April 8-12: Advanced machine learning and data analysis for the physical sciences

Morten Hjorth-Jensen[1,2]

[1]Department of Physics and Center for Computing in Science Education, University of Oslo, Norway
[2]Department of Physics and Astronomy and Facility for Rare Isotope Beams, Michigan State University, East Lansing, Michigan, USA

April 8-12, 2024

Plans for the week April 8-12

Generative methods, energy models and Boltzmann machines.

1. Restricted Boltzmann machines, reminder from last week
2. Reminder on Markov Chain Monte Carlo and Gibbs sampling
3. Discussions of various Boltzmann machines
4. Implementation of Boltzmann machines using TensorFlow and Pytorch
5. Reading recommendation: Goodfellow et al chapters 18, 20.1-20-7

Essential elements of generative models

The aim of generative methods is to train a probability distribution p. The methods we will focus on are

1. Energy based models, with the family of Boltzmann distributions as a typical example
2. Variational autoencoders, based on our discussions on autoencoders
3. Generative adversarial networks (GANs) and
4. Diffusion models

Energy models

Last week we defined a domain $\boldsymbol{X}$ of stochastic variables $\boldsymbol{X} = \{x_0, x_1, \dots, x_{n-1}\}$ with a pertinent probability distribution

$$p(\boldsymbol{X}) = \prod_{x_i \in \boldsymbol{X}} p(x_i),$$

where we have assumed that the random varaibles x_i are all independent and identically distributed (iid).

We will now assume that we can defined this function in terms of optimization parameters $\boldsymbol{\Theta}$, which could be the biases and weights of deep network, and a set of hidden variables we also assume to be random variables which also are iid. The domain of these variables is $\boldsymbol{H} = \{h_0, h_1, \dots, h_{m-1}\}$.

Probability model

We define a probability

$$p(x_i, h_j; \boldsymbol{\Theta}) = \frac{f(x_i, h_j; \boldsymbol{\Theta})}{Z(\boldsymbol{\Theta})},$$

where $f(x_i, h_j; \boldsymbol{\Theta})$ is a function which we assume is larger or equal than zero and obeys all properties required for a probability distribution and $Z(\boldsymbol{\Theta})$ is a normalization constant. Inspired by statistical mechanics, we call it often for the partition function. It is defined as (assuming that we have discrete probability distributions)

$$Z(\boldsymbol{\Theta}) = \sum_{x_i \in \boldsymbol{X}} \sum_{h_j \in \boldsymbol{H}} f(x_i, h_j; \boldsymbol{\Theta}).$$

Marginal and conditional probabilities

We can in turn define the marginal probabilities

$$p(x_i; \boldsymbol{\Theta}) = \frac{\sum_{h_j \in \boldsymbol{H}} f(x_i, h_j; \boldsymbol{\Theta})}{Z(\boldsymbol{\Theta})},$$

and

$$p(h_i; \boldsymbol{\Theta}) = \frac{\sum_{x_i \in \boldsymbol{X}} f(x_i, h_j; \boldsymbol{\Theta})}{Z(\boldsymbol{\Theta})}.$$

Similarly, we have the conditional probabilities

$$p(\boldsymbol{h}|\boldsymbol{x}) = \prod_j p(h_j|\boldsymbol{x}),$$

and

$$p(\boldsymbol{x}|\boldsymbol{h}) = \prod_i p(x_i|\boldsymbol{h}). \tag{1}$$

Optimization problem

We are not interested in the probabilities of the hidden variables. The probability we thus want to optimize is

$$p(\boldsymbol{X};\boldsymbol{\Theta}) = \prod_{x_i \in \boldsymbol{X}} p(x_i;\boldsymbol{\Theta}) = \prod_{x_i \in \boldsymbol{X}} \left(\frac{\sum_{h_j \in \boldsymbol{H}} f(x_i, h_j;\boldsymbol{\Theta})}{Z(\boldsymbol{\Theta})} \right),$$

which we rewrite as

$$p(\boldsymbol{X};\boldsymbol{\Theta}) = \frac{1}{Z(\boldsymbol{\Theta})} \prod_{x_i \in \boldsymbol{X}} \left(\sum_{h_j \in \boldsymbol{H}} f(x_i, h_j;\boldsymbol{\Theta}) \right).$$

We simplify further by rewriting it as

$$p(\boldsymbol{X};\boldsymbol{\Theta}) = \frac{1}{Z(\boldsymbol{\Theta})} \prod_{x_i \in \boldsymbol{X}} f(x_i;\boldsymbol{\Theta}),$$

where we used $p(x_i;\boldsymbol{\Theta}) = \sum_{h_j \in \boldsymbol{H}} f(x_i, h_j;\boldsymbol{\Theta})$. The optimization problem is then

$$\arg \max_{\boldsymbol{\Theta} \in \mathbb{R}^p} p(\boldsymbol{X};\boldsymbol{\Theta}).$$

Optimizing the logarithm instead

Computing the derivatives with respect to the parameters $\boldsymbol{\Theta}$ is easier (and equivalent) with taking the logarithm of the probability. We will thus optimize

$$\arg \max_{\boldsymbol{\Theta} \in \mathbb{R}^p} \log p(\boldsymbol{X};\boldsymbol{\Theta}),$$

which leads to

$$\nabla_{\boldsymbol{\Theta}} \log p(\boldsymbol{X};\boldsymbol{\Theta}) = 0.$$

Reminder on Markov Chain Monte Carlo

Gibbs sampling

Boltzmann Machines, marginal and conditional probabilities

Why use a generative model rather than the more well known discriminative deep neural networks (DNN)?

- Discriminitave methods have several limitations: They are mainly supervised learning methods, thus requiring labeled data. And there are tasks they cannot accomplish, like drawing new examples from an unknown probability distribution.

- A generative model can learn to represent and sample from a probability distribution. The core idea is to learn a parametric model of the probability distribution from which the training data was drawn. As an example

    1. A model for images could learn to draw new examples of cats and dogs, given a training dataset of images of cats and dogs.
    2. Generate a sample of an ordered or disordered Ising model phase, having been given samples of such phases.
    3. Model the trial function for Monte Carlo calculations

- Both use gradient-descent based learning procedures for minimizing cost functions
- Energy based models don't use backpropagation and automatic differentiation for computing gradients, instead turning to Markov Chain Monte Carlo methods.
- DNNs often have several hidden layers. A restricted Boltzmann machine has only one hidden layer, however several RBMs can be stacked to make up Dbeep Belief Networks, of which they constitute the building blocks.

History: The RBM was developed by amongst others Geoffrey Hinton, called by some the "Godfather of Deep Learning", working with the University of Toronto and Google.

A BM is what we would call an undirected probabilistic graphical model with stochastic continuous or discrete units.

It is interpreted as a stochastic recurrent neural network where the state of each unit(neurons/nodes) depends on the units it is connected to. The weights in the network represent thus the strength of the interaction between various units/nodes.

It turns into a Hopfield network if we choose deterministic rather than stochastic units. In contrast to a Hopfield network, a BM is a so-called generative model. It allows us to generate new samples from the learned distribution.

A standard BM network is divided into a set of observable and visible units $\hat{x}$ and a set of unknown hidden units/nodes $\hat{h}$.

Additionally there can be bias nodes for the hidden and visible layers. These biases are normally set to 1.

BMs are stackable, meaning they cwe can train a BM which serves as input to another BM. We can construct deep networks for learning complex PDFs. The layers can be trained one after another, a feature which makes them popular in deep learning

However, they are often hard to train. This leads to the introduction of so-called restricted BMs, or RBMS. Here we take away all lateral connections between nodes in the visible layer as well as connections between nodes in the hidden layer. The network is illustrated in the figure below.

The network

The network layers:

1. A function **x** that represents the visible layer, a vector of M elements (nodes). This layer represents both what the RBM might be given as training input, and what we want it to be able to reconstruct. This might for example be the pixels of an image, the spin values of the Ising model, or coefficients representing speech.

2. The function **h** represents the hidden, or latent, layer. A vector of N elements (nodes). Also called "feature detectors".

The goal of the hidden layer is to increase the model's expressive power. We encode complex interactions between visible variables by introducing additional, hidden variables that interact with visible degrees of freedom in a simple manner, yet still reproduce the complex correlations between visible degrees in the data once marginalized over (integrated out).

The network parameters, to be optimized/learned:

1. **a** represents the visible bias, a vector of same length as **x**.

2. **b** represents the hidden bias, a vector of same lenght as **h**.

3. W represents the interaction weights, a matrix of size $M \times N$.

Joint distribution

The restricted Boltzmann machine is described by a Boltzmann distribution

$$P_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})}, \tag{2}$$

where Z is the normalization constant or partition function, defined as

$$Z = \int \int e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})} d\mathbf{x} d\mathbf{h}. \tag{3}$$

It is common to ignore T_0 by setting it to one.

Network Elements, the energy function

The function $E(\mathbf{x}, \mathbf{h})$ gives the **energy** of a configuration (pair of vectors) $(\mathbf{x}, \mathbf{h})$. The lower the energy of a configuration, the higher the probability of it. This function also depends on the parameters $\mathbf{a}$, $\mathbf{b}$ and W. Thus, when we adjust them during the learning procedure, we are adjusting the energy function to best fit our problem.

Defining different types of RBMs

There are different variants of RBMs, and the differences lie in the types of visible and hidden units we choose as well as in the implementation of the energy function $E(\mathbf{x}, \mathbf{h})$. The connection between the nodes in the two layers is given by the weights w_{ij}.

Binary-Binary RBM: RBMs were first developed using binary units in both the visible and hidden layer. The corresponding energy function is defined as follows:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{i}^{M} x_i a_i - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} x_i w_{ij} h_j, \tag{4}$$

where the binary values taken on by the nodes are most commonly 0 and 1.

Gaussian-Binary RBM: Another varient is the RBM where the visible units are Gaussian while the hidden units remain binary:

$$E(\mathbf{x}, \mathbf{h}) = \sum_{i}^{M} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2}. \tag{5}$$

1. RBMs are Useful when we model continuous data (i.e., we wish $\mathbf{x}$ to be continuous)
2. Requires a smaller learning rate, since there's no upper bound to the value a component might take in the reconstruction

Other types of units include:

1. Softmax and multinomial units
2. Gaussian visible and hidden units
3. Binomial units
4. Rectified linear units

Cost function

When working with a training dataset, the most common training approach is maximizing the log-likelihood of the training data. The log likelihood characterizes the log-probability of generating the observed data using our generative model. Using this method our cost function is chosen as the negative log-likelihood. The learning then consists of trying to find parameters that maximize the probability of the dataset, and is known as Maximum Likelihood Estimation (MLE). Denoting the parameters as $\boldsymbol{\theta} = a_1, ..., a_M, b_1, ..., b_N, w_{11}, ..., w_{MN}$, the log-likelihood is given by

$$\mathcal{L}(\{\theta_i\}) = \langle \log P_\theta(\boldsymbol{x}) \rangle_{data} \tag{6}$$

$$= -\langle E(\boldsymbol{x}; \{\theta_i\}) \rangle_{data} - \log Z(\{\theta_i\}), \tag{7}$$

where we used that the normalization constant does not depend on the data, $\langle \log Z(\{\theta_i\}) \rangle = \log Z(\{\theta_i\})$ Our cost function is the negative log-likelihood, $\mathcal{C}(\{\theta_i\}) = -\mathcal{L}(\{\theta_i\})$

Optimization / Training

The training procedure of choice often is Stochastic Gradient Descent (SGD). It consists of a series of iterations where we update the parameters according to the equation

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla \mathcal{C}(\boldsymbol{\theta}_k) \tag{8}$$

at each k-th iteration. There are a range of variants of the algorithm which aim at making the learning rate η more adaptive so the method might be more efficient while remaining stable.

We now need the gradient of the cost function in order to minimize it. We find that

$$\frac{\partial \mathcal{C}(\{\theta_i\})}{\partial \theta_i} = \langle \frac{\partial E(\boldsymbol{x}; \theta_i)}{\partial \theta_i} \rangle_{data} + \frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i} \tag{9}$$

$$= \langle O_i(\boldsymbol{x}) \rangle_{data} - \langle O_i(\boldsymbol{x}) \rangle_{model}, \tag{10}$$

where in order to simplify notation we defined the "operator"

$$O_i(\boldsymbol{x}) = \frac{\partial E(\boldsymbol{x}; \theta_i)}{\partial \theta_i}, \tag{11}$$

and used the statistical mechanics relationship between expectation values and the log-partition function:

$$\langle O_i(\boldsymbol{x}) \rangle_{model} = \mathrm{Tr} P_\theta(\boldsymbol{x}) O_i(\boldsymbol{x}) = -\frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i}. \tag{12}$$

The data-dependent term in the gradient is known as the positive phase of the gradient, while the model-dependent term is known as the negative phase of

the gradient. The aim of the training is to lower the energy of configurations that are near observed data points (increasing their probability), and raising the energy of configurations that are far from observed data points (decreasing their probability).

The gradient of the negative log-likelihood cost function of a Binary-Binary RBM is then

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial w_{ij}} = \langle x_i h_j \rangle_{data} - \langle x_i h_j \rangle_{model} \tag{13}$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial a_{ij}} = \langle x_i \rangle_{data} - \langle x_i \rangle_{model} \tag{14}$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial b_{ij}} = \langle h_i \rangle_{data} - \langle h_i \rangle_{model}. \tag{15}$$

$$\tag{16}$$

To get the expectation values with respect to the *data*, we set the visible units to each of the observed samples in the training data, then update the hidden units according to the conditional probability found before. We then average over all samples in the training data to calculate expectation values with respect to the data.

Kullback-Leibler relative entropy

When the goal of the training is to approximate a probability distribution, as it is in generative modeling, another relevant measure is the **Kullback-Leibler divergence**, also known as the relative entropy or Shannon entropy. It is a non-symmetric measure of the dissimilarity between two probability density functions p and q. If p is the unkown probability which we approximate with q, we can measure the difference by

$$\mathrm{KL}(p||q) = \int_{-\infty}^{\infty} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x}. \tag{17}$$

Thus, the Kullback-Leibler divergence between the distribution of the training data $f(\boldsymbol{x})$ and the model distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ is

$$\mathrm{KL}(f(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})) = \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log \frac{f(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{\theta})} d\boldsymbol{x} \tag{18}$$

$$= \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} - \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log p(\boldsymbol{x}|\boldsymbol{\theta}) d\boldsymbol{x} \tag{19}$$

$$= \langle \log f(\boldsymbol{x}) \rangle_{f(\boldsymbol{x})} - \langle \log p(\boldsymbol{x}|\boldsymbol{\theta}) \rangle_{f(\boldsymbol{x})} \tag{20}$$

$$= \langle \log f(\boldsymbol{x}) \rangle_{data} + \langle E(\boldsymbol{x}) \rangle_{data} + \log Z \tag{21}$$

$$= \langle \log f(\boldsymbol{x}) \rangle_{data} + \mathcal{C}_{LL}. \tag{22}$$

The first term is constant with respect to $\boldsymbol{\theta}$ since $f(\boldsymbol{x})$ is independent of $\boldsymbol{\theta}$. Thus the Kullback-Leibler Divergence is minimal when the second term is minimal. The second term is the log-likelihood cost function, hence minimizing the Kullback-Leibler divergence is equivalent to maximizing the log-likelihood.

To further understand generative models it is useful to study the gradient of the cost function which is needed in order to minimize it using methods like stochastic gradient descent.

The partition function is the generating function of expectation values, in particular there are mathematical relationships between expectation values and the log-partition function. In this case we have

$$\langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{model} = \int p(\boldsymbol{x}|\boldsymbol{\theta}) \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} d\boldsymbol{x} = -\frac{\partial \log Z(\theta_i)}{\partial \theta_i}. \tag{23}$$

Here $\langle \cdot \rangle_{model}$ is the expectation value over the model probability distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$.

Setting up for gradient descent calculations

Using the previous relationship we can express the gradient of the cost function as

$$\frac{\partial \mathcal{C}_{LL}}{\partial \theta_i} = \langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{data} + \frac{\partial \log Z(\theta_i)}{\partial \theta_i} \tag{24}$$

$$= \langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{data} - \langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{model} \tag{25}$$

$$\tag{26}$$

This expression shows that the gradient of the log-likelihood cost function is a **difference of moments**, with one calculated from the data and one calculated from the model. The data-dependent term is called the **positive phase** and the model-dependent term is called the **negative phase** of the gradient. We see now that minimizing the cost function results in lowering the energy of configurations $\boldsymbol{x}$ near points in the training data and increasing the energy of configurations not observed in the training data. That means we increase the model's probability of configurations similar to those in the training data.

The gradient of the cost function also demonstrates why gradients of unsupervised, generative models must be computed differently from for those of for example FNNs. While the data-dependent expectation value is easily calculated based on the samples $\boldsymbol{x}_i$ in the training data, we must sample from the model in order to generate samples from which to caclulate the model-dependent term. We sample from the model by using MCMC-based methods. We can not sample from the model directly because the partition function Z is generally intractable.

As in supervised machine learning problems, the goal is also here to perform well on **unseen** data, that is to have good generalization from the training data. The distribution $f(x)$ we approximate is not the **true** distribution we wish to

estimate, it is limited to the training data. Hence, in unsupervised training as well it is important to prevent overfitting to the training data. Thus it is common to add regularizers to the cost function in the same manner as we discussed for say linear regression.

Mathematical details

Because we are restricted to potential functions which are positive it is convenient to express them as exponentials.

The original RBM had binary visible and hidden nodes. They were showned to be universal approximators of discrete distributions. It was also shown that adding hidden units yields strictly improved modelling power.

Binary-binary RBMs

The common choice of binary values are 0 and 1. However, in some physics applications, -1 and 1 might be a more natural choice. We will here use 0 and 1. We habe the energy function

$$E_{BB}(\boldsymbol{x}, \mathbf{h}) = -\sum_{i}^{M} x_i a_i - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} x_i w_{ij} h_j. \tag{27}$$

Marginal probability

We have the binary-binary marginal probability defined as

$$p_{BB}(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z_{BB}} e^{\sum_i^M a_i x_i + \sum_j^N b_j h_j + \sum_{ij}^{M,N} x_i w_{ij} h_j} \tag{28}$$

$$= \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a} + \boldsymbol{b}^T \boldsymbol{h} + \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{h}} \tag{29}$$

with the partition function

$$Z_{BB} = \sum_{\boldsymbol{x}, \boldsymbol{h}} e^{\boldsymbol{x}^T \boldsymbol{a} + \boldsymbol{b}^T \boldsymbol{h} + \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{h}}. \tag{30}$$

Marginal Probability Density Function for the visible units

In order to find the probability of any configuration of the visible units we derive the marginal probability density function.

$$
\begin{aligned}
p_{BB}(\boldsymbol{x}) =& \sum_{\boldsymbol{h}} p_{BB}(\boldsymbol{x}, \boldsymbol{h}) \qquad (31)\\
=& \frac{1}{Z_{BB}} \sum_{\boldsymbol{h}} e^{\boldsymbol{x}^T \boldsymbol{a} + \boldsymbol{b}^T \boldsymbol{h} + \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{h}} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \sum_{\boldsymbol{h}} e^{\sum_j^N (b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}) h_j} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \sum_{\boldsymbol{h}} \prod_j^N e^{(b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}) h_j} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \Big(\sum_{h_1} e^{(b_1 + \boldsymbol{x}^T \boldsymbol{w}_{*1}) h_1} \times \sum_{h_2} e^{(b_2 + \boldsymbol{x}^T \boldsymbol{w}_{*2}) h_2} \times \\
& \ldots \times \sum_{h_2} e^{(b_N + \boldsymbol{x}^T \boldsymbol{w}_{*N}) h_N} \Big) \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \prod_j^N \sum_{h_j} e^{(b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}) h_j} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \prod_j^N (1 + e^{b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}}). \qquad (32)
\end{aligned}
$$

Marginal probability for hidden units

A similar derivation yields the marginal probability of the hidden units

$$
p_{BB}(\boldsymbol{h}) = \frac{1}{Z_{BB}} e^{\boldsymbol{b}^T \boldsymbol{h}} \prod_i^M (1 + e^{a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}}). \qquad (33)
$$

Conditional Probability Density Functions

We derive the probability of the hidden units given the visible units using Bayes' rule

$$
\begin{aligned}
p_{BB}(\boldsymbol{h}|\boldsymbol{x}) =& \frac{p_{BB}(\boldsymbol{x},\boldsymbol{h})}{p_{BB}(\boldsymbol{x})} \\
=& \frac{\frac{1}{Z_{BB}} e^{\boldsymbol{x}^T\boldsymbol{a}+\boldsymbol{b}^T\boldsymbol{h}+\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{h}}}{\frac{1}{Z_{BB}} e^{\boldsymbol{x}^T\boldsymbol{a}} \prod_j^N (1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}})} \\
=& \frac{e^{\boldsymbol{x}^T\boldsymbol{a}} e^{\sum_j^N (b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})h_j}}{e^{\boldsymbol{x}^T\boldsymbol{a}} \prod_j^N (1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}})} \\
=& \prod_j^N \frac{e^{(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})h_j}}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}} \\
=& \prod_j^N p_{BB}(h_j|\boldsymbol{x}). \qquad (34)
\end{aligned}
$$

On and off probabilities

From this we find the probability of a hidden unit being "on" or "off":

$$
\begin{aligned}
p_{BB}(h_j=1|\boldsymbol{x}) =& \frac{e^{(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})h_j}}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}} \qquad (35) \\
=& \frac{e^{(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})}}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}} \qquad (36) \\
=& \frac{1}{1+e^{-(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})}}, \qquad (37)
\end{aligned}
$$

and

$$
p_{BB}(h_j=0|\boldsymbol{x}) = \frac{1}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}}. \qquad (38)
$$

Conditional probability for visible units

Similarly we have that the conditional probability of the visible units given the hidden are

$$
\begin{aligned}
p_{BB}(\boldsymbol{x}|\boldsymbol{h}) =& \prod_i^M \frac{e^{(a_i+\boldsymbol{w}_{i*}^T\boldsymbol{h})x_i}}{1+e^{a_i+\boldsymbol{w}_{i*}^T\boldsymbol{h}}} \qquad (39) \\
=& \prod_i^M p_{BB}(x_i|\boldsymbol{h}). \qquad (40)
\end{aligned}
$$

$$p_{BB}(x_i = 1|\boldsymbol{h}) = \frac{1}{1 + e^{-(a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h})}} \tag{41}$$

$$p_{BB}(x_i = 0|\boldsymbol{h}) = \frac{1}{1 + e^{a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}}}. \tag{42}$$

Gaussian-Binary Restricted Boltzmann Machines

Inserting into the expression for $E_{RBM}(\boldsymbol{x}, \boldsymbol{h})$ in equation results in the energy

$$\begin{aligned} E_{GB}(\boldsymbol{x}, \boldsymbol{h}) =& \sum_i^M \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_j^N b_j h_j - \sum_{ij}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2} \\ =& ||\frac{\boldsymbol{x} - \boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 - \boldsymbol{b}^T \boldsymbol{h} - (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \boldsymbol{h}. \end{aligned} \tag{43}$$

Joint Probability Density Function

$$\begin{aligned} p_{GB}(\boldsymbol{x}, \boldsymbol{h}) =& \frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x} - \boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T \boldsymbol{h} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \boldsymbol{h}} \\ =& \frac{1}{Z_{GB}} e^{-\sum_i^M \frac{(x_i - a_i)^2}{2\sigma_i^2} + \sum_j^N b_j h_j + \sum_{ij}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2}} \\ =& \frac{1}{Z_{GB}} \prod_{ij}^{M,N} e^{-\frac{(x_i - a_i)^2}{2\sigma_i^2} + b_j h_j + \frac{x_i w_{ij} h_j}{\sigma_i^2}}, \end{aligned} \tag{44}$$

with the partition function given by

$$Z_{GB} = \int \sum_{\tilde{\boldsymbol{h}}}^{\tilde{\boldsymbol{H}}} e^{-||\frac{\tilde{\boldsymbol{x}} - \boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T \tilde{\boldsymbol{h}} + (\frac{\tilde{\boldsymbol{x}}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \tilde{\boldsymbol{h}}} d\tilde{\boldsymbol{x}}. \tag{45}$$

Marginal Probability Density Functions

We proceed to find the marginal probability densitites of the Gaussian-binary RBM. We first marginalize over the binary hidden units to find $p_{GB}(\boldsymbol{x})$

$$\begin{aligned} p_{GB}(\boldsymbol{x}) =& \sum_{\tilde{\boldsymbol{h}}}^{\tilde{\boldsymbol{H}}} p_{GB}(\boldsymbol{x}, \tilde{\boldsymbol{h}}) \\ =& \frac{1}{Z_{GB}} \sum_{\tilde{\boldsymbol{h}}}^{\tilde{\boldsymbol{H}}} e^{-||\frac{\boldsymbol{x} - \boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T \tilde{\boldsymbol{h}} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \tilde{\boldsymbol{h}}} \\ =& \frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x} - \boldsymbol{a}}{2\boldsymbol{\sigma}}||^2} \prod_j^N (1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}). \end{aligned} \tag{46}$$

We next marginalize over the visible units. This is the first time we marginalize over continuous values. We rewrite the exponential factor dependent on $\boldsymbol{x}$ as a Gaussian function before we integrate in the last step.

$$
\begin{aligned}
p_{GB}(\boldsymbol{h}) =& \int p_{GB}(\tilde{\boldsymbol{x}}, \boldsymbol{h}) d\tilde{\boldsymbol{x}} \\
=& \frac{1}{Z_{GB}} \int e^{-||\frac{\tilde{\boldsymbol{x}}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T\boldsymbol{h} + (\frac{\tilde{\boldsymbol{x}}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W}\boldsymbol{h}} d\tilde{\boldsymbol{x}} \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \int \prod_i^M e^{-\frac{(\tilde{x}_i - a_i)^2}{2\sigma_i^2} + \frac{\tilde{x}_i \boldsymbol{w}_{i*}^T \boldsymbol{h}}{\sigma_i^2}} d\tilde{\boldsymbol{x}} \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \Bigg(\int e^{-\frac{(\tilde{x}_1 - a_1)^2}{2\sigma_1^2} + \frac{\tilde{x}_1 \boldsymbol{w}_{1*}^T \boldsymbol{h}}{\sigma_1^2}} d\tilde{x}_1 \\
& \times \int e^{-\frac{(\tilde{x}_2 - a_2)^2}{2\sigma_2^2} + \frac{\tilde{x}_2 \boldsymbol{w}_{2*}^T \boldsymbol{h}}{\sigma_2^2}} d\tilde{x}_2 \\
& \times ... \\
& \times \int e^{-\frac{(\tilde{x}_M - a_M)^2}{2\sigma_M^2} + \frac{\tilde{x}_M \boldsymbol{w}_{M*}^T \boldsymbol{h}}{\sigma_M^2}} d\tilde{x}_M \Bigg) \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{(\tilde{x}_i - a_i)^2 - 2\tilde{x}_i \boldsymbol{w}_{i*}^T \boldsymbol{h}}{2\sigma_i^2}} d\tilde{x}_i \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{\tilde{x}_i^2 - 2\tilde{x}_i(a_i + \tilde{x}_i \boldsymbol{w}_{i*}^T \boldsymbol{h}) + a_i^2}{2\sigma_i^2}} d\tilde{x}_i \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{\tilde{x}_i^2 - 2\tilde{x}_i(a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}) + (a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h})^2 - (a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h})^2 + a_i^2}{2\sigma_i^2}} d\tilde{x}_i \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{(\tilde{x}_i - (a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}))^2 - a_i^2 - 2a_i \boldsymbol{w}_{i*}^T \boldsymbol{h} - (\boldsymbol{w}_{i*}^T \boldsymbol{h})^2 + a_i^2}{2\sigma_i^2}} d\tilde{x}_i \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M e^{\frac{2a_i \boldsymbol{w}_{i*}^T \boldsymbol{h} + (\boldsymbol{w}_{i*}^T \boldsymbol{h})^2}{2\sigma_i^2}} \int e^{-\frac{(\tilde{x}_i - a_i - \boldsymbol{w}_{i*}^T \boldsymbol{h})^2}{2\sigma_i^2}} d\tilde{x}_i \\
=& \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \sqrt{2\pi\sigma_i^2} e^{\frac{2a_i \boldsymbol{w}_{i*}^T \boldsymbol{h} + (\boldsymbol{w}_{i*}^T \boldsymbol{h})^2}{2\sigma_i^2}}.
\end{aligned} \tag{47}
$$

Conditional Probability Density Functions

We finish by deriving the conditional probabilities.

$$\begin{aligned}
p_{GB}(\boldsymbol{h}|\boldsymbol{x}) =& \frac{p_{GB}(\boldsymbol{x},\boldsymbol{h})}{p_{GB}(\boldsymbol{x})} \\
=& \frac{\frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T\boldsymbol{h} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W}\boldsymbol{h}}}{\frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2} \prod_j^N (1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}})} \\
=& \prod_j^N \frac{e^{(b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}) h_j}}{1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}} \\
=& \prod_j^N p_{GB}(h_j|\boldsymbol{x}). \qquad (48)
\end{aligned}$$

The conditional probability of a binary hidden unit h_j being on or off again takes the form of a sigmoid function

$$\begin{aligned}
p_{GB}(h_j = 1|\boldsymbol{x}) =& \frac{e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}}{1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}} \\
=& \frac{1}{1 + e^{-b_j - (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}} \qquad (49) \\
p_{GB}(h_j = 0|\boldsymbol{x}) =& \frac{1}{1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}}. \qquad (50)
\end{aligned}$$

The conditional probability of the continuous $\boldsymbol{x}$ now has another form, however.

$$
\begin{aligned}
p_{GB}(\boldsymbol{x}|\boldsymbol{h}) =& \frac{p_{GB}(\boldsymbol{x},\boldsymbol{h})}{p_{GB}(\boldsymbol{h})} \\
=& \frac{\frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T\boldsymbol{h} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W}\boldsymbol{h}}}{\frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \sqrt{2\pi\sigma_i^2} e^{\frac{2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} \frac{e^{-\frac{(x_i-a_i)^2}{2\sigma_i^2} + \frac{x_i \boldsymbol{w}_{i*}^T\boldsymbol{h}}{2\sigma_i^2}}}{e^{\frac{2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} \frac{e^{-\frac{x_i^2 - 2a_i x_i + a_i^2 - 2x_i \boldsymbol{w}_{i*}^T\boldsymbol{h}}{2\sigma_i^2}}}{e^{\frac{2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{x_i^2 - 2a_i x_i + a_i^2 - 2x_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + 2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i - b_i - \boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}} \\
=& \prod_i^M \mathcal{N}(x_i|b_i + \boldsymbol{w}_{i*}^T\boldsymbol{h}, \sigma_i^2) \qquad (51) \\
\Rightarrow p_{GB}(x_i|\boldsymbol{h}) =& \mathcal{N}(x_i|b_i + \boldsymbol{w}_{i*}^T\boldsymbol{h}, \sigma_i^2). \qquad (52)
\end{aligned}
$$

The form of these conditional probabilities explains the name "Gaussian" and the form of the Gaussian-binary energy function. We see that the conditional probability of x_i given $\boldsymbol{h}$ is a normal distribution with mean $b_i + \boldsymbol{w}_{i*}^T\boldsymbol{h}$ and variance σ_i^2.

Code for RBMs using PyTorch

```
import numpy as np
import torch
import torch.utils.data
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
from torchvision import datasets, transforms
from torchvision.utils import make_grid , save_image
import matplotlib.pyplot as plt


batch_size = 64
train_loader = torch.utils.data.DataLoader(
datasets.MNIST('./data',
    train=True,
    download = True,
```

```
    transform = transforms.Compose(
        [transforms.ToTensor()])
     ),
     batch_size=batch_size
)

test_loader = torch.utils.data.DataLoader(
datasets.MNIST('./data',
    train=False,
    transform=transforms.Compose(
    [transforms.ToTensor()])
    ),
    batch_size=batch_size)


class RBM(nn.Module):
   def __init__(self,
                n_vis=784,
                n_hin=500,
                k=5):
        super(RBM, self).__init__()
        self.W = nn.Parameter(torch.randn(n_hin,n_vis)*1e-2)
        self.v_bias = nn.Parameter(torch.zeros(n_vis))
        self.h_bias = nn.Parameter(torch.zeros(n_hin))
        self.k = k

   def sample_from_p(self,p):
       return F.relu(torch.sign(p - Variable(torch.rand(p.size()))))

   def v_to_h(self,v):
        p_h = F.sigmoid(F.linear(v,self.W,self.h_bias))
        sample_h = self.sample_from_p(p_h)
        return p_h,sample_h

   def h_to_v(self,h):
        p_v = F.sigmoid(F.linear(h,self.W.t(),self.v_bias))
        sample_v = self.sample_from_p(p_v)
        return p_v,sample_v

   def forward(self,v):
        pre_h1,h1 = self.v_to_h(v)

        h_ = h1
        for _ in range(self.k):
            pre_v_,v_ = self.h_to_v(h_)
            pre_h_,h_ = self.v_to_h(v_)

        return v,v_

   def free_energy(self,v):
        vbias_term = v.mv(self.v_bias)
        wx_b = F.linear(v,self.W,self.h_bias)
        hidden_term = wx_b.exp().add(1).log().sum(1)
        return (-hidden_term - vbias_term).mean()



rbm = RBM(k=1)
train_op = optim.SGD(rbm.parameters(),0.1)
```

```
for epoch in range(10):
    loss_ = []
    for _, (data,target) in enumerate(train_loader):
        data = Variable(data.view(-1,784))
        sample_data = data.bernoulli()

        v,v1 = rbm(sample_data)
        loss = rbm.free_energy(v) - rbm.free_energy(v1)
        loss_.append(loss.data)
        train_op.zero_grad()
        loss.backward()
        train_op.step()

    print("Training loss for {} epoch: {}".format(epoch, np.mean(loss_)))


def show_adn_save(file_name,img):
    npimg = np.transpose(img.numpy(),(1,2,0))
    f = "./%s.png" % file_name
    plt.imshow(npimg)
    plt.imsave(f,npimg)

show_adn_save("real",make_grid(v.view(32,1,28,28).data))
show_adn_save("generate",make_grid(v1.view(32,1,28,28).data))
```