© Santanu Pattanayak 2017
S. Pattanayak, *Pro Deep Learning with TensorFlow*, https://doi.org/10.1007/978-1-4842-3096-1_5

Unsupervised learning is a branch of machine learning that tries to find hidden structures within unlabeled data and derive insights from it. Clustering, data dimensionality-reduction techniques, noise reduction, segmentation, anomaly detection, fraud detection, and other rich methods rely on unsupervised learning to drive analytics. Today, with so much data around us, it is impossible to label all data for supervised learning. This makes unsupervised learning all the more important. Restricted Boltzmann machines and auto-encoders are unsupervised methods that are based on artificial neural networks. They have a wide range of uses in data compression and dimensionality reduction, noise reduction from data, anomaly detection, generative modeling, collaborative filtering, and initialization of deep neural networks, among other things. We will go through these topics in detail and then touch upon a couple of unsupervised pre-processing techniques for images, namely PCA (principal component analysis) whitening and ZCA (Mahalanobis) whitening. Also, since restricted Boltzmann machines use sampling techniques during training, I have briefly touched upon Bayesian inference and Markov Chain Monte Carlo sampling for reader's benefit.

Boltzmann Distribution

Restricted Boltzmann machines are energy models based on the Boltzmann Distribution Law of classical physics, where the state of particles of any system is represented by their generalized coordinates and velocities. These generalized coordinates and velocities form the phase space of the particles, and the particles can be in any location in the phase space with specific energy and probability. Let's consider a classical system that contains *N* gas molecules and let the generalized position and velocity of any particle be represented by $r \in \mathbb{R}^{3 \times 1}$ and $v \in \mathbb{R}^{3 \times 1}$ respectively. The location of the particle in the phase space can be represented by (r, v). Every such possible value of (r, v) that the particle can take is called a *configuration* of the particle. Further, all the *N* particles are identical in the sense that they are equally likely to take up any state. Given such a system at thermodynamic temperature *T*, the probability of any such configuration is given as follows:

$$P(r,v) \propto e^{-\frac{E(r,v)}{KT}}$$

E(*r*, *v*) is the energy of any particle at configuration (*r*, *v*), and *K* is the Boltzmann Constant. Hence, we see that the probability of any configuration in the phase space is proportional to the exponential of the negative of the energy divided by the product of the Boltzmann Constant and the thermodynamic temperature. To convert the relationship into an equality, the probability needs to be normalized by the sum of the probabilities of all the possible configurations. If there are *M* possible phase-space configurations for the particles, then the probability of any generalized configuration (r, v) can be expressed as

$$P(r,v)=\frac{e^{-\frac{E(r,v)}{KT}}}{Z}$$

where *Z* is the partition function given by

$$Z=\sum_{i=1}^{M} e^{-\frac{E((r,v)_i)}{KT}}$$

There can be several values of *r* and *v* separately. However, *M* denotes all the unique combinations of *r* and *v* possible, which have been denoted by $(r, v)_i$ in the preceding equation. If *r* can take up *n* distinct coordinate values whereas *v* can take up *m* distinct velocity values, then the total number of possible configurations $M = n \times m$. In such cases, the partition function can also be expressed as follows:

$$Z=\sum_{j=1}^{m}\sum_{i=1}^{n} e^{-\frac{E(r_i,v_j)}{KT}}$$

The thing to note here is that the probability of any configuration is higher when its associated energy is low. For the gas molecules, it's intuitive as well given that high-energy states are always associated with unstable equilibrium and hence are less likely to retain the high-energy configuration for long. The particles in the high-energy configuration will always be in a pursuit to occupy much more stable low-energy states.

If we consider two configurations $s_1=(r_1,v_1)$ and $s_2=(r_2,v_2)$, and if the number of gas molecules in these two states are N_1 and N_2 respectively, then the probability ratio of the two states is a function of the energy difference between the two states:

$$\frac{N_1}{N_2}=\frac{P(r_1,v_1)}{P(r_2,v_2)}=e^{-\frac{E(r_1,v_1)-(r_2,v_2)}{KT}}$$

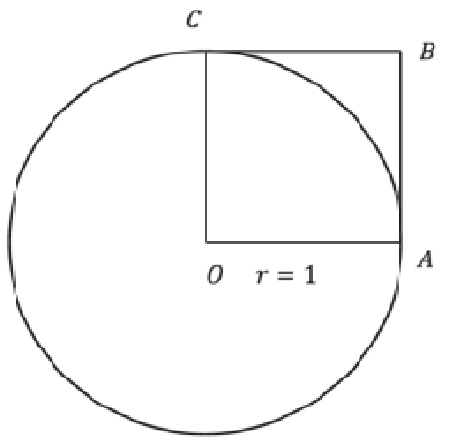
We will digress a little now and briefly discuss Bayesian inference and Markov Chain Monte Carlo (MCMC) methods since restricted Boltzmann machines use sampling through MCMC techniques, especially Gibbs sampling, and some knowledge of these would go a long way toward helping the readers appreciate the working principles of restricted Boltzmann machines.

Markov Chain Monte Carlo Methods for Sampling

Markov Chain Monte Carlo methods, or MCMC, are some of the most popular techniques for sampling from complicated posterior probability distributions or in general from any probability distribution for multi-variate data. Before we get to MCMC, let's talk about Monte Carlo sampling methods in general. Monte Carlo sampling methods try to compute the area under a curve based on sampled points.

For example, the area of the transcendental number *Pi*(π) can be computed by sampling points within a square of radius 1 and noting down the number of sampled points within one-fourth of the circle of diameter 2 enclosed within the square. As shown in Figure 5-2, the area of *Pi* can be computed as follows:

$$\frac{4 * Area(OAC)}{Area(OABC)}=4*\frac{\left(\frac{1}{4}\right)\pi r^2}{r^2}=\pi$$



$$\pi = \frac{4 * Area(OAC)}{Area(OABC)}$$

Figure 5-2. *Area of Pi*

In Listing 5-1, the Monte Carlo method for computing the value of *Pi* is illustrated. As we can see, the value comes out to nearly the value of *Pi*. The accuracy can be improved by sampling more points.

Listing 5-1. Computation of Pi Through Monte Carlo Sampling

```
import numpy as np
number_sample = 100000
inner_area,outer_area = 0,0
for i in range(number_sample):
    x = np.random.uniform(0,1)
    y = np.random.uniform(0,1)
    if (x**2 + y**2) < 1 :
        inner_area += 1
    outer_area += 1

print("The computed value of Pi:",4*(inner_area/float(outer_area)))

--Output--
('The computed value of Pi:', 3.142)
```

The simple Monte Carlo method is highly inefficient if the dimension space is large since the larger the dimensionality is the more prominent the effects of correlation are. Markov Chain Monte Carlo methods are efficient in such scenarios since they spend more time collecting samples from high-probability regions than from lower-probability regions. The normal Monte Carlo method explores the probability space uniformly and hence spends as much time exploring low-probability zones as it does high-probability zones. As we know, the contribution of a low-probability zone is insignificant when computing the expectation of functions through sampling, and hence when an algorithm spends a lot of time in such a zone it leads to significantly higher processing time. The main heuristic behind the Markov Chain Monte Carlo method is to explore the probablity space not uniformly but rather to concentrate more on the high-probability zones. In high-dimensional space, because of correlation, most of the space is sparse, with high density found only at specific areas. So, the idea is to spend more time and collect more samples from those high-probability zones and spend as little time as possible exploring low-probability zones.

Markov Chain can be thought of as a stochastic/random process to generate a sequence of random samples evolving over time. The next value of the random variable is only determined by the prior value of the variable. Markov Chain, once it enters a high-probability zone, tries to collect as many points with a high-probability density as possible. It does so by generating the next sample, conditioned on the current sample value, so that points near the current sample are chosen with high probability and points far away are chosen with low probability. This ensures that the Markov Chain collects as many points as possible from a current high-probability zone. However, occasionally a long jump from the current sample is required to explore other potential high-probability zones far from the current zone where the Markov Chain is working.

The Markov Chain concept can be illustrated with the movement of gas molecules in an enclosed container at a steady state. A few parts of the container have a higher density of gas molecules than the other areas, and since the gas molecules are at a steady state, the probabilities of each state (determined by the position of a gas molecule) would remain constant even though there might be gas molecules moving from one position to another.

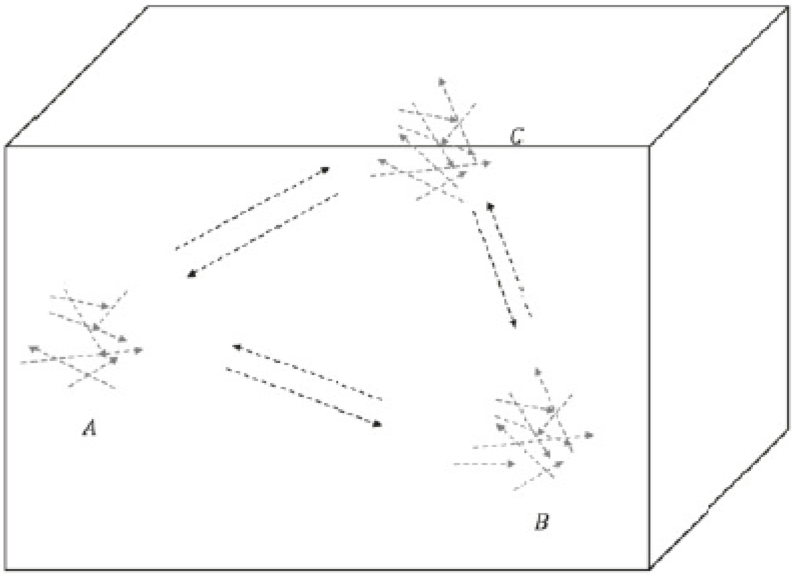


Figure 5-3. *Movement of gases in an enclosed container at steady state with only three states: A, B, and C*

For simplicity's sake, let us assume there are only three states (position of the gas molecules, in this case) for the gas molecules, as shown in Figure 5-3. Let us denote those states by *A*, *B*, and *C* and their corresponding probabilities by P_A, P_B and P_C.

Since the gas molecules are in steady state, if there are gas molecules transitioning to other states, equilibrium needs to be maintained to keep the probability distribution stationary. The simplest assumption to consider is that probability mass going from state *A* to state *B* should come back to *A* from *B*; i.e., pair-wise, the states are in equilibrium.

Let's say $P(B/A)$ determines the transition probability from *A* to *B*. So, probability mass going from *A* to *B* is given by

$$P(A)(B/A) \tag{5.2.1}$$

Likewise, probability mass coming to A from B is given by

$$P(B)P(A/B) \tag{5.2.2}$$

So, in steady state from (5.2.1) and (5.2.2), we have

$$P(A)(B/A)=P(B)P(A/B) \tag{5.2.3}$$

to maintain the stationarity of the probability distribution. This is called a *detailed balance condition*, and it is a sufficient but not necessary condition for the stationarity of a probability distribution. The gas molecules can be in equilibriumin more complex ways, but since this form of detail balance is mathematically convenient when the possible state space is infinite, this approach has been widely used in Markov Chain Monte Carlo methods to sample the next point based on the current point and has a high acceptance probability. In short, movement of the Markov Chain is expected to behave like gas molecules at steady state spending more time in high probability region than in lowprobability keeping the detailed balance condition intact.

A few other conditions that need to be satisfied for a good implementation of Markov Chain are listed here:

Irreducibility — A desirable property of the Markov Chain is that we can go from one stateto any other state. This is important since in Markov Chain, although we want to keep exploring nearby states of a given state with high probability, at times we might have to take a jump and explore some far neighborhood with the expectation that the new zone might be another high-probability zone.

Aperiodicity — The Markov Chain shouldn't repeat too often, as otherwise it won't be possible to traverse the whole space. Imagine a space with 20 states. If, after exploring five states, the chain repeats, it would not be possible to traverse all 20 states, thus leading to sub-optimal sampling.

Metropolis Algorithm

The Metropolis algorithm is a Markov Chain Monte Carlo method that uses the current accepted state to determine the next state. A sample at time $(t+1)$ is conditionaly dependent upon the sample at time t. The proposed state at time $(t+1)$ is drawn from a normal distribution with a mean equal to the current sample at time t with a specified variance. Once drawn, the ratio of the probability is checked between the sample at time $(t+1)$ and time t. If $P(x^{(t+1)})/(Px^{(t)})$ is greater than or equal to 1 then the sample $x^{(t+1)}$ is chosen with a probability of 1; if it is less than 1 then the sample is chosen randomly. Mentioned next are the detailed implementation steps.

- Start with any random sample point $X^{(1)}$.
- Choose the next point $X^{(2)}$ that is conditionally dependent on $X^{(1)}$. You can choose $X^{(2)}$ from a normal distribution with a mean of $X^{(1)}$ and some finite variance, let's say $S^{(2)}$. So, $X^{(2)} \sim Normal\ (X^{(1)}, S^2)$. A key deciding factor for good sampling is to choose the variance S^2 very judicially. The variance shouldn't be too large, since in that case the next sample $X^{(2)}$ has less of a chance of staying nearthe current sample $X^{(1)}$, in which case a high-probability region might not be explored as much since the next sample is selected far away from the current sample most of the time. At the same time, the variance shouldn't be too small. In such cases, the next samples would almost always stay near the current point and hence the probability of exploring a different high-probability zone far from the current zone would reduce.

As per detailed balance,

$$P(X_1 / X_2)P(X_1) = P(X_2 / X_1)P(X_2)$$

Replacing the ideal transition probability as the product of the assumed transition probability and the acceptance probability we get

$$T(X_2 / X_1)A(X_2 / X_1)P(X_1) = T(X_1 / X_2)A(X_1 / X_2)P(X_2)$$

Rearranging this, we get the acceptance probability ratio as

$$\frac{A(X_2 / X_1)}{A(X_1 / X_2)} = \frac{T(X_1 / X_2)P(X_2)}{T(X_2 / X_1)P(X_1)}$$

One simple proposal that satisfies this is given by the Metropolis algorithm as

$$A(X_2 / X_1) = min\left(1, \frac{T(X_1 / X_1)P(X_2)}{T(X_2 / X_1)P(X_1)}\right)$$

In the Metropolis algorithm, the assumed transitional probability is generally assumed to be a normal distribution that is symmetric, and hence $T(X_1/X_2) = T(X_2/X_1)$. This simplifies the acceptance probability of the move from X_1 to X_1 as

$$A(X_2 / X_1) = min\left(1, \frac{P(X_2)}{P(X_1)}\right)$$

If the acceptance probability is 1, then we accept the move with probability 1, while if the acceptance probability is less than 1, let's say *r*, then we accept the new sample with probability *r* and reject the sample with probability $(1-r)$. This rejection of samples with the probability $(1-r)$ is achieved by comparing the ratio with the randomly generated sample r_u from a uniform distribution between 0 and 1 and rejecting the sample in cases where $r_u > r$. This is because for a uniform distribution probability $P(r_u > r) = 1 - r$, which ensures the desired rejection probability is maintained.

In Listing 5-2 we illustrate the sampling from a bivariate Gaussian distribution through the Metropolis algorithm.