Laboratory Presentation:
Introduction to Restricted Boltzmann Machine

Woramethe CHINCHUTHAKUN

May 9, 2022

Yamashita Laboratory
Department of Transdisciplinary Science and Engineering (GEDES)
School of Environment and Society
Tokyo Institute of Technology

Introduction

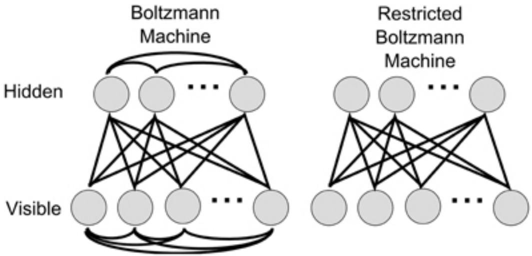
- **A little bit of history [1]**
  - Boltzmann machine [2] and its restricted variant [3] were proposed in 1980s
  - Few years later, **back propagation** [4] was proposed, but it suffered from **vanishing gradients**
  - In 2006, Deep Belief Network [5] can be successfully trained with **a good weight initialization** (or **pre-training**)
  - Thus, the winter of neural network gradually ended together with the golden era of support vector machine (SVM) [6]
- With availability of more advanced network architectures and optimization algorithms, RBM and its variants are essentially obsolete now...

Outline

1. Definition
2. Application
3. Training
4. Extensions
5. Available Implementations

Definition: Network Architecture

- A two-layer **generative** neural network that learns the data distribution in a **unsupervised manner**
  - **Visible nodes** and **hidden nodes** form a bipartite graph
  - Input and their latent representations are restricted to **binary** (**Bernoulli RBM**)
  - A network state is represented by $(\mathbf{a}, \mathbf{b}, \mathbf{W})$ which are (1) **bias for input units**, (2) **bias for hidden units**, and (3) **weight of connections**



Definition: Energy and Probability

- For a given state $(\mathbf{a}, \mathbf{b}, \mathbf{W})$, the **joint probability distribution** of input data $\mathbf{x}$ and its latent representation $\mathbf{h}$ are

$$P(\mathbf{x}, \mathbf{h}) = \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{Z} \tag{1a}$$

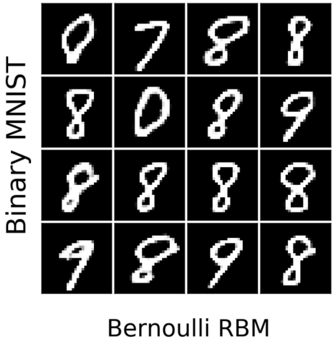
$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^\intercal \mathbf{W} \mathbf{x} - \mathbf{a}^\intercal \mathbf{x} - \mathbf{b}^\intercal \mathbf{x} \tag{1b}$$

  where Z is a normalizing constant (also known as **partition function**)
- Therefore, we can learn the most appropriate **marginal distribution of input** $P(\mathbf{x})$ by adjusting state of a network

The lower the energy, the higher a probability

How can we use a RBM?

- If we have a trained RBM, we can (1) **calculate the latent representation of an input** and (2) **generate a new sample** by using **conditional distributions**
  - Generated samples from a Bernoulli RBM trained on MNIST dataset [7] in [8]



$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x}) \tag{2a}$$

$$P(h_i = 1|\mathbf{x}) = \sigma(b_i + \mathbf{W}[i,:]\mathbf{x}) \tag{2b}$$

$$P(\mathbf{x}|\mathbf{h}) = \prod_j P(x_j|\mathbf{h}) \tag{3a}$$

$$P(x_j = 1|\mathbf{h}) = \sigma(a_j + \mathbf{h}^\mathsf{T}\mathbf{W}[:,j]) \tag{3b}$$

Training a RBM: Problem Formulation

- Given a training dataset $\mathbf{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, the optimal network configuration can be expressed by

$$(\mathbf{a}^*, \mathbf{b}^*, \mathbf{W}^*) = \underset{(\mathbf{a},\mathbf{b},\mathbf{W})}{\arg\min}\, \mathbb{E}_{\mathbf{X}}[-\log P(\mathbf{x})] = \underset{(\mathbf{a},\mathbf{b},\mathbf{W})}{\arg\min}\, \frac{1}{N}\sum_{i=1}^{N}\Big(-\log P(\mathbf{x}^{(i)})\Big) \quad (4)$$

- Similar to other neural networks, we employ **Stochastic Gradient Descent (SGD)** to solve it:

$$-\frac{\partial \log P(\mathbf{x}^{(i)})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}^{(i)}, \mathbf{h})}{\partial \boldsymbol{\theta}}\middle|\mathbf{x}^{(i)}\right] - \mathbb{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \boldsymbol{\theta}}\right] \quad (5)$$

However, the negative phase is intractable because the number of possible values of $\mathbf{x}$ grows exponentially with the input dimension.

Contrastive Divergence [9]

- We use a **point estimate at $\widehat{\mathbf{x}}$** instead of the expectation $\mathbb{E}_{\mathbf{X}}$
- $\widehat{\mathbf{x}}$ is chosen via **Gibb sampling** [10]
  - Generate **a sequence of samples** drawn from a joint distribution such that it asymptotically converges to the true joint distribution
  - Set the initial value to be $\mathbf{x}^{(i)}$

Initialization: Initialize $\mathbf{x}^{(0)} \in \mathcal{R}^D$ and number of samples N

- **for** $i = 0$ to $N-1$ **do**
- $x_1^{(i+1)} \sim p(x_1 | x_2^{(i)}, x_3^{(i)}, ..., x_D^{(i)})$
- $x_2^{(i+1)} \sim p(x_2 | x_1^{(i+1)}, x_3^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_j^{(i+1)} \sim p(x_j | x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_D^{(i+1)} \sim p(x_D | x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{D-1}^{(i+1)})$

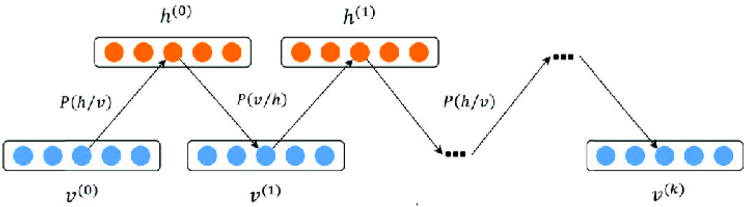
return $(\{\mathbf{x}^{(i)}\}_{i=0}^{N-1})$

Gibb sampling

[1] Image taken from `https://towardsdatascience.com/can-you-do-better-sampling-strategies-with-an-emphasis-on-gibbs-sampling-practicals-and-code-c97730d54ebc`

Training a RBM: Algorithm (2)

- Sampling on conditional probabilities are performed by using the RBM
- **Contrastive Divergence with k-step Gibb sampling (CD-k)**
  1. Approximate the positive phase by sampling $\widehat{\mathbf{h}}^{(i)} \sim P(\mathbf{h}|\mathbf{x}^{(i)})$
  2. Obtain $\widehat{\mathbf{x}}$ by Gibb sampling with an initial value of $\mathbf{x}^{(i)}$ and sampling $\widehat{\mathbf{h}} \sim P(\mathbf{h}|\widehat{\mathbf{x}})$
  3. Approximate the negative phase with $\widehat{\mathbf{x}}$ and $\widehat{\mathbf{h}}$
  4. Repeat this step for all training samples $\mathbf{x}^{(i)} \in \mathbf{X}$



[1] Image taken from `https://www.researchgate.net/figure/Alternating-Gibbs-sampling-of-the-RBM_fig2_338722271`

Training a RBM: Algorithm (3)

- The gradient can be approximated as

$$-\frac{\partial \log P(\mathbf{x}^{(i)})}{\partial \boldsymbol{\theta}} \approx \frac{\partial E(\mathbf{x}^{(i)}, \widehat{\mathbf{h}}^{(i)})}{\partial \boldsymbol{\theta}} - \frac{\partial E(\widehat{\mathbf{x}}, \widehat{\mathbf{h}})}{\partial \boldsymbol{\theta}} \tag{6}$$

- I will skip the derivations, but essentially the update rules become

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha\left(\widehat{\mathbf{h}}^{(i)}\left(\mathbf{x}^{(i)}\right)^{\mathsf{T}} - \widehat{\mathbf{h}}\widehat{\mathbf{x}}^{\mathsf{T}}\right) \tag{7a}$$

$$\mathbf{a} \leftarrow \mathbf{a} + \alpha\left(\mathbf{x}^{(i)} - \widehat{\mathbf{x}}\right) \tag{7b}$$

$$\mathbf{b} \leftarrow \mathbf{b} + \alpha\left(\widehat{\mathbf{h}}^{(i)} - \widehat{\mathbf{x}}\right) \tag{7c}$$

Training a RBM: Algorithm (4)

- **How to choose the value for k?**
  - Of course, as large as computationally feasible because the gradient estimate will be less biased
  - $k = 1$ is used for **pre-training**
- **Persistent Contrastive Divergence (PCD) [11]**
  1. Approximate the positive phase by sampling $\widehat{\mathbf{h}}^{(i)} \sim P(\mathbf{h}|\mathbf{x}^{(i)})$
  2. Obtain $\widehat{\mathbf{x}}$ by Gibb sampling with an initial value of **the previous** $\widehat{\mathbf{x}}$ and sampling $\widehat{\mathbf{h}} \sim P(\mathbf{h}|\widehat{\mathbf{x}})$
  3. Approximate the negative phase with $\widehat{\mathbf{x}}$ and $\widehat{\mathbf{h}}$
  4. Repeat this step for all training samples $\mathbf{x}^{(i)} \in \mathbf{X}$

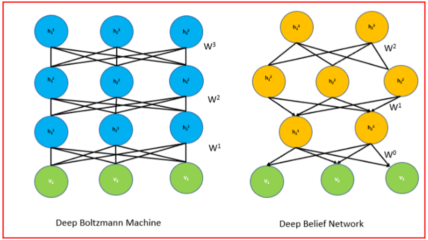
Some Extensions of RBM

- **Guassian-Bernoulli RBM (GRBM)** [12]
  - Relax the restriction that $\mathbf{x} \in \{0, 1\}^n$
  - Change only the energy function, then we can derive every formula in a similar manner

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^\intercal \mathbf{W} \mathbf{x} - \mathbf{a}^\intercal \mathbf{x} - \mathbf{b}^\intercal \mathbf{x} + \frac{1}{2} \mathbf{x}^\intercal \mathbf{x} \tag{8}$$

  - Now, we have $P(\mathbf{x}|\mathbf{h}) \sim \mathcal{N}(\mathbf{a} + \mathbf{W}^\intercal \mathbf{h}, \mathbf{I})$
  - Learning rate is generally lower than the vanilla Bernoulli RBM

- **Deep Boltzmann Machine (DBM)** [13]
  - In principle, stacked multiple RBMs on top of each other
  - Pre-train by **greedy layer-wise CDs** and fine-tune with **back propagation**
- **Deep Belief Network (DBN)** [5]
  - Architecture is very similar to DBM, different in only connection between layers



[1] Image taken from `https://medium.datadriveninvestor.com/deep-learning-deep-boltzmann-machine-dbm-e3253bb95d0f`

What if I want to play around with it?

- **Scikit-learn** [1]
  - Bernoulli RBM trained with PCD
- **TensorFlow** [2]
  - An opensource implementation for Bernoulli RBM and Gaussian-Bernoulli RBM
  - Compatible with TensorFlow 2, but no longer maintained
- **PyTorch** [3]
  - It seems there are many open-source implementations of Bernoulli RBM, a quick Google search will suffice...

[1] *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.BernoulliRBM.html*
[2] *https://github.com/meownoid/tensorflow-rbm*
[3] *https://github.com/bacnguyencong/rbm-pytorch*

References i

[1] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, “Restricted boltzmann machine and deep belief network: Tutorial and survey,” *CoRR*, vol. abs/2107.12521, 2021. [Online]. Available: https://arxiv.org/abs/2107.12521

[2] G. E. Hinton and J. Sejnowski, “Optimal perceptual inference,” 1983.

[3] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cogn. Sci.*, vol. 9, pp. 147–169, 1985.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, p. 1527–1554, jul 2006. [Online]. Available: https://doi.org/10.1162/neco.2006.18.7.1527

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT ’92. New York, NY, USA: Association for Computing Machinery, 1992, p. 144–152. [Online]. Available: https://doi.org/10.1145/130385.130401

[7] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[8] C. K. Fisher, A. M. Smith, and J. R. Walsh, “Boltzmann encoded adversarial machines,” 2018. [Online]. Available: https://arxiv.org/abs/1804.08682

[9] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural Comput.*, vol. 14, no. 8, p. 1771–1800, aug 2002. [Online]. Available: https://doi.org/10.1162/089976602760128018

[10] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[11] T. Tieleman, "Training restricted boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 1064–1071. [Online]. Available: https://doi.org/10.1145/1390156.1390290

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[13] R. Salakhutdinov and H. Larochelle, "Efficient learning of deep boltzmann machines," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 693–700. [Online]. Available: https://proceedings.mlr.press/v9/salakhutdinov10a.html

Derivation of $P(\mathbf{h}|\mathbf{x})$

- Here, we show the brief outline to derive $P(\mathbf{h}|\mathbf{x})$

$$
\begin{aligned}
P(\mathbf{h}|\mathbf{x}) &= P(\mathbf{x},\mathbf{h}) / \sum_{\mathbf{h}} (\mathbf{x},\mathbf{h}) \\
&= \frac{\exp(\mathbf{h}^\intercal \mathbf{W}\mathbf{x} + \mathbf{a}^\intercal \mathbf{x} + \mathbf{b}^\intercal \mathbf{x})}{\sum_{\mathbf{h}} \exp(\mathbf{h}^\intercal \mathbf{W}\mathbf{x} + \mathbf{a}^\intercal \mathbf{x} + \mathbf{b}^\intercal \mathbf{x})} \\
&= \frac{\prod_i \exp(h_i \mathbf{W}_i \mathbf{x} + b_i h_i)}{\prod_i \left(\sum_{h_i \in \{0,1\}} (\exp(h_i \mathbf{W}_i \mathbf{x} + b_i h_i)) \right)} \\
&= \prod_i \left(\frac{\exp(h_i \mathbf{W}_i \mathbf{x} + b_i h_i)}{1 + \exp(h_i \mathbf{W}_i \mathbf{x} + b_i)} \right)
\end{aligned}
\tag{9}
$$

$$\therefore P(h_i = 1|\mathbf{x}) = \frac{\exp(\mathbf{W}_i \mathbf{x} + b_i)}{1 + \exp(\mathbf{W}_i \mathbf{x} + b_i)} = \sigma(b_i + \mathbf{W}_i \mathbf{x})$$

Derivation of Update rule for $\mathbf{W}$

- First, we consider the gradient of $E(\mathbf{x}, \mathbf{h})$ with respect to $\mathbf{W}$

$$\begin{aligned}\nabla_{\mathbf{W}} E(\mathbf{x}, \mathbf{h}) &= \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial W_{ij}}\right]_{ij} \\ &= \left[-\frac{\partial}{\partial W_{ij}} \sum_{ij} W_{ij} h_i x_j\right]_{ij} = -\mathbf{h}\mathbf{x}^{\mathsf{T}} \qquad (10)\end{aligned}$$

- Thus, it follows that

$$\nabla_{\mathbf{W}} E(\mathbf{x}^{(i)}, \widehat{\mathbf{h}}^{(i)}) = -\widehat{\mathbf{h}}^{(i)} \left(\mathbf{x}^{(i)}\right)^{\mathsf{T}} \qquad (11a)$$

$$\nabla_{\mathbf{W}} E(\widehat{\mathbf{x}}, \widehat{\mathbf{h}}) = -\widehat{\mathbf{h}}(\widehat{\mathbf{x}})^{\mathsf{T}} \qquad (11b)$$

Derivation of Update rule for $\mathbf{W}$ (2)

- The update rule of stochastic gradient descent becomes

$$
\begin{aligned}
\mathbf{W} &\leftarrow \mathbf{W} - \alpha\left(-\nabla_{\mathbf{W}} \log P(\mathbf{x}^{(i)})\right) \\
&\leftarrow \mathbf{W} - \alpha\left(\nabla_{\mathbf{W}} E(\mathbf{x}^{(i)}, \widehat{\mathbf{h}}^{(i)}) - \nabla_{\mathbf{W}} E(\widehat{\mathbf{x}}, \widehat{\mathbf{h}})\right) \\
&\leftarrow \mathbf{W} + \alpha\left(\widehat{\mathbf{h}}^{(i)}\left(\mathbf{x}^{(i)}\right)^{\mathsf{T}} - \widehat{\mathbf{h}}\widehat{\mathbf{x}}^{\mathsf{T}}\right)
\end{aligned}
$$

- We can derive update rules for bias $\mathbf{a}$ and $\mathbf{b}$ in a similar manner