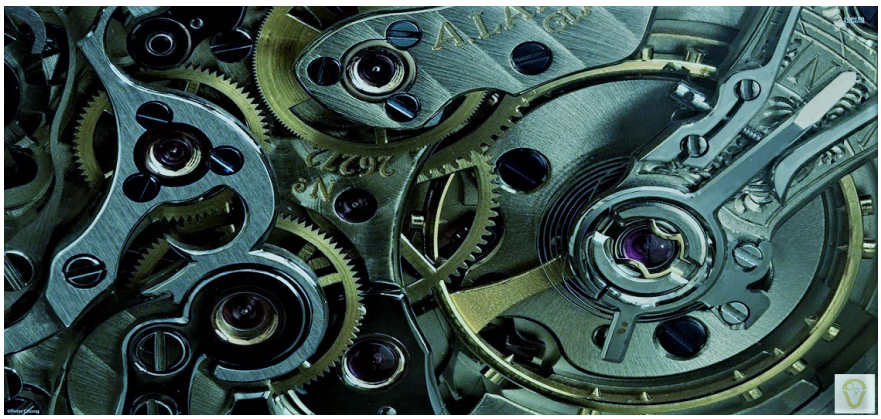
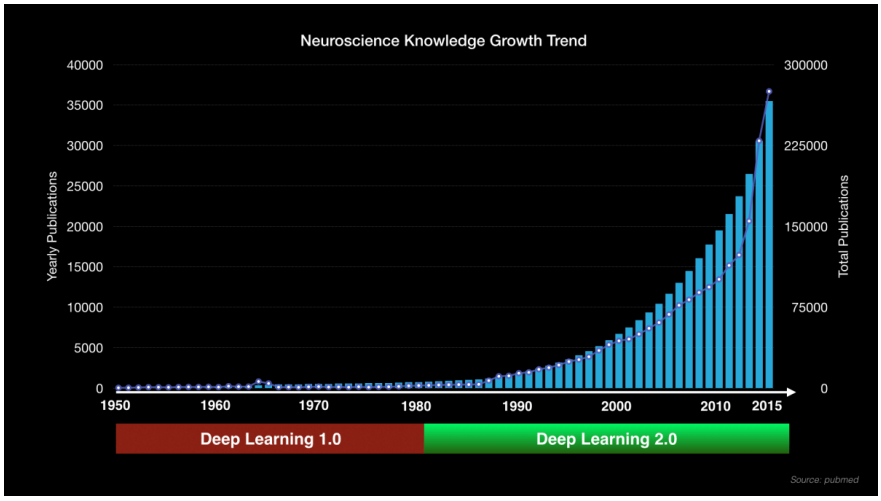
Alok Jha Follow
Cognitive Computing Consultant
Apr 20 · 13 min read

Boltzmann Machines | Transformation of Unsupervised Deep Learning—Part 1



Unlike task-specific algorithms, Deep Learning is a part of Machine Learning family based on learning data representations. With massive amounts of computational power, machines can now recognize objects and translate speech in real time, enabling a smart Artificial intelligence in systems. The concept of a software simulating the neocortex's large array of neurons in an artificial *neural network* is decades old, and it has led to as many disappointments as breakthroughs. But because of improvements in mathematical formulas and increasingly powerful computers, today researchers & data scientists can model many more layers of virtual neurons than ever before. As Peter Lee of Microsoft Research once said —

> **"Recent improvements in Deep Learning has reignited some of the grand challenges in Artificial Intelligence."**



Noticeable upward trend of Deep Learning from 1990's

Languishing through the 1970's, early neural networks could simulate only a very limited number of neurons at once, so they could not recognize patterns of great complexity. In the mid 1980's, Geoffrey Hinton and others helped spark an amelioration in neural networks with so-called *deep models* that made better use of many layers of software neurons. But the technique still required heavy human involvement as programmers had to label data before feeding it to the network and complex speech/image recognition required more computer power than was then available. Just to have a feel of requirements against cost, look at the representation below:

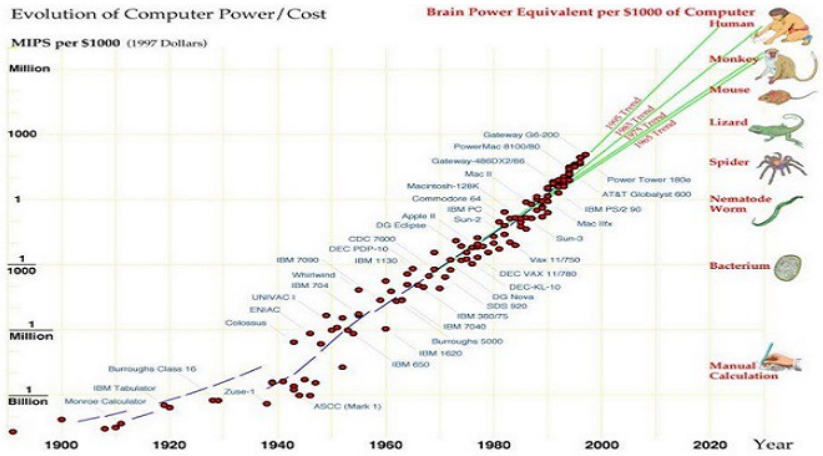


Image Source (I am not that gifted to present such a nice representation)

However in 2006, Hinton developed a more efficient way to teach individual layers of neurons where the first layer learns primitive features, like an edge in an image or the tiniest unit of speech sound by

finding combinations of digitized pixels or sound waves that occur more often than they should by chance. Once that layer accurately recognizes those features, they're fed to the next layer, which trains itself to recognize more complex features, like a corner or a combination of speech sounds. The process is repeated in successive layers until the system can reliably recognize phonemes or objects and this is what forms the base of Supervised Deep Learning models like Artificial/Convolutional /Recurrent Neural Networks.

Even prior to it, Hinton along with Terry Sejnowski in 1985 invented an Unsupervised Deep Learning model, named **Boltzmann Machine**. This model is based on *Boltzmann Distribution* (also known as *Gibbs Distribution*) which is an integral part of Statistical Mechanics and helps us to understand impact of parameters like Entropy and Temperature on Quantum States in Thermodynamics.

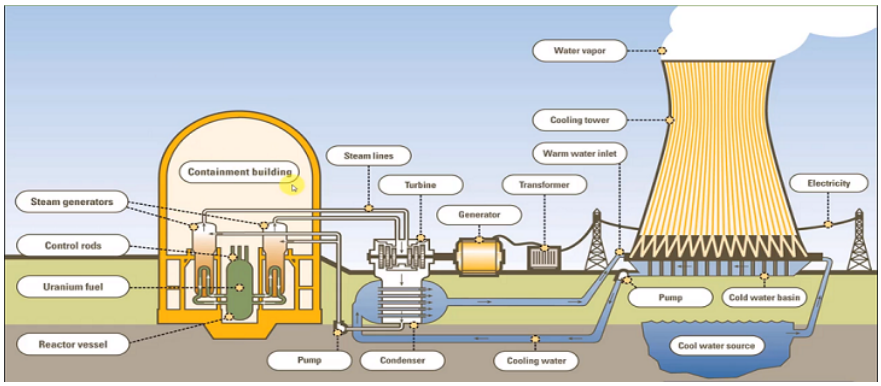
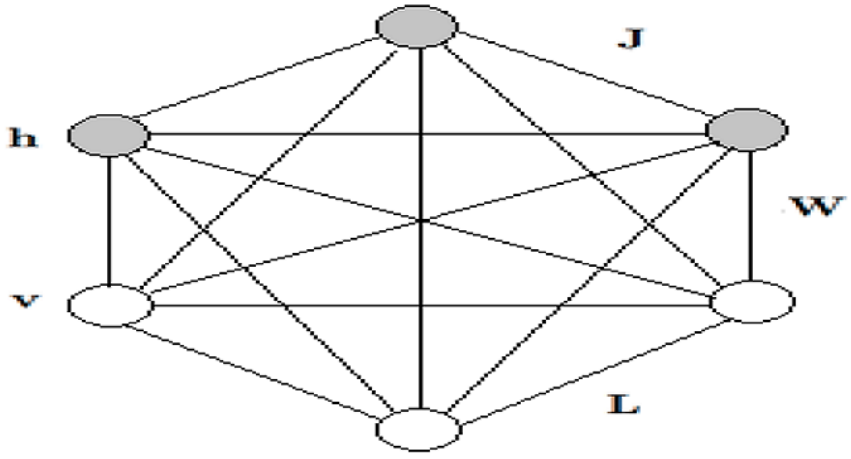


Image Source for a simplified version of Nuclear Power Plant parameters

Hinton once referred to illustration of a Nuclear Power plant as an example for understanding Boltzmann Machines. This is a complex topic so we shall proceed slowly to understand intuition behind each concept, with minimum amount of mathematics and physics involved. So in simplest introductory terms, Boltzmann Machines are primarily divided into two categories: **Energy-based Models (EBMs)** and **Restricted Boltzmann Machines (RBM)**. When these RBMs are stacked on top of each other, they are known as **Deep Belief Networks (DBN)**. These DBNs are further sub-divided into **Greedy Layer-Wise Training** and **Wake-Sleep Algorithm**. There is also another type of Boltzmann Machine, known as **Deep Boltzmann Machines (DBM)**. Unless we're involved with complex AI research work, ideally stacked RBMs are more than enough for us to know, and that gets taught in all

the Deep Learning MOOCs. The most common use-case for RBMs are Advanced Recommender Systems so if you preparing for an interview in companies like AirBnB, Amazon, eBay and Netflix, then it is time to get extra attentive.

Instead of specific model, let us begin with layman **understanding of general functioning in a Boltzmann Machine** as our preliminary goal. *A Boltzmann Machine is a stochastic (non-deterministic) or Generative Deep Learning model which only has Visible (Input) and Hidden nodes.* You got that right! There is no Output node in this model hence like our other classifiers, we cannot make this model learn **1** or **0** from the Target variable of training dataset after applying Stochastic Gradient Descent (SGD), etc. Exactly similar case with our regressor models as well where it cannot learn the pattern from Target variables. These attributes make the model non-deterministic. Thinking of how does this model then learn and predict, is that intriguing enough? To break the ice, kindly allow me to explain functioning of Boltzmann Machines.



Boltzmann System Inter-connections (Sorry for this blurry image!)

Image presents six nodes in it and all of them are inter-connected, and are also often referred to as *States*. Grey ones represent Hidden nodes (**h**)and white ones are for Visible nodes (**v**). Flashback in your own medial temporal lobe shall tell you that A/C/R Neural networks never had their Input nodes connected, whereas Boltzmann Machines have their inputs connected & that is what makes them fundamentally different. All these nodes exchange information among themselves and self-generate subsequent data, hence termed as **Generative deep**

model. Here, Visible nodes are what we measure and Hidden nodes are what we don't measure. When we input data, these nodes learn all the parameters, their patterns and correlation between those on their own and forms an efficient system, hence Boltzmann Machine is termed as an Unsupervised Deep Learning model. This model then gets ready to monitor and study abnormal behavior depending on what it has learnt. This model is also often considered as a counterpart of Hopfield Network, which are composed of binary threshold units with recurrent connections between them.

. . .

Energy-based Models (EBMs): The main purpose of statistical modeling and machine learning is to encode dependencies between variables. EBMs capture dependencies between variables by associating a scalar energy to each configuration of the variables. *Inference* consists of clamping the value of observed variables and finding configurations of the remaining variables that minimize the energy. *Learning* consists of finding an energy function in which observed configurations of the variables are given lower energies than unobserved ones. EBMs can be seen as an alternative to probabilistic estimation for prediction, classification, or decision-making tasks because there is no requirement for proper normalization. We discussed Thermodynamics, poked your medial lobes, compared models to ANN/CNN/RNN and still no mathematical formula on our screen. So just to ensure that we're still in business, kindly allow me to paste a formula snippet and let us remember it in simple terms as **Boltzmann Distribution and Probability**:

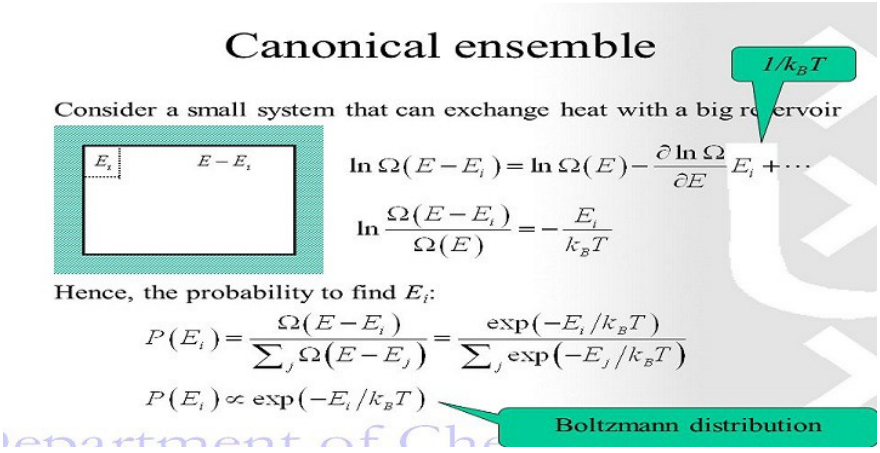


Image Source: Boltzmann Distribution (Swear to make it hunky dory!)

I know you might be thinking if I really had to deal with these, I would have chosen Ph.D instead of reading your blog post. But what if I make this cooler than your Xbox or PlayStation? Let us imagine an air-tight room with just 3–4 people in it. Ignoring the possibility of ghosts, what else can we think of to be present in this room apart from these people? You're right! The air (gas molecules) and the interesting part that we know is that these gas molecules are evenly spread out in the room. Now, think for a minute why these molecules are evenly spread out and not present in any corner of their choice, (which ideally is statistically feasible)? This is what got (conceptually)explained with **Boltzmann Distribution**, where it justifies an extremely low probability of such a cornering as that would enormously increase the energy of gas molecules due to their enhanced movement. And just like us, even these gas molecules prefer to be normal instead of wandering in space like supercharged *The Hulk*. So, we understand that at equilibrium the distribution of particles only depend on the energy difference between the *states* (or, micro-states).

Above equation is what we use in sampling distribution memory for a Boltzmann Machine. Focusing on the equation now, **P** stands for *Probability*, **E**for *Energy* (in respective states, like Open or Closed), **T** stands for *Time*, **k** is your homework and **summation** & **exponents** symbol stand for '*please google for closest to your house high-school*' (*kidding!*). Thus for a system at temperature **T**, the probability of a state with energy, **E** is given by the above distribution reflecting inverse correlation with higher the energy of a state, lower the probability of that state. Energy is defined through the weights of the synapses, and once the system is trained with set weights, then system keeps on searching for lowest energy state for itself by self-adjusting. EBMs for sequence labeling and structured outputs can be further sub-divided into 3 categories: > Linear Graph-based (CRF, SVMM, & MMMN)> Non-Linear Graph-based > Hierarchical Graph based EBMs. **Conditional Random Fields (CRF)** use the negative log-likelihood loss function to train a linear structured model. **Max-Margin Markov Networks (MMMN)** uses a margin loss to train the linearly parameterized factor graph with energy function, and can be optimized with Stochastic Gradient Descent (SGD). **Support Vector Markov Models (SVMM)** aims to derive a maximum margin formulation for the joint kernel learning setting.

Learning in EBM: Utmost critical question that affects the efficiency of learning is: “How many energies of incorrect answers must be explicitly pulled up before the energy surface takes the right shape?”. Energy-based loss functions that pull up the most offending incorrect answer only pull up on a single energy at each learning iteration. By contrast, the negative log-likelihood loss pulls up on all incorrect answers at each iteration, including those that are unlikely to produce a lower energy than the correct answer. An important open question is whether alternative loss functions exist whose contrastive term and its derivative are considerably simpler to compute than that of the negative log-likelihood loss, while preserving the nice property that they pull up a large volume of incorrect answers whose energies are *threateningly low*. For models in the *intractable* category, each individual energy that needs to be pulled up or pushed down requires an evaluation of the energy and of its gradient (if a gradient-based optimization method is used). Hence, finding parameterizations of the energy surface that will cause the energy surface to take the right shape with the minimum amount of pushing of pulling is of crucial importance. More ‘*rigid*’ energy surfaces may take a suitable shape with less pulling, but are less likely to approach the correct shape. There seems to be a bias-variance dilemma similar to the one that influences the generalization performance. Very often, the inference algorithm can only give us an approximate answer, or is not guaranteed to give us the global minimum of the energy.

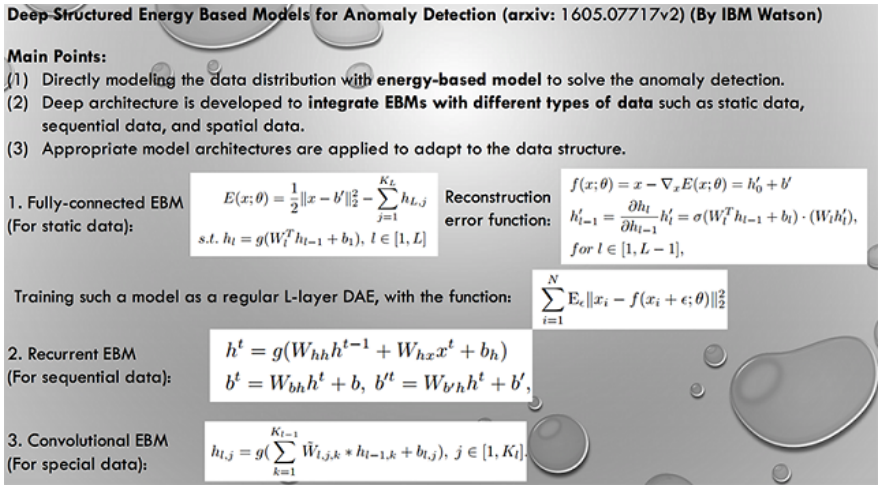


Image Source

Important take-away from EBM can be: > Many existing learning models can be simply be expressed in energy-based learning

framework. > Among various loss functions, some are good (with non-zero margin), and some can be bad. > Probabilistic learning is a special case of energy-based learning where loss function is the negative log-likelihood, a.k.a. the maximum mutual information criterion. > Optimizing the loss function with stochastic gradient methods is often more efficient than black-box convex optimization methods. > Stochastic gradient methods can be applied to any loss function including non-convex ones. Local minima are rarely a problem in practice because of high dimensionality of the space. > SVMM, MMMN, and CRFs are all sequence modeling systems that use linearly parameterized energy factors. >Graph transformer networks are hierarchical sequence modeling systems in which the objects that are manipulated are trellises containing all the alternative interpretations at a given level. Global training can be performed using stochastic gradient by using a form of back-propagation algorithm to compute the gradients of the loss with respect to all the parameters in the system.

. . .

Restricted Boltzmann Machines (RBMs): Full Boltzmann Machine implementation is extremely difficult and hence comes into picture these RBMs that have only one difference, **Visible nodes are not inter-connected**. RBM is a parameterized generative model representing a probability distribution used to compare the probabilities of (unseen) observations and to sample from the learnt distribution, in particular from marginal distributions of interest. It received a lot of attention after being proposed as building blocks of multi-layer learning architectures called **Deep Belief Networks**. The idea is that the hidden neurons extract relevant features from the observations that serve as input to next RBM that is stacked on top of it, forming a deterministic feed-forward neural network. Our inputs are initialized with generalized weights and passed on to Hidden nodes, which in turn reconstructs our Input nodes, and these reconstructed nodes are never identical to our original Visible nodes. This difference is because as stated earlier, our Visible nodes were never inter-connected so couldn't observe and learn from each other.

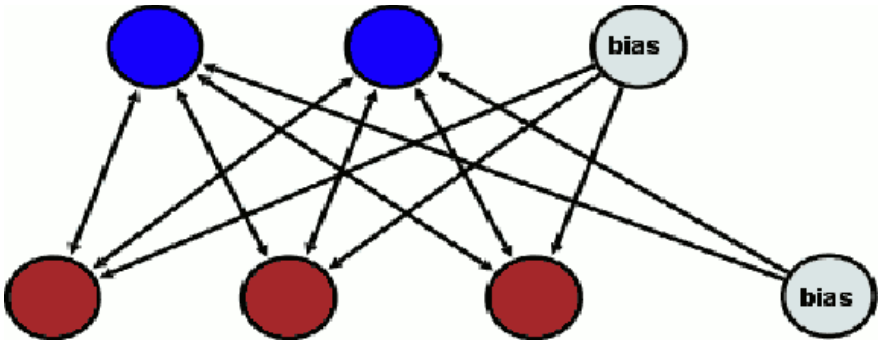


Image Source: Restricted Boltzmann Machine

This reconstruction sequence with *Contrastive Divergence* keeps on continuing till global minimum energy is achieved, and is known as **Gibbs Sampling**. RBM can be interpreted as a stochastic neural network, where nodes and edges correspond to neurons and synaptic connections, respectively. The conditional probability of a single variable being one can be interpreted as the firing rate of a (stochastic) neuron with sigmoid activation function. The independence between the variables in one layer makes Gibbs sampling especially easy because instead of sampling new values for all variables subsequently, the states of all variables in one layer can be sampled jointly. Thus, Gibbs sampling can be performed in just two sub steps: sampling a new state **h** for the hidden neurons based on **p(h|v)** and sampling a state **v**for the visible layer based on **p(v|h)**. This is also referred to as **Block Gibbs sampling**. All common training algorithms for RBMs approximate the *log-likelihood gradient* given some data and perform gradient ascent on these approximations. One such important learning algorithms is contrastive divergence learning. The idea of k-step **Contrastive Divergence Learning(CD-k)** is: Instead of approximating the second term in the log-likelihood gradient by a sample from the RBM-distribution (which would require to run a Markov chain until the stationary distribution is reached), a Gibbs chain is run for only **k** steps (and usually **k = 1**). The Gibbs chain is initialized with a training example **v(0)** of the training set and yields the sample **v(k)** after **k**steps. Each step **t** consists of sampling **h(t)** from **p(h|v(t))** and sampling **v(t+1)** from **p(v|h(t))** subsequently. The gradient w.r.t. **θ** of the *log-likelihood* for one training pattern **v(0)** is then approximated by:

$$\mathrm{CD}_k(\boldsymbol{\theta}, \boldsymbol{v}^{(0)}) = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(0)}) \frac{\partial E(\boldsymbol{v}^{(0)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} + \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(k)}) \frac{\partial E(\boldsymbol{v}^{(k)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \ .$$

Gradient for Contrastive Divergence

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, training batch S
Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, m$
forall the $\boldsymbol{v} \in S$ **do**
  $\boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$
  for $t = 0, \ldots, k-1$ **do**
    for $i = 1, \ldots, n$ **do** sample $h_i^{(t)} \sim p(h_i \,|\, \boldsymbol{v}^{(t)})$
    for $j = 1, \ldots, m$ **do** sample $v_j^{(t+1)} \sim p(v_j \,|\, \boldsymbol{h}^{(t)})$
  for $i = 1, \ldots, n$, $j = 1, \ldots, m$ **do**
    $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \,|\, \boldsymbol{v}^{(0)}) \cdot v_j^{(0)} - p(H_i = 1 \,|\, \boldsymbol{v}^{(k)}) \cdot v_j^{(k)}$
    $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$
    $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \,|\, \boldsymbol{v}^{(0)}) - p(H_i = 1 \,|\, \boldsymbol{v}^{(k)})$

Batch version of CD-k Algorithm

Learning process in **CD-k** also involves possible distortion due to Bias if **k** isn't large as the log-likelihood is not tractable in reasonable sized RBMs. Because the effect depends on the magnitude of the weights, '*weight decay*' can help to prevent it but again it isn't easy to tune them. If weight is too small, *weight decay* has no effect and if too large, the learning converges to models with low likelihood. But recently proposed algorithms try to yield better approximations of the log-likelihood gradient by sampling from Markov chains with increased mixing rate. Then, we also have **Persistent Contrastive Divergence (PCD)** or it's enhanced version as, **Fast Persistent Contrastive Divergence (FPCD)** that tries to reach faster mixing of the Gibbs chain by introducing additional parameters for sampling (& not in the model itself), where learning update rule for fast parameters equals the one for regular parameters, but with an independent, large learning rate leading to faster changes as well as a large *weight decay* parameter. But even this could not sufficiently enlarge mixing rate to avoid the divergence problem.

Divergence concern gave rise to **Parallel Tempering**, which is the most promising learning algorithm for training RBMs as it introduces supplementary Gibbs chains that sample from even more smoothed replicas of the original distribution. In each step of the algorithm, we run **k** (usually **k = 1**) Gibbs sampling steps in each tempered Markov

chain yielding samples **(v1, h1)**,…,**(vM , hM)**. After this, two neighboring Gibbs chains with temperatures **Tr** and **T r−1** may exchange particles **(vr, hr)** and **(vr−1, hr−1)** with an exchange probability based on the Metropolis ratio. After performing these swaps between chains, which enlarge the mixing rate, we take the (eventually exchanged) sample **v1** of original chain (with temperature **T1 = 1**) as a sample from the model distribution. This procedure is repeated **L** times yielding samples **v1**,**1**,…, **v1**,**L** used for the approximation of the expectation under the RBM distribution in the log-likelihood gradient. Usually **L** is set to the number of samples in the (mini) batch of training data as shown in algorithm below,

Algorithm 2. k-step parallel tempering with M temperatures

Input: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, training batch S, current state $\boldsymbol{v}_r$ of Markov chain with stationary distribution p_r for $r = 1, \ldots, M$

Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, m$
forall the $\boldsymbol{v} \in S$ **do**
  for $r = 1, \ldots, M$ **do**
    $\boldsymbol{v}_r^{(0)} \leftarrow \boldsymbol{v}_r$
    for $i = 1, \ldots, n$ **do** sample $h_{r,i}^{(0)} \sim p(h_{r,i} \,|\, \boldsymbol{v}_r^{(0)})$
    for $t = 0, \ldots, k-1$ **do**
      for $j = 1, \ldots, m$ **do** sample $v_{r,j}^{(t+1)} \sim p(v_{r,j} \,|\, \boldsymbol{h}_r^{(t)})$
      for $i = 1, \ldots, n$ **do** sample $h_{r,i}^{(t+1)} \sim p(h_{r,i} \,|\, \boldsymbol{v}_r^{(t+1)})$
    $\boldsymbol{v}_r \leftarrow \boldsymbol{v}_r^{(k)}$
  /* swapping order below works well in practice [26] */
  for $r \in \{s \,|\, 2 \leq s \leq M \text{ and } s \bmod 2 = 0\}$ **do**
    swap $(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)})$ and $(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)})$ with probability given by (40)
  for $r \in \{s \,|\, 3 \leq s \leq M \text{ and } s \bmod 2 = 1\}$ **do**
    swap $(\boldsymbol{v}_r^{k}, \boldsymbol{h}_r^{k})$ and $(\boldsymbol{v}_{r-1}^{k}, \boldsymbol{h}_{r-1}^{k})$ with probability given by (40)
  for $i = 1, \ldots, n$, $j = 1, \ldots, m$ **do**
    $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \,|\, \boldsymbol{v}) \cdot v_j - p(H_i = 1 \,|\, \boldsymbol{v}_1^{(k)}) \cdot v_{1,j}^{(k)}$
    $\Delta b_j \leftarrow \Delta b_j + v_j - v_{1,j}^{(k)}$
    $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \,|\, \boldsymbol{v}) - p(H_i = 1 \,|\, \boldsymbol{v}_1^{(k)})$

Image Source: Batch version of Parallel Tempering Algorithm

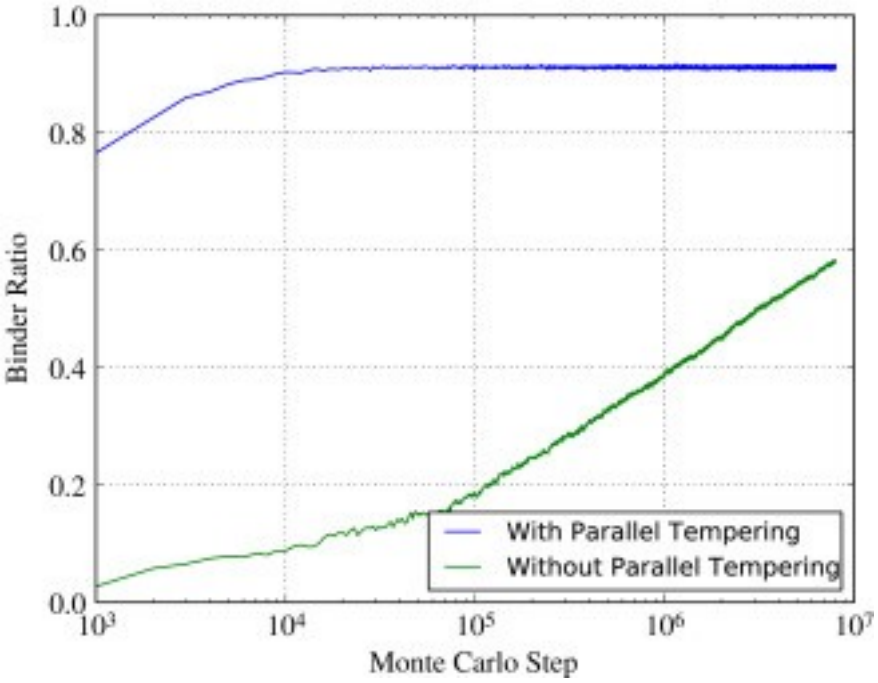


Image Source: Impact of Parallel Tempering

. . .

Information in this post is quite exhaustive and you might feel like getting off the page right now than never so here comes a super cute pair to bring little smile on your face (Nature lovers can use Google search or just manage with the lawn for now!):

No more misusing **Cats** and **Dogs** for **Convolutional Neural network**

Have a cup of coffee, take a small break if required, and head to Part-2, of this story where we shall discuss what actually shall make you stand out in