Unsupervised Learning in Neural Networks - Summary

Fabian Brix, Renato Kempter

December 26, 2013

1 Neuroscience

1.1 Visual Cortex

1.1.1 Primary Visual Cortex

The primary visual cortex is the simplest, earliest cortical visual area. It is highly specialized for processing information about static and moving objects and is excellent in pattern recognition.

1.2 Receptive Fields

The term receptive field originally described an area of the body service where a stimulus could elicit a reflex (Sherrington, 1906). The definition was later extended to sensory neurons defining the receptive field as a restricted region of visual space where a luminous stimulus could drive electrical responses in a retinal ganglion cell: *"Responses can be obtained in a given optic nerve fiber only upon illumination of a certain restricted region of the retina, termed the receptive field of the fiber"* (Hartline, 1938). $http://www.scholarpedia.org/article/Receptive_field$

General definition spanning different types of neurons across sensory modalities: the receptive field is a portion of sensory space that can elicit neuronal responses when stimulated. The sensory space can be defined in a single dimension (e.g. carbon chain length of an odorant), two dimensions (e.g. skin surface) or multiple dimensions (e.g. space, time and tuning properties of a visual receptive field). The neuronal response can be defined as FIRING RATE (i.e. **number of action potentials generated by a neuron**) or include also subthreshold activity (i.e. depolarizations and

hyperpolarizations in membrane potential that do not generate action potentials).

1.2.1 Visual Receptive Fields

2 Supervised vs. Unsupervised learning

Supervised learning a system is trained for regression or classification via a set of labeled training data. The goal is to optimize the parameters of an objective function using the labeled data (How to correctly predict the labelling "l" or "f" of input patterns according to the training data?). Supervised learning is dealt with in Prof. Seegers class "Pattern Classification and Machine Learning".

Unsupervised learning In unsupervised learning one tries to extract patterns from the data without having any information about their classification/outputs. One therefore tries to detect the intrinsic structure of the data. (How does an hand-written "l" or "f" look like?).

Reinforcement learning In this context an "agent" is employed and given a certain choice of actions and rewards upon chosing the correct action. The actions exist according to training input patterns (animal conditioning: a mouse sees an "l" or an "f" and chooses action a_1/a_2 accordingly and is given a reward upon success). Reinforcement learning can be either conducted in a supervised or an unsupervised manner.

3 Hebbian Learning Rule

change in a given synaptic weight is proportional to both the pre-synaptic input and the output activity of the post-synaptic neuron.

- time-dependent
- local
- strongly interactive

3.1 Synaptic plasticity

change in connection strength

> Hebb, 1949: *"When an axon of cell j repeatedly or persistently takes part in firing (through synapses to dendrites of) cell i, then j's efficiency as one of the cells firing is increased".*

The Hebbian learning rule is a local UNSUPERVISED LEARNING rule, because it depends solely on the states of the respective pre-synaptic i and post-synaptic j neurons and the present efficacy w_{ij}, but not on the state of other neurons k. The repsective pre-synaptic neurons and post-synaptic neuron can however be simultaneously active, so that correlations in their activity lead to firing of the neuronal cell i. In Hebbian learning these correlations are used to learn the synaptic transmission efficacies w_{ij}, so that in essence it is correlation-based learning.

3.2 Homeostatic plasticity

Homeostatic plasticity refers to the capacity of neurons to regulate their own excitability relative to network activity, a compensatory adjustment that occurs over the timescale of days. The term homeostatic plasticity derives from two opposing concepts: "homeostatic" (a product of the Greek words for "same" and "state" or "condition") and plasticity (or "change"), thus homeostatic plasticity means "staying the same through change."

3.3 Rate-based Hebbian Learning

Goal is to find a mathematically formulated learning rule based on Hebb's postulate.
We begin by focusing on a single synapse with efficacy w_{ij} transmitting signals from a pre-synaptic neuron i to a post-synaptic neuron j. The activity ((mean) firing rate - average number of spikes per unit time) of i is denoted by v_i and that of j by v_j. Based on the (1) `locality` aspect of Hebbian Learning we can write a general description of the change of the synaptic efficacy.

$$\frac{d}{dt}w_{ij} = \mathbf{F}(w_{ij}; v_j^{pre}, v_i^{post})$$

The membrane potential is further uniquely defined by the postsynaptic firing rate $v_i = g(u_i)$ with a monotone gain function g and therefore doesn't have to be included in F.
The (2) `cooperativity` aspect of Hebb's postulates implies that pre- and postsynaptic neuron have to be active simultaneously for a synaptic weight change to occur and this property can be used to learn something about the

function F. Taylor series expansion around $v_i = v_j = 0$ leads to:

$$\frac{d}{dt}w_{ij} = c_0(w_{ij}) + c_1^{post}(w_{ij})v_i + c_1^{pre}(w_ij)v_j + c_2^{pre}(w_{ij})v_j^2 + c_2^{post}(w_{ij})v_i^2 + c_2^{corr}v_iv_j + \mathcal{O}(v^3)$$

The term with c_2^{corr} is bilinear in pre- and postsynaptic activity showing that the change of the synaptic efficacy w_{ij} is indeed subject to a combination of changes in both activities.
Simplest choice of F is to fix $c_2^{corr} = c > 0$ and set all other terms of the taylor expansion to zero resulting in the prototype of hebbian learning.

$$\frac{d}{dt}w_{ij} = c_2^{corr}v_iv_j$$

This learning rule includes only first-order terms and therefore models non-Hebbian plasticity. More complicated learning rules include higher-order terms of the taylor expansion. If $c_2^{corr} = c < 0$, the learning rule is anti-hebbian.
F is dependet on the efficacy w_{ij}, so that it will not grow without limit. A saturation of synaptic weights can be achieved if parameter c_2^{corr} goes to zero as w_{ij} approaches its maximum value:

$$c_2^{corr}(w_{ij}) = \gamma_2(1 - w_{ij})$$

Originally Hebb did not consider a rule for decreasing the synaptic weights. In order to have a feasible model however, the synaptic efficacy should decrease in the absence of stimulation.

$$c_0(w_{ij}) = -\gamma_0 w_{ij}$$

The combination results in the simplest feasible learning rule:

$$\frac{d}{dt}w_{ij} = \gamma_2(1 - w_{ij})v_iv_j - \gamma_0 w_{ij}$$

3.3.1 Gating

Postsynaptic gating A weight change occurs only if the postsynaptic neuron is active, $v_i > 0$. $\Rightarrow$ the weight changes are "gated" by the postsynaptic neuron. Only the efficacies of highly active presynaptic neurons $v_j > v_\phi$ are strengthened.

$$\frac{d}{dt}w_{ij} = \gamma v_i(v_j - v_\phi(w_{ij})), \gamma > 0$$

Presynaptic gating role of pre- and postsynaptic firing rate are exchanged. Now, a change in synaptic weights can only occur if the presynaptic neuron is active, $v_j > 0$ and the activity of the postsynaptic neuron determines the direction of the change.

$$\frac{d}{dt}w_{ij} = \gamma v_j(v_i - v_\phi))$$

3.3.2 BCM rule

The "Bienenstock-Cooper-Munroe" rule is a generalization of the presynaptic gating rule, where Φ is a non-linear function and the reference rate v_ϕ is the running average of the postsynaptic activity v_i.

$$\frac{d}{dt}w_{ij} = \eta\Phi(v_i - v_\phi))v_j - \gamma w_{ij}$$

3.4 Learning in Rate Models (functional consequences of hebbian learning)

Goal: understand how activity-dependent learning rules influence the formation of connections between neurons in the brain.
Plasticity is controlled by the statistical properties of the presynaptic input for the postsynaptic neuron.

3.4.1 Evolution of synaptic weights

For the sake of simplicity we model the presynaptic input as a set of static patterns $\{x^\mu \in \mathcal{R}^N; 0 \leq \mu \leq p\}$. At each time step one of the patterns is selected at random and the presynaptic rates are fixed to $v_i = x_i^\mu$. The synaptic weights are modified according to a Hebbian learning rule dependent on the correlation of pre- and postsynaptic activity: $\frac{d}{dt}w_{ij} = a_2^{corr}v_i^{post}v_j^{pre}$.
In a general rate model the firing rate v_i^{post} is given by a nonlinear function of the total input: $v^{post} = g(\sum_i w_i v_i^{pre})$. For simplification reasons we focus on a linear rate model:

$$v_i^{post} = \sum_k w_{ik}v_k^{pre}$$

We can now combine the learning rule and the linear rate model to form:

$$\frac{d}{dt}w_{ij} = a_2^{corr}\sum_k w_{ik}v_k^{pre}v_j^{pre}$$

We are interested in the long-term behaviour of the synaptic weights and therefore consider the expectation value of the weight vector, i.e., the weight vector averaged over the sequence $(x^{\mu_1}, x^{\mu_2}, ..., x^{\mu_n})$ that have so far been presented to the network.

$$\langle \frac{d}{dt} w_{ij} \rangle = a_2^{corr} \sum_k \langle w_{ik} \rangle \underbrace{\langle x_k^\mu x_j^\mu \rangle}_{C_{kj}} \qquad C_{kj} = \langle x_k x_j \rangle = \frac{1}{p} \sum_{\mu=1}^{p} x_k^\mu x_j^\mu$$

C_{kj} are the correlation matrices for respective components i and j in the sum of the learning rule. Using PCA we can detect the direction of maximal variance in the set of patterns. $\Rightarrow$ Express weight vector w in eigenvectors of C. What do principal components mean in this respect?

PCA think of neuronal output as projection of weight vector, fill in PCA equations

3.5 Oja's rule

The Oja learning rule is a mathematical formalization of the Hebbian learning rule, such that over time the neuron actually learns to compute a principal component of its input stream. The forgetting term added to balance the growth of the weights this time is not only proportional to the value of the weight, but also to the square of the output of the neuron v_i.

$$\frac{d}{dt} w_{ij} = a_2^{corr} v_j v_i - \underbrace{a_2^{post}}_{a_2^{corr} w_{ij}} (v_i^{post})^2 = a_2^{corr} (v_j v_i - w_{ij} (v_i^{post})^2)$$

Now, the forgetting term balances the growth of the weight. The squared output $(v_i^{post})^2$ guarantees that the larger the output of the neuron becomes, the stronger is this balancing effect.

3.5.1 Oja's rule and PCA

$$v_i^{post} = \mathbf{w}^T \mathbf{x}^\mu$$

$$\frac{d}{dt} \mathbf{w} = a_2^{corr} (\mathbf{x}^\mu \mathbf{x}^{\mu T} \mathbf{w} - \mathbf{w}^T \mathbf{x}^\mu \mathbf{x}^{\mu T} \mathbf{w} \mathbf{w}), \{x^\mu, 0 \leq \mu \leq p\}$$

This is the incremental change for just one input vector x^μ. When the algorithm is run for a long time, changing the input vector at every step, one can look at the average behaviour. An especially interesting question

is what is the value of the weights when the average change in the weight is zero. This is the point of convergence of the algorithm. Averaging right hand side over x^μ while w stays constant yields the following equation.

$$\frac{d}{dt}\mathbf{w} = a_2^{corr}(\underbrace{\langle \mathbf{x}\mathbf{x}^\mathbf{T} \rangle}_{C}\mathbf{w} - \underbrace{(\mathbf{w}^T \underbrace{\langle \mathbf{x}\mathbf{x}^\mathbf{T} \rangle}_{C}\mathbf{w})}_{\lambda}\mathbf{w}) = 0, \overline{x} = 0$$

Considering that the quadratic form $w^T C w$ is a scalar, this equation clearly is the eigenvalue-eigenvector equation for the covariance matrix C. This analysis shows that if the weights converge in the Oja learning rule $C\mathbf{w} = \lambda\mathbf{w}$, then the weight vector becomes one of the eigenvectors of the input covariance matrix $\mathbf{w} = \mathbf{e_1}$ (ONLY THIS STATE IS STABLE), and the output of the neuron becomes the corresponding principal component. Principal components are defined as the inner products between the eigenvectors and the input vectors. For this reason, the simple neuron learning by the Oja rule becomes a principal component analyzer (PCA). Although not shown here, it has been proven that the neuron will find the the `first principal component` and the norm of the weight vector tends to one.

3.6 Independent Component Analysis (ICA)

Goal: Compute a transformation such that the entries of the transformed vector (components) are mutually independent. Statistical Independence stronger than uncorrelatedness required by the PCA. (Two conditions are equivalent *only* for Gaussian random variables).

Assumption: Random input vector $\mathbf{x}$ is generated by a linear combination of statistically independent $p(x_1, \ldots, x_n) = p_1(x_1) \ldots p_n(x_n)$ and stationary components (sources), $\mathbf{x} = A\mathbf{y}$. The task is now is to find a matrix W so as to recover the components of $\mathbf{y}$, the original signal sources $\mathbf{y} = \begin{bmatrix} s_1 & s_2 & \ldots & s_n \end{bmatrix}^T$ by exploiting the information hidden in many samples of the mixed signal patterns $\mathbf{x}$ (blind source separation). $\mathcal{A}$ is known as the mixing and W as the demixing matrix, respectively.
In contrast to PCA which can always be performed, ICA is meaningful only if the involved random variables are non-Gaussian. Each of the resulting independent components is `uniquely` estimated up to a multiplicative constant. For this reason the components are considered to be of unit variance.

Preprocessing (whitening)

- Zero mean data
- Whitening (unit variance). Perform PCA on mixed signals and divide each component by the square-root of its eigenvalue $x_k^n = \frac{1}{\sqrt{\lambda_k}} x_k \Rightarrow C^n = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. After PCA independence of projections $p(y_1, \dots y_n) = p_1(y_1) \dots p_n(y_n)$

This approach in performing ICA is a direct generalization of the PCA technique. The Karhunen-Love transform focuses on the second-order statistics and demands the cross-correlations $\langle y_i y_j \rangle$ to be zero. In ICA demanding that the components of $\mathbf{y}$ be statistically independent is equivalent to demanding all the higher order cross-cumulants to be zero (going up to the fourth-order cumulants is sufficiant for many applications).

If we look at two-dimensional data, an arbitrary projection will be close to a Gaussian distribution which doesn't tell us anything about our data (neuronal inputs). $\Rightarrow$ search for direction which is maximally non-Gaussian. Searching for independent rather than uncorrelated features gives us the means of exploiting a lot more information, hidden in the higher order statistics of the data. Independent component analysis =¿ independence $p(y_1, y_2) = p(y_1)p(y_2)$. Independence of patterns implies them being uncorrelated? How to find natural axis?

3.6.1 Gaussianity vs. non-gaussianity

We need a measure for non-gaussianity which we will obtain by looking at higher-order statistics. Normalized gaussian distribution after PCA has no preferred axis.

3.6.2 FastICA

4 Competitive Learning

4.1 Clustering

4.2 Neuronal Version of Clustering

Kohonen learning rule

5 Reinforcement learning

RL vs. supervised learning Supervised learning, used in statistical pattern recognition and artifical neural networks is learning from examples provided by a knowledgable external supervisor whereas in RL an agent is instructed to learn from its own experience. It is a "behavioral" learning problem: Through interaction between the learning system and its environment, whereas the system seeks to achieve a specific goal.

trade-off exploration vs. exploitation An agent has to exploit what it ALREADY KNOWS in order to obtain a reward, but it also has to EXPLORE in order to make BETTER ACTION SELECTIONS in the future. The agent must try a variety of actions and progressively favour those that appear to be best, because neither schemes can be pursued exclusively without failing a the task. Estimate reward probabilities vs. take action to maximize reward.

RLs main characteristic RL explicitly considers the WHOLE problem of a goal-directed agent INTERACTING with an uncertain environment. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. Moreover, it is usually assumed from the beginning that the agent has to operate despite significant UNCERTAINTY ABOUT THE ENVIRONMENT it faces. When reinforcement learning involves PLANNING, it has to address the *interplay between planning and real-time action selection*, as well as the question of how environmental models are acquired and improved.
An agent's actions are permitted to affect the future state of the environment (e.g., the next chess position, the level of reservoirs of the refinery, the next location of the robot), thereby affecting the options and opportunities available to the agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, and thus may require foresight or planning. The knowledge the agent brings to the task at the start–either from previous experience with related tasks or built into it by design or evolution–influences what is useful or easy to learn, but interaction with the environment is essential for adjusting behavior to exploit specific features of the task.

5.1 Elements of reinforcement learning

there are four main subelements of a reinforcement learning system.

Policy

commonly denoted by π, it defines the learning agents way of behaving at a given time. Mapping from the perceived states of the environment to the actions that present themselves to the agent in these states. The policy alone is sufficient to determine the behaviour of an agent. In oder words, a policy is a rule used by the learning system to decide what to do, given the current state of the environment.

Reward function

defines the goal in a reinforcement learning problem. Maps each perceived state of the environment to a single number, a reward, which describes the intrinsic desirability of the state. The agent's sole objective is to maximize its total gain from the rewards it receives.
The reward function may serve as a basis for altering the agent's policy.

Value function

While the reward function indicates which actions are immediately beneficial to the agent, a **value function** specifies what is good for the agent in the long run. The value of a state is the EXPECTED total amount of rewards the agent will accumulate from this state onwards. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards.

Model of the environment

A model mimics the behaviour of the environment. *For example, given a state and action, the model might predict the resultant next state and next reward.* Models are used for PLANNING, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively NEW DEVELOPMENT. Early reinforcement learning systems were explicitly trial-and-error learners; what they did was viewed as almost the opposite of planning. Nevertheless, it gradually became clear that reinforcement learning methods are closely related to dynamic programming methods, which do use models, and that they in turn are closely related to state-space planning methods.

5.2 Evolutionary methods

Genetic algorithms, genetic programming, simulated annealing, and other function optimization methods have also been used to solve reinforcement learning problems. These methods search directly in the space of policies without ever appealing to value functions. We call these **evolutionary methods** because their operation is analogous to the way biological evolution produces organisms with skilled behaviour even when they do not learn during their individual lifetimes (organisms that have chosen adaptive policies survive). Effective when space of policies is sufficiently small or accurately sensing the state of the environment is impossible.

5.3 Reward-based action learning: Q-values

$Q(S,a)$ is the expected reward and can be written as $\sum_{s'} P^a_{s\overrightarrow{s'}} R^a_{s\overrightarrow{s'}}$. The optimal strategy is to take the action such that $Q(s,a^*) > Q(s,a_j)$. Q values are not known at the beginning, therefore, we need to learn them -¿ iteratively update. $\Delta Q(s,a) = \theta[r - Q(s,a)]$. If the mean update has converged, then $< \Delta Q(s,a) >= 0$

There are different strategies:

- Greedy strategy
- η - greedy strategy $P = 1.\eta$
- Softmax strategy $P(a') = \frac{exp[\beta Q(a')]}{\sum_a exp[\beta Q(a)]}$
- Optimistic greedy (Q values too big at start)

5.4 Bellmann Equation and Sarsa

Multi step horizon reward-based action learning!

$$\Delta Q(s,a) = \eta[r - (Q(s,a) - \gamma Q(s',a'))]$$

We can use the Bellman equation to write $Q(s,a)$

$$Q(s,a) = \sum_{s'} P^a_{s\to s'} * [R_{s\to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a')]$$

The Bellman equation can be used for the SARSA algorithm:

1. Being in state s, chose action a according to policy

2. Observe reward r, next state s'

3. Choose action a' in state s' according to policy

4. Update $\Delta Q(s, a)$

5. s' -¿ s; a' -¿ a

6. Goto 1)

We can rewrite the Bellman equation and express it as State values V.

$$V^{\pi}(s) = \sum_{a} \pi(s, a) \sum_{s'} P^{a}_{s \overrightarrow{s}'} [R^{a}_{s \overrightarrow{s}'} + \gamma * V^{\pi}(s')]$$

which we can write in form of the value function as:

$$V^{\pi}(s) = \sum_{a} \pi(s, a) \sum_{s'} P^{a}_{s \to s'} [R^{a}_{s \to s'} + \gamma V^{\pi}(s')]$$

The difference between Q and V is the following: The value function is the expected return starting from state s_t and following policy π. The state-action value function $Q(s, a)$ is the expected return starting from state s, taking action a and thereafter following policy π.

5.5 On-policy vs off-policy learning

On-policy means SARSA; off-policy means Q-learning. The difference lies in the method how to compute $Q(s, a)$. While for SARSA, we use the policy for the discounted term $\gamma \sum_{a'} \pi(s', a') Q(s', a')$, the Q-learning algo simply does $\gamma * max_{a'} Q(s', a')$. Therefore, Q-learning uses greedy for update and the policy for action selection (η – greedy)

5.6 Eligibility traces

At the moment of the reward update, update also previous action values along trajectories. The problem is that learning is very slow and the information diffusion across several states takes time.

γ is the memory reduction parameter. Every time an action a is taken, one should do the following:

- If a = action taken in state j: $e_{aj}(t) = \gamma * \lambda * e(s, a) + 1$
- else: $e(s, a)(t) = \gamma \lambda e(s, a)$

Then, for all states s and actions a:

$$\Delta Q(s,a) = \eta \delta_t e(s,a)$$

An eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action (backward view). The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. The more theoretical view is that they are a bridge from TD to Monte Carlo methods.

5.7 Continuous states

$$Q(s,a) = \sum_j w_{aj} * r_j(s) = \sum_j w_{aj} * \phi(s - s_j)$$

If the time interval between subsequent weight updates approaches zero, the difference between two consecutive states $s' - s = v(a) * \Delta t$, where $v(a)$ is the velocity resulting from the choice of the action a. $s' - s$ approximates zero, therefore $s' \approx s$. Similarly, the finer the time resolution becomes, the likelier is that $a \approx a'$. Thus, for small time steps, an online gradient descent on the erro term E_t becomes closer to a δ_t modulated Hebbian rule.

5.8 Temporal Difference TD(λ)

TD incorporates eligibility and continuous state space. We do a function approximation using weights and instead of updating $Q(s,a)$. In general, we do not work with $Q(s,a)$ but rather with the $V(s)$ values.

$$Q_w(s,a) = \sum_{i=1}^{n} w_i^a * r_i(s)$$

$$\Delta w_{aj} = \eta \delta_t e_{aj}$$

TODO: DOES SOMEBODY HAVE THE BLACKBOARD: REPRESENTATION OF STATE-VALUE Lecture 7 (Backgammon)

We can write the eligibility trace as $e_{aj}(t) = \gamma \lambda e_{aj}(t - \Delta t) + r_j \delta_{a,a'}$

6 Energy-Based Models (EBM)

A scalar energy is associated to each configuration of the variables of interest in the model (cost function). Learning then corresponds to modifying the energy function so that its shape has desirable properties, for example we

would like desirable configurations to have low energy. The probabilistic distribution of for the model is define by an energy function:

$$p(x) = \frac{\exp(-E(x))}{Z}$$

This distribution is the Boltzmann or Gibbs distribution which originates from classical statistical mechanics.
The normalization factor Z is called the partition function $Z = \sum_x \exp(-E(x))$.
An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data.

6.1 EBMs with hidden units

Often one does not observe the examples x fully or one wants to introduce some non-observed variables to increase the expressive power of the model. So we consider a hidden part y additionally to the observed part x, giving:

$$P(x) = \sum_y P(x, y) = \sum_y \frac{\exp(-E(x, y))}{Z}$$

By introducing the notation of FREE ENERGY

$$\mathcal{F} = -\log \sum_y \exp(-\mathcal{F})$$

we can the rewrite the probability distribution in the condensed of the gibbs distribution:

$$P(x) = \frac{exp(-\mathcal{F}(x))}{Z}, with Z = \sum_x \exp(-\mathcal{F}(x))$$

6.2 Restricted Boltzmann Machines (RBM)

Boltzmann Machines (BMs) are a particular form of log-linear (Gibbs distribution) Markov Random Fields (MRF), where the energy function is linear in its free parameters. In order to represent complicated distributions (limited parametric setting $\rightarrow$ non-parametric setting) we consider some of the variables to be hidden (not observed). One can increase the modeling capacity of the Boltzmann Machine (BM) by adding for hidden variables/units. Furthermore a RBM is restricted to a bipartite Graph, because there no

lateral connections visible-visible and hidden-hidden are allowed. We introduce the notation of $v = x$ for visible units and $h = y$ for hidden units. The energy function therefore is defined as:

$$E(v, h) = -b^t v - c^t h - h^t W_{u,v} u$$

where $W_{u,v}$ represents a connection matrix of weights for hidden and visible units and b, c are the offsets of the visible and hidden layers respectively. By plugging in the energy function we get the following representation of the free energy formula:

$$F(v) = -b^t v - \sum_i \log \sum_{h_i} \exp(h_i(c_i + W_i v))$$

Because of the specific structure of RBMs, visible and hidden units are conditionally independent of one-another

A particular set of "synaptic" weights is said to constitute a perfect model of the environmental structure if it leads to exactly the same probability distribution of the states of the visible units as when these units are clamped by the environmental input vectors.

RBMs with binary units

Commonly RBMs are studied with binary units $v_i, h_i \in \{0, 1\}$ ... The free energy simplifies to:

$$\mathcal{F}(v) = -b^t v - \sum_i \log(1 + \exp(c_i + W_i v))$$

deeplearning.net/tutorial/rbm.html
sigmoid function (sigm):

$$\sigma(t) = sig(t) = \frac{1}{1 + \exp(-t)} = \frac{1}{2} \cdot (1 + \tanh(\frac{t}{2}))$$