Restricted Boltzmann Machines

Jason Liang, Julian Sia

***Abstract*—In this report**

I. Introduction

Restricted Boltzmann Machines (RBM) are a type of undirected graphical model consisting of two sets of nodes, a set of hidden units and a set of visible units. Given the graph $G = (V, E)$ that describes the RBM, $\forall\, v_{vis}\, \epsilon\, V_{vis} \subset V, \exists e' \epsilon E$, such that $e' = (v_{vis}, v_{hid})$, $\forall\, v_{hid}\, \epsilon\, V_{hid} \subset V$. RBMs are a variant of the Boltzmann Machine [1] posed by Hinton and Sejnowski, with the restriction that there are neither edges connecting any visible units nor edges connecting any hidden units [2]. This restriction guarantees that the visible units are conditionally independent of other visible units; and hidden units, conditionally independent of hidden units. RBMs today are commonly used in classification tasks, dimensionality reduction, and feature learning tasks. The structure can be visually elucidated with the representation below. This specific RBM has a hidden layer of 3 units and a visible layer of 6 units.

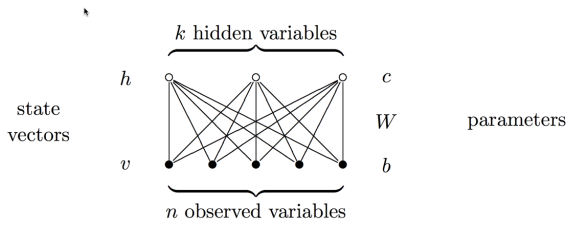

Fig. 1. A graphical representation of a Restricted Boltzmann Machine illustrated by Cueto et. al [3]

II. Previous Work

RBMs were created as a variant of Boltzmann Machines after practical difficulties were encountered in training Boltzmann machines, namely the exponential training time required to learn weights in a Boltzmann Machine and sequential nature of training particular to the Boltzmann Machine. Boltzmann Machine themselves were motivated primarily by limitations of Hopfield Networks [4]: their poor storage capacity and tendency to converge at local energy minima.

In the early days of Boltzmann machines, Hinton and Sejnowski calculated the gradient of the log likelihood by "[fixing] a training vector on the visible units, initializing the hidden units to random binary states, and using sequential Gibbs sampling of the hidden units" [1], using simulated annealing to speed up the convergence process. Other attempts were made by Neal with persistent Markov chains [5] and Peterson and Anderson with mean-field methods (instead of Gibbs Sampling) to speed up the learning process [6]. After RBMs were posed as a simplifying restriction by Smolensky, Hinton discovered a way to speed up the learning process, coined "contrastive divergence," (CD) which is described in this report [7], [2], [8].

III. Algorithm Description

Given a set of training vectors, V, to train a RBM, one aims to maximize the average log probability, $p(v), v \epsilon V$, where

$$p(v) = \frac{1}{Z} \sum_{h} e^{-E(v,h)},\ Z = \sum_{v,h} e^{-E(v,h)} \tag{1}$$

One can do accomplish this by taking the derivative of the log probability of $p(v)$ with respect to the weights defined in the energy function, $E(v, h)$ defined as

$$\begin{aligned} E(v,h) =& -\sum_{i\,\epsilon\,visible} a_i v_k - \sum_{j\,\epsilon\,hidden} b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j \\ =& -b'v - c'h - h'Wv \end{aligned} \tag{2}$$

Differentiating with respect to $w_{i,j}$, we find that the partial derivative reduces to the following:

$$\frac{\partial p(v)}{\partial w_{i,j}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{3}$$

With this reduction the following learning rule with ϵ learning rate can be derived

$$\Delta w_{i,j} = \epsilon \langle \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \rangle \tag{4}$$

Alternatively, using free energy notation, the partial derivative can be defined as

$$-\frac{\partial p(v)}{\partial \Theta} = \frac{\partial \mathcal{F}(v)}{\partial \Theta} - \sum_{\tilde{v}} p(\tilde{v}) \frac{\partial \mathcal{F}(\tilde{v})}{\partial \Theta} \tag{5}$$

where we define free energy as

$$\mathcal{F}(v) = -log \sum_{h} e^{-E(v,h)} \tag{6}$$

Because the computation of the second term in the difference that defines the partial derivative is intractable, we can approximate the second term using a fixed number of samples from the model, such that

$$-\frac{\partial p(v)}{\partial \Theta} = \frac{\partial \mathcal{F}(v)}{\partial \Theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{v}\epsilon\mathcal{N}} p(\tilde{v}) \frac{\partial \mathcal{F}(\tilde{v})}{\partial \Theta} \tag{7}$$

where theta represents the set of parameteters which include the weights and the biases for the hidden and visible nodes.

In the case of a binary RBM, the free energy function reduces to the following:

$$\mathcal{F}(v) = -b'v - \sum_i log \sum_{h_i} e^{h_i(c_i + W_i v)} \tag{8}$$

Using the free energy function defined in (8) and the derivative defined in (7), we can define the following update equations for the RBM's log likelihood gradient:

$$\begin{aligned} -\frac{\partial p(v)}{\partial W_{i,j}} &= E_v[p(h_i|v) \cdot v_j] - v_j^{(i)} \cdot sigm(W_i \cdot v^{(i)} + c_i) \\ -\frac{\partial p(v)}{\partial c_i} &= E_v[p(h_i|v)] - v_j^{(i)} \cdot sigm(W_i \cdot v^{(i)}) \\ -\frac{\partial p(v)}{\partial b_j} &= E_v[p(h_i|v)] - v_j^{(i)} \end{aligned} \tag{9}$$

To obtain samples from p(x), one can use Gibbs sampling until a Markov chain converges or near converges. Because of the conditional independence assumptions of the visible and hidden nodes such that

$$\begin{aligned} p(h|v) &= \prod_i p(h_i|v) \\ p(v|h) &= \prod_j p(v_j|h) \end{aligned} \tag{10}$$

we can define the following activation functions:

$$\begin{aligned} p(h_i = 1|v) &= sigm(c_i + W_i v) \\ p(v_j = 1|j) &= sigm(b_j + W_i' h) \end{aligned} \tag{11}$$

Given the conditional independence assumptions of the RBM, we can perform block Gibbs Sampling with the following update rules:

$$\begin{aligned} h^{(n+1)} &\sim sigm(W' v^{(n)} + c) \\ v^{(n+1)} &\sim sigm(W h^{(n+1)} + b) \end{aligned} \tag{12}$$

As $t \rightarrow \infty, (v^{(t)}, h^{(t)})$, the samples of $p(v, h)$ will be accurate. Nevertheless, one can further speed up the process by using "contrastive divergence" (CD) which involves initializing the Markov chain with a training example $v \epsilon V$ and running the Markov chain for k (often with $k = 1$) iterations. In the context of the derivative defined in (7), CD can be roughly outlined as the following:

1) Replace the first term (expectation over all input samples) with a single sample.
2) For the second term, run the Markov chain for fixed k steps.

More explicitly, CD-1 can be described as the following:

1) Take a training sample v, compute the probabilities of the hidden units and sample a hidden activation vector h from this probability distribution.
2) Compute the outer product of v and h and call this the positive gradient.
3) From h, sample a reconstruction v' of the visible units, then resample the hidden activations h' from this. (Gibbs sampling step)
4) Compute the outer product of v' and h' and call this the negative gradient.
5) Let the weight update to $w_{i,j}$ be the positive gradient minus the negative gradient, times some learning rate (see equation (4)).

$$\Delta w_{i,j} = \epsilon(v h^T - v' h'^T) \tag{13}$$

The bias updates for the visible and hidden layers respectively can be defined by these expressions:

$$\begin{aligned} \Delta v_b &= \epsilon(v - v') \\ \Delta h_b &= \epsilon(h - h') \end{aligned} \tag{14}$$

CD-k is a generalization of CD-1 by repeating the sampling process in step three k times, though in practice it has been found that 1 iteration (CD-1) works fairly well in "[ensuring] that hidden features retain most of the information in the data vector" [8]. One extension of contrastive divergence is persistent contrastive divergence (PCD). PCD enhances parameter updates by resuming iterations of the involved Markov chain from the last iteration instead of rerunning the Markov chain [9].

IV. Recent Applications

One of the more recent applications of RBMs is their use in stacks. By stacking RBMs, hidden layers can be trained on top of other hidden layers. After training multiple hidden layers, these layers can be unrolled to form a deterministic deep neural network. A deep neural network of such variety can then be finely tuned with common training methods such as backpropagation. In 2006, Hinton, using a similar structure alluded to here, trained a deep neural network, which achieved an error rate of 1.39%, which outperformed a SVM with RBF kernel, which were considered state of the art at the time [2].

V. Experimental Results

We implement our own RBM in Python with the NumPy library [10]. We train a RBM with 784 visible units and 500 hidden units on the MNIST handwritten digit dataset [11], a standard benchmark for image classification. The learning algorithm used is CD-1, which is the simplest of the CD methods but tends to work very well in practice. We initialize the weights of the RBM randomly by sampling them from a normal distribution with zero mean and 0.01 standard deviation. Next, we apply CD-1 to all 60,000 training images for 15 epochs with a learning rate 0.005.

In Fig. 2, we visualize the trained weights of the RBM. Each hidden node is connected to all 784 visible nodes and thus we can display the weights corresponding to a hidden node as an image with the same dimension as the input digits. As shown by the figure, the filters learned are interesting, with a diverse variety of different edge and blob detectors.

Next in Fig. 3, we show the samples generated by the RBM when given test digits as input. The first column shows the

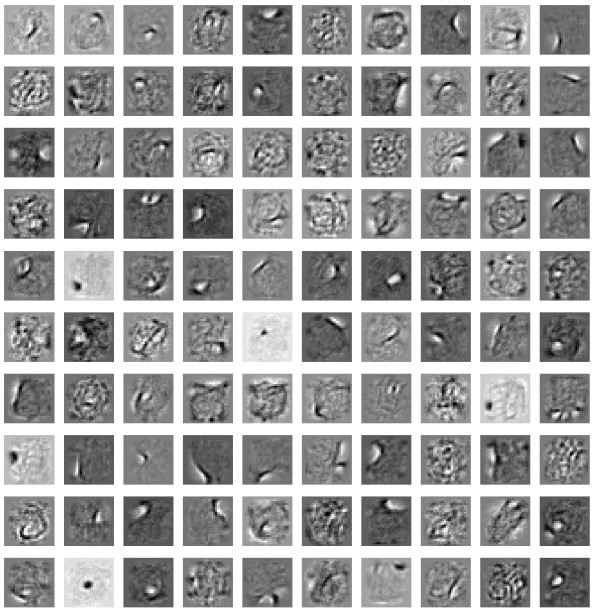

Fig. 2. Filters corresponding the weights of the first 100 hidden nodes.

original digits while the subsequent columns shows samples outputted by the model after an interval of 1000 steps of Gibbs sampling (to ensure proper mixing of the Markov chain). The images in Fig. 3 show that RBM has properly learned the distribution for input digits and that samples are drawn from that distribution properly.

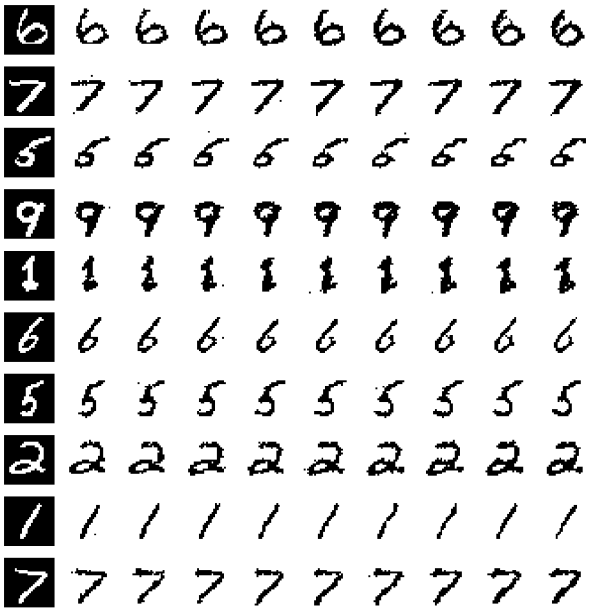

Fig. 3. Samples generated by RBM. The first column shows original test digit sample. Subsequent columns show digits sampled from RBM with an interval of 1000 steps of Gibbs sampling.

Lastly, we explore the usefulness of RBMs in learning good representations of the input data and performing dimensionality reduction. As a preprocessing step prior to classification, we treat our trained RBM as a deterministic single layer neural network and use it to transform the input digits into a 500 dimensional feature vector. This feature vector is then used for supervised training of a logistic regression classifier. Without an RBM layer, the misclassification rate on a test set of 10,000 digits is 8.20%. With the addition of the RBM layer, the error rate of the logistic regression classifier decreases to 3.07%, an relative improvement of over 150%.

VI. Conclusion

As demonstrated in previous research and in our explorations, RBMs are quite useful for classification tasks. With their relatively simple training procedures and given recent innovations in the use of RBMs, RBMs have a promising future uses in deep neural networks.

References

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines*. *Cognitive science*, 9(1):147–169, 1985.

[2] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[3] Maria Angelica Cueto, Jason Morton, and Bernd Sturmfels. Geometry of the restricted boltzmann machine. *arXiv preprint arXiv:0908.4425*, 2009.

[4] John J Hopfield and David W Tank. neural computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[5] Radford M Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.

[6] Carsten Peterson. A mean field theory learning algorithm for neural networks. *Complex systems*, 1:995–1019, 1987.

[7] Miguel A Carreira-Perpinán and Geoffrey E Hinton. On contrastive divergence learning. *AISTATS 2005*, page 33, 2005.

[8] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[9] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[10] Travis E Oliphant. *A Guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[11] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 141:142, 2012.