its vertices such that every two vertices in the subset are connected by an edge. For example in Figure 1, the following is an example of clique: $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{2,4\}$, $\{3,4\}$, $\{3,5\}$, $\{1,2,3\}$, and $\{2,3,4\}$. (If 1 and 4 were connected, $\{1,2,3,4\}$ would be also a clique; but it is not in this case). A maximal clique is a clique which cannot be made any larger without losing the clique property. In this case, maximal cliques are $\{1,2,3\}$,$\{2,3,4\}$, and $\{3,5\}$. The following Hammersley-Clifford theorem connects potential functions $\phi_c(y_c|\theta_c)$ for clique c with each maximal clique in the graph.

Definition 3 (Hammersley-Clifford). *A positive distribution $p(y) > 0$ satisfies the conditional independence properties of an undirected graph G iff p can be represented as a product of factors, one per maximal clique, i.e.,*

$$p(y|\theta) = \frac{1}{Z(\theta)} \prod_{c \in C} \phi_c(y_c|\theta_c) \tag{12}$$

where C is the set of all the (maximal) cliques of G, and $Z(\theta)$ is the partition function given by

$$Z(\theta) = \sum_x \prod_{c \in C} \phi_c(y_c|\theta_c) \tag{13}$$

Note that the partition function is what ensures the overall distribution sums to 1.

By this Hammersley-Clifford theorem, if p satisfies the conditional independence properties, the model which is drawn in the left in Figure 11 can be written

as follows

$$p(y|\theta) = \frac{1}{Z(\theta)}\phi_{123}(y_1, y_2, y_3)\phi_{234}(y_2, y_3, y_4)\phi_{35}(y_3, y_5) \tag{14}$$

where partition function $Z = \sum_y \phi_{123}(y_1, y_2, y_3)\phi_{234}(y_2, y_3, y_4)\phi_{35}(y_3, y_5)$.

By the connection with statistical physics, there is a model known as the Gibbs distribution:

$$p(y|\theta) = \frac{1}{Z(\theta)} \exp(-\sum_c E(y_c|\theta_c)) \tag{15}$$

where $E(y_c) > 0$ is the energy associated with the variables in clique c. This can be converted into undirected graphical model.

$$\phi_c(y_c|\theta_c) = \exp(-E(y_c|\theta_c)) \tag{16}$$

This undirected graphical model is called an energy-based model (energy-based probabilistic model): the high probability states correspond to the low energy configurations.

loss	formula	margin
NLL / MMI	$E(W, Y^i, X^i) + \frac{1}{\beta} log \int_{y \in \mathcal{Y}} e^{-E(W, y, X^i)}$	>0
energy loss	$E(W, Y^i, X^i)$	none
perceptron	$E(W, Y^i, X^i) - \min_{Y \in (Y)} E(W, Y, X^i)$	none
hinge	$\max(0, m + E(W, Y^i, X^i) - E(W, Y^i, X^i))$	m
log	$\log(1 + e^{E(W, Y^i, X^i) - E(W, Y^i, X^i)})$	>0
LVQ2	$\min(M, max(0, E(W, Y^i, X^i) - E(W, Y^i, X^i))$	0
MCE	$(1 + e^{E(W, Y^i, X^i) - E(W, Y^i, X^i)})^{-1}$	>0
square-square	$E(W, Y^i, X^i)^2$ - $(\max(0, E(W, Y^i, X^i) - E(W, Y^i, X^i)))^2$	m
square-exp	$E(W, Y^i, X^i)^2 + \beta e^{-E(W, Y^i, X^i)}$	>0
MEE	$1 - e^{-\beta E(W, Y^i, X^i)} / \int_{y \in \mathcal{Y}} e^{-E(W, y, X^i)}$	>0

Table 3. Table shows the overview of various loss functions. "NLL" means negative log-likelihood. [LeCun]