A COMPARATIVE STUDY OF DIFFERENT GRADIENT APPROXIMATIONS FOR RESTRICTED BOLTZMANN MACHINES

LIHONG JI

Thesis supervisor: ENRIQUE ROMERO MERINO (Department of Computer Science)

Thesis co-supervisor: FERRAN MAZZANTI CASTRILLEJO (Department of Physics)

Degree: Master Degree in Artificial Intelligence

Thesis report

School of Engineering
Universitat Rovira i Virgili (URV)

Faculty of Mathematics
Universitat de Barcelona (UB)

Barcelona School of Informatics (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

29/06/2023

Contents

List of Figures

List of Tables

List of Algorithms

A Comparative Study of Different Gradient Approximations for Restricted Boltzmann Machines

By Lihong Ji

Abstract

This project consists of the theoretical study of Restricted Boltzmann Machines(RBMs) and focuses on the gradient approximations of RBMs. RBMs suffer from the dilemma of accurate learning with the exact gradient. Based on Contrastive Divergence(CD) and Markov Chain Monte Carlo(MCMC), CD-k, an efficient algorithm of approximating the gradients, is proposed and now it becomes the mainstream to train RBMs. In order to improve the algorithm efficiency and mitigate the bias existing in the approximation, many CD-related algorithms have emerged afterwards, such as Persistent Contrastive Divergence(PCD) and Weighted Contrastive Divergence(WCD). In this project the comprehensive comparison of the gradient approximation algorithms is presented, mainly including CD, PCD, WCD. The experimental results indicate that among all the conducted algorithms, WCD has the fastest and best convergence for parameter learning. Increasing the Gibbs sampling time and adding a persistent chain in CD-related can enhance the performance and alleviate the bias in the approximation, also taking advantage of Parallel Tempering can further improve the results. Moreover, the cosine similarity of approximating gradients and exact gradients is studied and it proves that CD series algorithms and WCD series algorithms are heterogeneous. The general conclusions in this project can be the reference when training RBMs.

Acknowledgement

First, my sincere appreciation to my supervisors Enrique Romero and Ferran Mazzanti. I am deeply grateful for your patience and it has always been happy and inspiring to talk with you. Although many things happened throughout the entire period, persevering through it all has provided me with invaluable experience. Then I would like to express my deepest gratitude to all those who have supported and guided me through the process of completing this master thesis. Finally I am really grateful to the scholars who have made significant contributions in this field. It is their achievements that have guided me and enabled me to make continuous progress.

1 Introduction

1.1 Background

In recent years, the neural networks have achieved great success in machine learning fields. With different frameworks, the neural networks can solve a vast variety of specific tasks, such as image classification and segmentation, speech recognition, natural language understanding etc.. When the transformer framework is proposed in 2017[12], there is a tendency that people implement this framework as the priority because of the high efficiency brought by parallel computation framework design and superior capability of capturing the potential correlation in the sequence. Trained in unsupervised learning method, GPT series models demonstrate the exploding power in generative tasks, so people resume to see the great potential of generative neural network, and devote more attention to this field.

Backtracking to the origin, Restricted Boltzmann Machine(RBM) is one of the pioneer generative neural networks, which is composed of two layers, one visible layer and one hidden layer. The basic framework of RBM, independent nodes constitute layers and layers are linked by weights, is still guiding the advance of modern generative neural models. Moreover, RBMs can be stacked to construct Deep Belief Network, which is the prototype of "perceptron". As an energy-based model, RBMs get the insight from physics fields, and it is designed to distribute less energy to the observed cases so as to permit higher probability for the desirable cases. RBMs perform well in modeling complex, high-dimensional input models. Nevertheless, training RBMs is really hard because of the intractable partition function term, which would make computing the exact gradient impossible when the data size is large.

The emergence of Contrastive Divergence algorithm mitigates the tough training problem by estimating the intractable terms via Markov Chain Monte Carlo. Specifically, Gibbs sampling is utilized in RBMs training to approximate $P(v)$, and it improves the efficiency of the model learning significantly. However, proved by the evidence[1][5], CD is a biased algorithm. To nerf the effect of bias and analyze the cause of the bias, people have devoted great efforts in this field, and model learning algorithms with less bias are proposed such as Persistent Contrastive Divergence, Weighted Contrastive Divergence and so on. Particularly some general conclusions are still playing important roles in current deep learning training.

1.2 Motivation

As CD is a biased learning strategy, it is necessary to study how the bias affects the precision of the estimation. Also learning about the improved approximation algorithms can give us an insight of alleviating the gradient shift, and we can obtain a progressive understanding of RBM training. So far, as for the binary RBMs, there are several widely-used approximating training algorithms: Contrastive Divergence, Persistent Contrastive Divergence, Weighted Constrictive Divergence, Parallel Tempering as so on. Doing the comparative study on these algorithms can produce a comprehensive analysis. According to the analysis results, people can take advantage of these achievements so that they could determine which is the optimal algorithm for RBMs to use while facing the trade-off between the efficiency and the precision.

As a typical generative neural network, RBMs are worthy of learning in order to get known about the basic heuristic ideas of deep learning. Most importantly, RBM is a strongly probability-supported model that means this sort of model has reliable explanability. That is why probabilistic graphical models are irreplaceable, and still applied in many fields.

Understanding RBMs comprehensively requires the knowledge of stochastic process, and also stochastic sampling is an important part in Bayesian Machine Learning. Personally, this is the fantastic opportunity

for me to fullfill the blank of this part. At the same time, Monte Carlo approximation, importance sampling and Markov Chain Monte Carlo and other related methods are applied widely in many fields. In order to gain a more general perspective of quantitative analysis, it is crucial to learn this part. No question massive model training would be concluded in the experiment session, so it is a great time to learn more practical training techniques and apply them in practice so that we can have impressive and unique understanding of these techniques, which would help us to train much larger models in the future.

1.3 Objectives

First, we need to implement the approximation algorithms so that we have the freedom to adjust the algorithm to realize more exploration. The algorithms are implemented in Python, and the algorithms are constructed mainly based on Numpy. The target gradient algorithms are: Contrastive Divergence, Persistent Contrastive Divergence, Weighted Contrastive Divergence, Weighted Persistent Contrastive Divergence, Parallel Tempering as well as the exact gradient computation.

Secondly, we are supposed to manage the RBMs training. That means the proper hyperparameters and model parameters need to be determined for the good training results, which requires large amount of computation. The evaluation metrics are Kullback–Leibler Divergence, Negative Log-Likelihood, Probability Sum and Entropy. When calculating the probability sum, the distributed probability of each training data point would be collected up for analyzing the bias of approximation.

Thirdly, we would compare the approximation results with those via exact gradients, and we can observe how the bias varies during the training. Based on the results, the further exploration of the approximated gradients and the exact gradients would be conducted. Through these, we can know the commonalities and differences of all the algorithms so that we could determine the optimal algorithm for different conditions.

2 State of the Art

The exact gradients of the RBM loglikehood are impossible to compute when the size of input data is large, because the partition function is included in the process of calculating the exact gradient. The computation of the partition function reaches an exponential level making the model parameter learning is really tough, thus when RBM was proposed firstly[10] it did not receive much attention. Till the Contrastive Divergence algorithm was proposed by Geoffrey Hinton[7], the CD algorithm provides a way of approximating the gradient with Gibbs sampling thus reducing the computing burden of RBM. It is pointed out that sampling for 1 time in the Markov chain is sufficient for the model learning. This series of methods is designated as CDK where K is the sampling time in the Markov Chain. Based on the idea of CD, using a persistent Markov chain during the whole training process rather than establishing a new chain in each training epoch, Persistent Contrastive Divergence was proposed and appeared to give out better performance in the model parameter learning[11], but if we want to manage to train RBMs with PCD, some other training techniques are required, the reason is that we assume that Markov chain stays to the stationary distribution if the learning rate is sufficiently small, and the model changes only slightly between parameter updates[6]. Since the RBM gets the insight from Boltzmann Machine in the physics field[4],[3], there are lots of potential helpful ideas to take advantage to improve the RBMs. Following this point, Parallel Tempering is transferred to sample for better particles by exchanging the states in different temperature chains[2]. Faster and better RBM convergence can be obtained using PT sampling compared to standard sampling methods. Similarly, Langevin Monte Carlo can be integrated with Gibbs sampling to get better samples for Gaussian-Bernoulli RBMs(GRBMs), and this sort of method is named as Gibbs-Langevin sampling which is proposed in 2022[8]. In GRBMs, the energy function is modified to deal with the continuous data as equation(1).

$$E_\theta(v,h) = \frac{1}{2}(\frac{v-\mu}{\sigma})^T(\frac{v-\mu}{\sigma}) - (\frac{v}{\sigma^2})Wh - b^Th \tag{1}$$

Based on CDK algorithms, there thrive lots of good extensions, but CDK technically is a biased algorithm, which means this sort of gradient approximation would assign unbalanced probabilities to the training data points, and the ideal case is that the probability distribution of data points is consistent with a uniform distribution. The bias depends on the mixing rate of the Markov chain, and mixing slows down with increasing model parameters, and potentially the bias would lead to the divergence. Thus the residual term converges to 0 as $K \to \infty$. The bias part contained in the loglikelihood gradient is expressed quantitatively as following equation, and the bias part is $\mathbb{E}_{P(v^k|v^0)}[\bigtriangledown_\theta \log P(v^k)]$ [1].

$$\begin{aligned}\bigtriangledown_\theta \log P(v^0) = &-\sum_h P(h|v^0)\ \bigtriangledown_\theta E(v^0,h) \\ &+ \mathbb{E}_{P(v^k|v^0)}[\sum_h P(h|v^k)\ \bigtriangledown_\theta E(v^k,h)] + \mathbb{E}_{P(v^k|v^0)}[\bigtriangledown_\theta \log P(v^k)]\end{aligned}$$

According to the mathematical expression of the bias term, Bengio[1] and Fischer[5] propose two analytical bounds of the CD bias respectively as equation(2) and (3).

$$\mathbb{E}_{P(v^k|v^0)}[\bigtriangledown_\theta \log P(v^k)] \leq 2^m(1-2^m 2^n sig(-\alpha)^m sig(-\beta)^n)^k \tag{2}$$

The visible dimension is denoted as m, the hidden dimension is n, sig stands for sigmoid function $\frac{1}{1+e^{-x}}$, k is the Gibbs sampling time, α is $\alpha = \max_j(\sum_i |w_{ij}| + |b_j|)$, and β is $\beta = \max_i(\sum_j |w_{ij}| + |c_j|)$ where b and c stand for the bias of the visible layer and that of the hidden layer respectively.

$$\mathbb{E}_{P(v^k|v^0)}[\bigtriangledown_\theta \log P(v^k)] \leq (1 - e^{-(m+n)\triangle})^k \tag{3}$$

Specifically,

$$\triangle = \max(\max_{l\in(1,m)} \vartheta_l, \max_{l\in(1,n)} \zeta_l)$$

where

$$\vartheta_l = \max(|\sum_i^n I_{(w_{il}>0)} w_{il} + \alpha_l|, |\sum_i^n I_{(w_{il}<0)} w_{il} + \alpha_l|)$$

and

$$\zeta_l = \max(|\sum_j^m I_{(w_{lj}>0)} w_{lj} + \beta_l|, |\sum_j^m I_{(w_{lj}<0)} w_{lj} + \beta_l|)$$

Both of the two defined bounds have the relevance of controlling the absolute values of RBM parameters. Intuitively, the smaller the bias of the loglikelihood gradient estimation, the higher the chances to converge to a maximum loglikelihood solution quickly. So techniques that suppress the increasing of the weights would help to nerf the bias. Moreover, observing the bound expression, the bound would increase with the number of RBM variables or decreasing of the increasing value of k, and this could lead to potential larger bias in the training results. From this point, it quantitatively illustrates why CD10 outperforms CD1. Based on these conclusions, there emerges a training schedule to avoid much bias, and the idea is to enlarge k dynamically during the training when weights are increased[1].

The practical techniques of training RBMs are given here. First, considering the bias brought by the increasing of the weights, the straight way is to stop early at a proper checkpoint. Also, as mentioned before, using weight decay or increasing k dynamically can lead to a better convergence. Learning rate decay is an effective training technique to prevent the divergence, but the shortcoming is that choosing a proper learning rate decay schedule is hard. Adding more hidden units can bring lower risk of divergence.

Facing the dilemma of unavoidable bias brought by the CD algorithm, Weighted Contrastive Divergence is proposed by Romero and Mazzanti in 2019[9]and this approximation algorithm significantly alleviates the imbalance of the training probability distribution, and also reaches a faster and better convergence compared to the former algorithms. Compared to the standard Contrastive Divergence, there is a modification of the model expectation term in Weighted Contrastive Divergence. The assumption in CD is that each training data point is assigned with the same weight, $1/DataSize$, during updating, but in WCD the weight is defined as the importance of the corresponding $P(v)$. Similarly with Persistent Contrastive Divergence, Weighted Persistent Contrastive Divergence is also proposed by adding a persistent chain to WCD.

In this project, we would focus on Binary RBMs and study the gradient approximation algorithms as well as comparing these approximating gradients with the exact gradients.

3 Theoretical RBMs

3.1 Energy Based Models

Energy based models are a class of networks that define energy functions over inputs. The energy function specifies a probability distribution and the goal is to learn a set of parameters of energy function which assign lower energies to the desirable data, and higher energies to the rest. Based on the different cases, the energy function formulation could vary a lot in order to fit in the situation. In many cases, similar to Boltzmann distributions, the probability is an exponential function of the energy. Since the ultimate target is to obtain a normalized probability, all the possible cases would be accounted to calculate the sum of the individual contributions, and this concept is named as *partition function* Z. In summary, $p(v)$ is defined as following:

$$P(v) = \frac{e^{-Energy(v)}}{Z}$$

To enhance the performance of energy models, a series of latent variables h are introduced so that the energy models are capable of discovering and fitting in the extremely complex patterns. From now on, v will be referred as visible variable and h as hidden variable or latent variable. Then the objective $P(v)$ turns to the marginal probability of the joint probability, so the important elements of energy models are defined as following:

$$P(v,h) = \frac{e^{-Energy(v,h)}}{Z}$$

$$P(v) = \frac{\sum_h e^{-Energy(v,h)}}{Z}$$

$$Z = \sum_{v,h} e^{-Energy(v,h)}$$

Recall the aim is to push lower energies to the achieved observation, so the optimization is equal to maximizing $P(v)$, but instead of dealing with $P(v)$ straightly, the object is transferred to $log(P(v))$ because it makes the whole computing process much more efficient, which is named as Log Likelihood and used significantly in machine learning optimization. The original probability can be denoted as following, and $Energy(x)$ is simplified as $E(x)$:

$$\log P(v) = \log \sum_h exp(-E(v,h)) - \log \sum_{v,h} exp(-E(v,h))$$

With this formulation, the gradient of the model parameters are analytically expressed, then the parameters can be updated step by step following the gradient direction. This is the basic idea for energy models to learn from the data or environment. The specific derivation is displayed below, where the obtained variable information is denoted as $\hat{v}$, the possible cases are denoted as v, and the model parameters are denoted as θ.

$$
\begin{aligned}
\frac{\partial \log P(v)}{\partial \theta} &= \frac{\partial}{\partial v}(\log \sum_h exp(E(\hat{v}, h) - \log \sum_{v,h} exp(E(v, h))) \\
&= -\frac{1}{\sum_h exp(-E(\hat{v}, h))} \sum_h exp(-E(\hat{v}, h)) \frac{\partial E(\hat{v}, h)}{\partial \theta} \\
&+ \frac{1}{\sum_{v,h} exp(-E(v, h))} \sum_{v,h} exp(-E(v, h)) \frac{\partial E(v, h)}{\partial \theta} \\
&= -\frac{1}{P(\hat{v})} \sum_h P(\hat{v}, h) \frac{\partial E(\hat{v}, h)}{\partial \theta} + \sum_{v,h} P(v, h) \frac{\partial E(v, h)}{\partial \theta} \\
&= -\sum_h P(h \mid \hat{v}) \frac{\partial E(\hat{v}, h)}{\partial \theta} + \sum_v P(v) \sum_h P(h \mid v) \frac{\partial E(v, h)}{\partial \theta} \\
&= -\mathbb{E}_{P(h|\hat{v})}[\frac{\partial E(\hat{v}, h)}{\partial \theta}] + \mathbb{E}_{P(v)}[\mathbb{E}_{P(h|v)}[\frac{\partial E(v, h)}{\partial \theta}]]
\end{aligned}
$$

The expression of log likelihood gradient is finally represented as two sums of expectations. The first part is called data expectation also known as positive phase, which contains the information from the achieved data. The second part is called model expectation also known as negative phase. Since this part covers all the theoretical possibility, it makes the exact computation intractable and in most of cases it is impossible to be solved in limiting time. The reason is that the analytical expression of $P(v)$ involves the partition function Z. Therefore, to overcome the computation dilemma, the sampling method is utilized for higher efficiency. In this document, only RBM is involved and thanks to the properties of RBM, the process is simplified significantly. The specific introduction and derivation would be demonstrated in the next chapter.

3.2 RBM Representation

Restricted Boltzmann Machine, as an energy based model, has the general properties of energy models, and meanwhile according to its own unique merits this sort of model requires relatively less computing cost. The energy function of RBM is defined as (4):

$$E(v, h) = -(vWh^T + \alpha v^T + \beta h^T) \tag{4}$$

In the energy definition, the model parameter is composed of three parts: the linking weight is denoted as W, the visible unit bias as α, and the hidden unit bias as β. With the proper expression, the visible layer and hidden layer are combined and linked as a complete system, and capable of describing the energy state and transforming it to a probability.

From the perspective of Probabilistic Graphical Model(PGM), this sort of model is considered as a Markov Random Field with hidden nodes. The word "Restricted" in RBM means a regularization that separates the observed variables as well as hidden variables, equally meaning the nodes in the same layer are independent. Here are the statements prepared for the mathematical expression, the visible and hidden variables are denoted as x, the max clique as C_i, the potential function as $\phi_i(X_{C_i})$ and partition function as Z. Specifically, in RBM the potential function is defined as the same as the exponential form of energy function. Finally, $P(x)$ can be expressed as following, and the final form of $P(x)$ is named as

Boltzmann Distribution or Gibbs Distribution:

$$\begin{aligned} P(x) &= \frac{1}{Z}\prod_{i}^{k}\phi_i(X_{C_i}) \\ &= \frac{1}{Z}\prod_{i}^{k} exp\left\{-E(X_{C_i})\right\} \\ &= \frac{1}{Z} exp\left\{-\sum_{i=1}^{k} E(X_{C_i})\right\} \\ &= \frac{1}{Z} exp\left\{-E(x)\right\} \end{aligned}$$

After replacing potential function with energy function, the analytical expression of $P(v, h)$ is obtained, which is also designated as probability density function (5).

$$P(v,h) = \frac{1}{Z} exp\left\{-(vWh^T + \alpha v^T + \beta h^T)\right\} \tag{5}$$

The graph of RBM is shown as Figure 1.

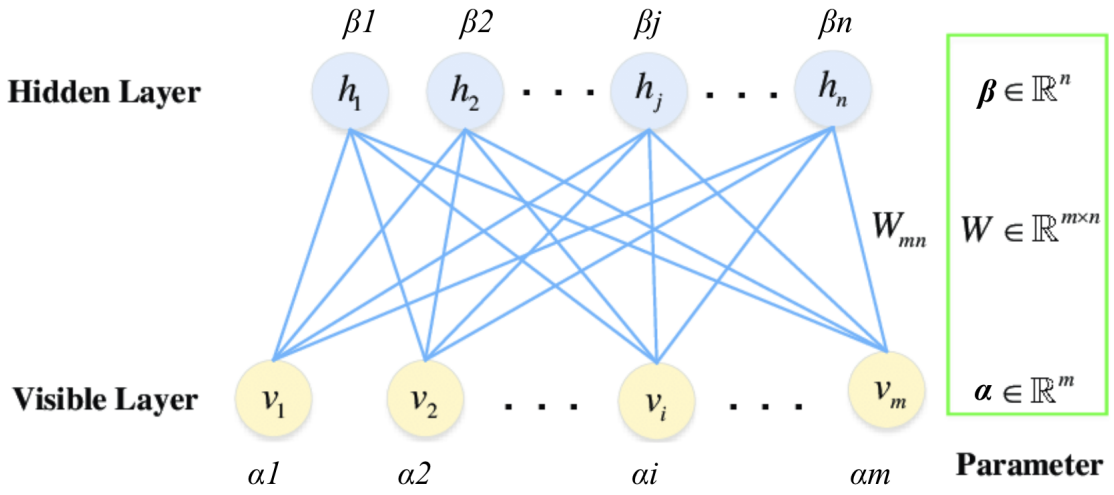


Figure 1 – The graph of Restricted Boltzmann Machine

3.3 RBM Inference

In the inference part, there are in total 3 objects: $P(h \mid v)$, $P(v \mid h)$ and $P(v)$. These 3 probabilities are the basic elements for the next training work, and as mentioned before, according to the superior property brought by the RBM energy function, the analytical derivation turns be much easier to deal with and the final expression extremely shrinks the computation cost.

Since the derivation processes of $P(h \mid v)$ and $P(v \mid h)$ are equivalent, here only demonstrate how $P(h \mid v)$ comes. At the beginning, important variables are defined as following: the binary visible unit is denoted as v, and the binary hidden unit is denoted as h. m is the number of hidden units, and n is the number of visible units.

In order to demonstrate the complete derivation, we take advantage of visible variables $\in [0, 1]$ since this

sort of input is widely used for most of the cases. In contrary, the inference of visible input $\in [-1, 1]$ would be given out in section 5:Experimental Analysis.

First, the initial form of $P(h \mid v)$ is expanded as (6).

$$P(h \mid v) = \prod_{l=1}^{m} P(h_l \mid v) \tag{6}$$

The values of h are only 0 or 1, then the case $h = 1$ is computed first.

$$\begin{aligned} P(h_{l=1} \mid v) = P(h_{l=1} \mid h_{-l}, v) &= \frac{P(h_{l=1}, h_{-l}, v)}{P(h_{-l}, v)} \\ &= \frac{P(h_{l=1}, h_{-l}, v)}{P(h_{l=1}, h_{-l}, v) + P(h_{l=0}, h_{-l}, v)} \end{aligned} \tag{7}$$

Since the numerator and denominator are similar, the goal is to compute numerator $P(h_{l=1}, h_{-l}, v)$ with the help of $E(h, v)$, then the l^{th} hidden unit is picked out, and $E(v, h)$ is transformed to (8). In convenience for derivation writing, in (8) the non-containing h_l are integrated and denoted as A, and the rest part is denoted as B. Hence, $E(v, h)$ is transformed to $-(A + h_l B)$.

$$\begin{aligned} E(v, h) &= -\sum_{i=1, i\neq l}^{m} \sum_{j=1}^{n} h_i w_{ij} v_j - \sum_{j=1}^{n} \alpha_j v_j - \sum_{i=1, i\neq l}^{m} \beta_i h_i - h_l \sum_{j=1}^{n} w_{ij} v_j - \beta_l h_l \\ &= -\underbrace{\left(\sum_{i=1, i\neq l}^{m} \sum_{j=1}^{n} h_i w_{ij} v_j + \sum_{j=1}^{n} \alpha_j v_j + \sum_{i=1, i\neq l}^{m} \beta_i h_i \right)}_{A} - h_l \underbrace{\left(\sum_{j=1}^{n} w_{ij} v_j + \beta_l \right)}_{B} \end{aligned} \tag{8}$$

Back to the numerator $P(h_{l=1}, h_{-l}, v)$ and denominator $P(h_{l=1}, h_{-l}, v) + P(h_{l=0}, h_{-l}, v)$, these two large expressions could be replaced by the former alphabets following the RBM energy function, then $P(h_{l=1} \mid v)$ is represented as (9).

$$\begin{aligned} P(h_{l=1} \mid v) &= \frac{\frac{1}{Z} exp\{A + B\}}{\frac{1}{Z} exp\{A + B\} + \frac{1}{Z} exp\{A\}} \\ &= \frac{1}{1 + exp\{-B\}} \\ &= \frac{1}{1 + exp\left\{-(\sum_{j=1}^{n} w_{ij} v_j + \beta_l)\right\}} \end{aligned} \tag{9}$$

In conclusion, the general formula for the conditional probability is (10), and similarly $P(v \mid h)$ is (11). It can be seen that the partition function is not involved so it could be really light to compute the conditional probabilities.

$$P(h \mid v) = \frac{1}{1 + exp\{-(vW + \beta)\}} = Sigmoid(vW + \beta) \tag{10}$$

$$P(v \mid h) = \frac{1}{1 + exp\{-(hW^T + \alpha)\}} = Sigmoid(hW^T + \alpha) \tag{11}$$

As for $P(v)$, it is much tougher to do the exact calculation, and also this is the computing barrier for RBMs, since it is unavoidable to face partition function involving in the derivation. Although intractable in most time, it is still necessary to figure out a shortcut for exact inference of $P(v)$, then the newly proposed methods could be analyzed and evaluated.

According to the energy function, and $P(v)$ is the marginal probability of $P(v, h)$, it could be written as (12).

$$\begin{aligned}
P(v) &= \sum_{h_1} ... \sum_{h_m} \frac{1}{Z} exp\left\{h^T W v + \alpha^T v + \beta^T h\right\} \\
&= \frac{1}{Z} exp\left\{\alpha^T v\right\} \sum_{h_1} ... \sum_{h_m} exp\left\{h^T W v + \beta^T h\right\} \\
&= \frac{1}{Z} exp\left\{\alpha^T v\right\} \sum_{h_1} ... \sum_{h_m} exp\left\{\sum_{i=1}^{m} (h_i w_i v + \beta_i h_i)\right\} \\
&= \frac{1}{Z} exp\left\{\alpha^T v\right\} \sum_{h_1} exp\left\{h_1 W_1 v + \beta_1 h_1\right\} ... \sum_{h_m} exp\left\{h_m W_m v + \beta_m h_m\right\} \\
&= \frac{1}{Z} exp\left\{\alpha^T v\right\} (1 + exp\left\{w_1 v + \beta_1\right\}) ... (1 + exp\left\{w_m v + \beta_m\right\}) \\
&= \frac{1}{Z} exp\left\{\alpha^T v\right\} exp\left\{\log(1 + exp\left\{w_1 v + \beta_1\right\})\right\} ... exp\left\{\log(1 + exp\left\{w_m v + \beta_m\right\})\right\} \\
&= \frac{1}{Z} exp\left\{\alpha^T v + \sum_{i=1}^{m} \log(1 + exp\left\{w_i v + \beta_i\right\})\right\} \\
&= \frac{1}{Z} exp\left\{\alpha^T v + \sum_{i=1}^{m} softplus(w_i v + \beta_i)\right\}
\end{aligned} \tag{12}$$

To simplify the formulation, it is better to vectorize the weights and variables, as (14). So far, the computing cost has been decreased significantly since the double summation has been transformed to only one summation. However, as mentioned before, the partition function is compulsory to compute if $P(v)$ is the final target, and this is the reason why the idea of rejection/importance sampling is introduced in the learning session.

$$P(v) = exp\left\{\alpha^T v + \log(1 + exp\left\{vW + \beta\right\})\right\} \tag{13}$$

$$= exp\left\{\alpha^T v + softplus(vW + \beta)\right\} \tag{14}$$

Now the complete derivations of 3 target probabilities have been demonstrated, and these formulations would play vital roles in the next part, RBM learning.

3.4 RBM Learning

Technically, it is really hard to train Restricted Boltzmann Machines since the partition function is involved in the model expectation which makes the exact gradient computation definitely expensive, but in contrary the data expectation can be efficiently computed. There are kinds of Markov Chain Monte Carlo(MCMC) based approximation methods proposed to alleviate this intractable problem, such as Contrastive Divergence, known as CDk, but in this chapter the aim is to analyze RBM learning process theoretically and accurately. The sampling-based methods would be introduced, analyzed and tested comprehensively in the next chapter.

Whenever learning parameters from the environment, the target is always trying to make the model fit in the observed and collected data. With the help of available regularization, the model is capable of avoiding being overfitted because in most time the distribution of the training data is not able to represent the true distribution. Back to RBMs, as an unsupervised model, the target is to maximize $P(v)$, equally $logP(v)$. According to the gradient derivation in the previous chapter, the partial gradient of the log probability $logP(v)$ with the model parameter θ is actually composed of two parts: data expectation and model expectation, and both parts require the partial gradient of the energy function with θ, and that is very easy to compute, as following:

$$\frac{\partial E(v,h)}{\partial w_i j} = -h_j v_i \tag{15}$$

$$\frac{\partial E(v,h)}{\partial \alpha_i} = -v_i \tag{16}$$

$$\frac{\partial E(v,h)}{\partial \beta_i} = -h_j \tag{17}$$

Based on the obtained results, and the v and h are actually binary variables,the first part, data expectation, of partial probabilities of $logP(v)$ with respect of W, α, β are figured out as following:

$$\begin{aligned}\mathbb{E}_{P(h|\hat{v})}[\frac{\partial E(\hat{v},h)}{w_{ij}}] &= -\sum_{h=0,1} P(h \mid \hat{v}) h_j \hat{v}_i \\ &= -\hat{v}_i P(h_j = 1 \mid \hat{v}) \end{aligned} \tag{18}$$

$$\begin{aligned}\mathbb{E}_{P(h|\hat{v})}[\frac{\partial E(\hat{v},h)}{\alpha_i}] &= -\sum_{h=0,1} P(h \mid \hat{v}) \hat{v}_i \\ &= -\hat{v}_i \end{aligned} \tag{19}$$

$$\begin{aligned}\mathbb{E}_{P(h|\hat{v})}[\frac{\partial E(\hat{v},h)}{\beta_j}] &= -\sum_{h=0,1} P(h \mid \hat{v}) h_j \\ &= -P(h_j = 1 \mid \hat{v}) \end{aligned} \tag{20}$$

As for the second part, the model expectation is still similar to deal with, but there are two different points from the former derivation: one more step to compute the expectation of $P(v)$, and v includes the all theoretical cases, which is different from the former $\hat{v}$. So the final formulas are displayed as

following:

$$\mathbb{E}_{P(v)}[\mathbb{E}_{P(h|v)}[\frac{\partial E(v,h)}{w_{ij}}]] = -\sum_{v} P(v)P(h_j = 1 \mid v)v_i \tag{21}$$

$$\mathbb{E}_{P(v)}[\mathbb{E}_{P(h|v)}[\frac{\partial E(v,h)}{\alpha_i}]] = -\sum_{v} P(v)v_i \tag{22}$$

$$\mathbb{E}_{P(v)}[\mathbb{E}_{P(h|v)}[\frac{\partial E(v,h)}{\beta_j}]] = -\sum_{v} P(v)P(h_j = 1 \mid v) \tag{23}$$

Integrate the two parts of expectation, the partial gradients of $logP(v)$ with respect to W, α and β are finally achieved.

$$\frac{\partial \log P(v)}{\partial W_{ij}} = -\hat{v}_i P(h_j = 1 \mid \hat{v}) + \sum_{v} P(v)P(h_j = 1 \mid v)v_i \tag{24}$$

$$\frac{\partial \log P(v)}{\partial \alpha_i} = -\hat{v}_i + \sum_{v} P(v)v_i \tag{25}$$

$$\frac{\partial \log P(v)}{\partial \beta_j} = -P(h_j = 1 \mid \hat{v}) + \sum_{v} P(v)P(h_j = 1 \mid v) \tag{26}$$

So far, the exact gradient derivation of the log likelihood of $P(v)$ has been completed. With the formulas, it is theoretically accessible to compute gradients anytime and update model parameters step by step with a reasonable learning rate till the convergence, and this sort of learning approach is called gradient ascent. Whereas, observing the formula it is obvious to notice that computing the model expectation part is definitely an intractable operation because it reaches an exponential level even if v is a binary variable. Therefore, the idea of sampling is utilized to relieve the computing pressure. The details would be stated in the next chapter.

4 Training RBMs

Restricted Boltzmann Machine training has been developed significantly due to the increasing of the computational power as well as the progress of learning strategy optimization. Since Contrastive Divergence was proposed in 2002[7], this sort of learning strategy has a great impact on training RBMs, and then it becomes the mainstream and prospers a lot of CD-based learning approaches, which take advantage of Markov Chain Monte Carlo, specifically Gibbs Sampling is used in RBM training. The main idea of RBM training is to assign less energy to the obtained data distribution, in other words, adjust the model parameters to maximize the probability of the training data denoted as $P(v)$. With Gibbs Sampling the gradient can be estimated to modify the model parameters, and the specific method is called gradient ascent, where the parameters are adjusted following the gradient direction in a short distance during each epoch.

Based on gradient ascent, in order to make the training stage more productive, the momentum is introduced to accelerate the training process. The idea of momentum is to store and add up the gradient at a default percentage in each epoch, usually as 0.9. In each updating epoch, the model parameters are modified with the product of the momentum and learning rate. This sort of technique is widely utilized in the neural network optimization, because it has 3 main advantages: 1.Momentum helps accelerate gradient descent by adding the previous gradients to the current one. This helps gradient ascent converge faster in relevant directions. 2.Momentum helps gradient ascent avoid local minima by "pushing" in the same direction over many iterations. 3.Momentum helps reduce oscillations in the optimization path by smoothing out the update directions over time. This leads to a more direct path to the minimum.

Although the Contrastive Divergence series approximations do bring much convenience and efficiency to the training session, this series of methods still have drawback in gradient approximation, because there exists approximation bias if the sampling process are not long enough. Nevertheless, in practice, it is sufficiently effective to set up the step of Gibbs Sampling as 1. In this chapter, the comprehensive illustration of current optimization methods would be introduced, from exact gradient to Contrastive Divergence, from Gibbs Sampling to Parallel Tempering. Through the whole sessions, we could feel and understand how RBM training advances.

4.1 Stochastic Sampling in RBMs

4.1.1 Markov Chain & Monte Carlo Approximations

Markov Chain Markov chain can be understood as a discrete time series stochastic process. Markov chain inherits the property of being "memorylessness", and can be expressed as the current state is only related with the previous state and independent with the other states. The mathematical expression is $P(X_{i+1} \mid X_1, .., X_i) = P(X_{i+1} \mid X_i)$, where the time points are denoted as $1, 2, .., i, i+1$.

One of the best properties of the Markov chain is that whenever an arbitrary Markov chain satisfies specific conditions then after a long time run and regardless of starting states, the chain would always converge to the same,unique probability distribution, which is called stationary distribution. If the starting distribution is denoted as $\mu^{(0)}$, the distribution at moment t is denoted as $\mu^{(t)}$, and the transition matrix is denoted as P, then $\mu^{(t)}$ is equal to $\mu^{(0)}P^t$. The stationary distribution π is defined as satisfying $\pi^t{=}\pi^t P$. Therefore, under the conditions, the distribution of a Markov chain would be equal to π after being linearly transformed t times. When arriving at moment t and if $t \to \infty$,, $\mu^t = \pi$ holds true, and regardless of the number of moments that passes, the equation $\mu^{t+k} = \pi$ always holds true too.

Now the specific conditions supporting the stationary distribution are explained as following. For which

there exists a unique stationary distribution, the Markov chain is supposed to be **irreducible** and **aperiodic**. Simply and plainly, if a Markov chain is irreducible, an arbitrary state can be transited to any other state in a finite number of transition in a finite state space. Before explain that a Markov chain is aperiodic, we need to know what is an aperiodic state. One state is aperiodic when starting from this state there are irregular times of returning to this state, that is to say, no obvious repeating pattern or periodicity in the number of returns. Based on this, a Markov chain is aperiodic if all the states are aperiodic and this chain satisfies being irreducible. Once these two conditions are satisfied, the Markov chain can be considered to obtain a stationary distribution.

Monte Carlo Approximations Monte Carlo methods are probabilistic approaches that rely on the core idea of using sampling and averaging (random approximation) to estimate values that are computationally complex. One important application of Monte Carlo methods is in the calculation of definite integrals, and the corresponding term is Monte Carlo Integral. For example, assume the target is $\int_a^b f(x)dx$, if we sample x_i from the uniform distribution $U(a,b)$, then it is easy to compute the approximation in $\frac{b-a}{N}\sum_i^N f(x_i)$, where N stands for the number of samples.

As can be seen, the Monte Carlo method is very intuitive, but in practice, the probability distribution of samples, $f(x)$, is often extremely complex, which makes it tough to draw a set of samples that match this probability distribution. To solve the challenge of sampling from a complex probability distribution $f(x)$, two solutions have been derived: Importance Sampling and Accept-Reject Sampling.

In the discrete space, integration can be expressed as summation. According to the previous statement, more generally, if x_i is sampled from $p(x)$, equivalently the expectation of $f(x)$ is expressed as $\mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{N}\sum_i^N f(x_i)p(x_i)$. As for importance sampling, if it is impossible to sample from $p(x)$, we can introduce a distribution $q(x)$ which is accessible to draw a sample easily. Finally, the original expectation is transformed to $\mathbb{E}_{q(x)}[f(x)] = \frac{1}{N}\sum_i^N \frac{p(x_i)}{q(x_i)}f(x_i)$, especially $\frac{p(x)}{q(x)}$ is designated as importance weight.

Switching to Accept-Reject Sampling, the basic idea is to use a proposal distribution $q(x)$, which is easy to sample from, such as a Gaussian distribution, for sampling. The samples obtained from $q(x)$ are processed by accepting the ones that satisfy the condition and rejecting the ones that do not. It is crucial to determine a constant k ensuring $kq(x) \geq p(x)$ always true. In order to ensure that the equation holds true, the constant k should be chosen as small as possible. The process of Accept-Reject Sampling can be described as follows:
1. Use the proposal distribution $q(x)$ to sample a data point x_0.
2. Sample from a uniform distribution U(0,1) and get one sample U .
3. If $U < \frac{p(x_0)}{kq(x_0)}$, accept the sample; otherwise, reject it.

In the previous explanation, it is known that $q(x)$ is an artificial distribution for the target value estimation. When the proposal distribution q(z) does not match the target distribution $p(x)$ properly, the sampling results may exhibit high variance, indicating that the samples are not reliable enough. This can potentially lead to more fluctuation of the model inference results. For instance, if the shape of the distribution $p(x)$ is irregular and it cannot be well captured by sampling from a uniform distribution $q(x)$, it may lead to ignoring important information of $p(x)$ and result in larger variances in the sampling outcomes. Acceptance-rejection sampling mitigates this problem by introducing a step to determine whether a sample is accepted or rejected, where the acceptance rate is defined as $p(x)/kq(x)$ and $kq(x)$ is a distribution that covers $p(x)$. Generally, $q(x)$ is required to as similar as possible to $p(x)$ and it is easy to sample.

Compared with these two sampling methods, Markov Chain Monte Carlo is a much more efficient sampling method.

4.1.2 Markov Chain Monte Carlo

Monte Carlo methods suffer from an overly long sampling process costing lots of time but maybe leaving the results with high variance when it faces to a relatively complex distribution. Although acceptance-rejection sampling improves the sampling efficiency and permits the samples able to reflect the features of target distribution if the sampling process is long enough, it is still limited with the low acceptance rate and slow convergence sometimes.

The limitation lies in the fact that each sampling iteration is independent of previous samples. Inspired by this, we can improve the sampling efficiency by taking advantage of the information from the accepted samples. If a previous sample was accepted, sampling near this accepted point increases the probability of accepting the next sample. This idea of using previously accepted samples to guide future sampling is a key concept in advanced sampling methods. Markov Chain Monte Carlo(MCMC) is proposed based on this idea.

The property of Markov chain-"stationary distribution" can be used here since the sampling process can be considered as a Markov Random Process. Therefore, if we build up a proper "transition matrix" which leads the ultimate stationary distribution to be the target distribution, undoubtedly the sample results are reliable and robust.

For example, Metropolis-Hastings is a typical MCMC. If the target sampling distribution is $p(x)$ and $q(x)$ is the proposal distribution, the sampling acceptance rate is defined as $min(1, \frac{p(j)q(i|j)}{p(i)q(j|i)})$.

4.1.3 Gibbs Sampling

Gibbs sampling is frequently used in sampling high-dimension distribution. The basic idea is to construct a Markov chain by updating each variable based on its conditional distribution given the other states. In the previous chapters, it is proved that for a RBM $P(h \mid v)$ and $P(v \mid h)$ are independent of the partition function, so they can be computed efficiently, which benefits from the RBM energy function construction, so this sort of MCMC sampling method is a suitable technique that can be used to sample the RBM.

Gibbs sampling is a special case of Metropolis-Hastings sampling. One of the most outstanding advantages of Gibbs sampling is that proposals are always accepted, which means in Gibbs sampling the acceptance rate is 1. This property significantly improves the sampling efficiency of sampling. Since Gibbs sampling is the core concept during RBMs training, it is necessary to give the proof of this property.

First, define a multi-variable distribution $Z(z_1, z_2, z_3, ...)$. The stationary distribution is just $p(z)$ and transition matrix is denoted as $Q(Z \to Z^*)$, where Z stands for the current state and Z^* for the state after transiting. Specially, since the conditional probability of one variable given the rest variables is defined as the transition matrix in each sampling step in Gibbs sampling, it is available to conduct $Q(z \to z^*) = p(z_{-i} \to z_i^*) = p(z_i^* \mid z_{-i})$, and $Q(z^* \to z) = p(z_{-i}^* \to z_i) = p(z_i \mid z_{-i}^*)$, where the i^{th} dimension is denoted as z_i, the other dimensions as z_{-i}. It is important to stress that each sampling time, only one dimension is updated while the others are "frozen" to their values, as a result, z_{-i} is the same as z_{-i}^*, which is illustrated in Figure 2.

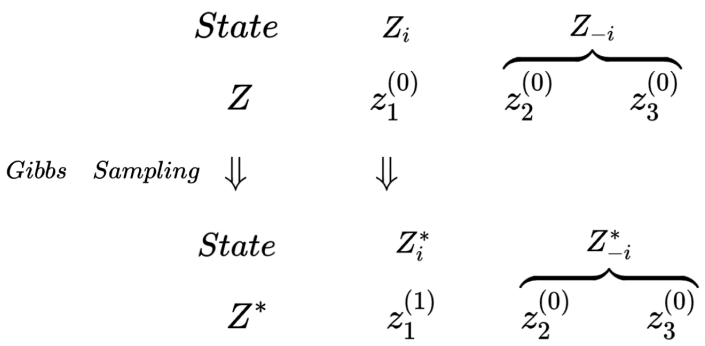


Figure 2 – Gibbs Sampling Schematic

Based on these conductions, it's time to substitute these representations into the acceptance rate equation. In (27), the equivalent parts are marked with the same color, specially z_{-i} is equivalent to z^*_{-i}! Finally, the acceptance rate is proved to be 1.

$$\begin{aligned}\alpha &= min(1, \frac{p(z^*)Q(z^* \to z)}{p(z)Q(z \to z^*)}) \\ &= min(1, \frac{p(z_i^* \mid z_{-i}^*)p(z_{-i}^*)Q(z^* \to z)}{p(z_i \mid z_{-i})p(z_{-i})Q(z \to z^*)}) \\ &= min(1, \frac{p(z_i^* \mid z_{-i}^*)p(z_{-i}^*)p(z_i \mid z_{-i}^*)}{p(z_i \mid z_{-i})p(z_{-i})p(z_i^* \mid z_{-i})}) = 1\end{aligned} \tag{27}$$

So far, the comprehensive introduction and explanation of stochastic sampling in RBMs have come to the end. Starting from Monte Carlo and Markov chain property and following the idea of acceptance rate, MCMC as an more efficient sampling method is introduced to sample underlying the knowledge of anterior sample results. Meanwhile Gibbs sampling, a special case of MCMC, is selected as the basic sampling method since RBMs provide available and light conditional probability inference, which advances the approximation of the intractable model expectation. In the subsequent parts, we would see how Gibbs sampling works in RBM training in different approximation methods.

4.2 Exact Gradient

The derivation of the exact gradient has been explained in $RBM - Learning$, and the model parameters are updated following the gradient ascent by calculating the precise gradient each epoch. All the approximations come from the exact gradient formulas, so it is better to transform the original formulas to a specific form, in mathematical expectation of $P(v)$. Moreover, in machine learning optimization, it is generally conventional to set up the loss function, or objective function, as the negative log likelihood, transferring the maximizing optimization problem to minimizing optimization one.

$$\begin{aligned}\frac{\partial - \log P(v)}{\partial W_{ij}} &= \hat{v}_i P(h_j = 1 \mid \hat{v}) - \mathbb{E}_{P(v)}[P(h_j = 1 \mid v)v_i] \\ \frac{\partial - \log P(v)}{\partial \alpha_i} &= \hat{v}_i - \mathbb{E}_{P(v)}[v_i] \\ \frac{\partial - \log P(v)}{\partial \beta_j} &= P(h_j = 1 \mid \hat{v}) - \mathbb{E}_{P(v)}[P(h_j = 1 \mid v)]\end{aligned}$$

Now the gradients are already prepared, it is time to use these information to update the model parameters via gradient ascent and momentum. The algorithm is listed in Algorithm 1, where batch size is set up as the same as the data size. In the process, it can be seen that the value of partition function is dynamically changing since the model parameters are being modified to fit in the distribution of training data. The other computing burden comes from the gradients of the model expectation, where all of the visible unit situations are taken into account and it results in a exponential level computing cost. These two points are the inevitable barriers existing in the road to the precise gradients, but in practice many advanced mathematical methods are designed to deal with the intractable Bayesian problems, and in this specific case, Gibbs Sample is the medication, which circumvents the straight and precise model expectation computation but infers an approximation of model expectation by using the information of different multi-variables.

Algorithm 1 Exact Gradient Update

Input: data, self.W, self.α, self.β
Output: self.W, self.α, self.β

m-w, m-α, m-β $\leftarrow$ 0,0,0
for epoch in epochs **do**
  Z $\leftarrow$ computeZ(self.W, self.α, self.β)
  for data in DataBatch **do**
    $v_0 \leftarrow data$
    $P(h_0 \mid v_0) \leftarrow sigmoid(v_0 W + \beta)$
    dw-data, dα-data, dβ-data $\leftarrow$ expectation-data(v_0, $P(h_0 \mid v_0)$)
    dw-model, dα-model, dβ-model $\leftarrow$ expectation-model(Z)

    dw = dw-data - dw-model
    dα = dα-data - dα-model
    dβ = dβ-data - dβ-model
  end for
  m-w = momentum * m-w + (1-momentum) * dw
  m-α = momentum * m-α + (1-momentum) * dα
  m-β = momentum * m-β + (1-momentum) * dβ

  self.W += learning-rate * m-w
  self.α += learning-rate * m-α
  self.β += learning-rate * m-β
end for
return self.W, self.α, self.β

4.3 Contrastive Divergence

In RBMs learning part, the gradient of the log likelihood $P(v)$ with model parameters is the difference between the empirical expectation and the model expectation. The first term could be calculated from the training data, but the second term seems intractable to compute precisely since partition function is involved. According to the derivation, the gradients with respect to the parameter, W,α and β, are actually homogeneous because all of the gradients can be represented as the expectation of corresponding functions under $p(v)$. Thus Gibbs sampling is selected to sample reliable v from $P(v, h)$, and once the chain reaches the stationary distribution then all the sampled v could be considered sampled from $P(v)$.

Based on these works, Contrastive Divergence(CD) provides a way to approximate gradient via Gibbs

sampling, and this sort of method is considered as the standard RBM training. The k-step Contrastive Divergence(CDk) is to initialize the Markov Chain with v_0 then run Gibbs sampling for k times and get v_k, substitute the model expectation term with v_k estimations, and compute the gradient used for updating model parameters. The mathematical expressions of gradients are demonstrated as following.

$$\bigtriangledown_w - \log P(v) = P(h = 1 \mid v_0)v_0 - P(h = 1 \mid v_k)v_k \tag{28}$$

$$\bigtriangledown_\alpha - \log P(v) = v_0 - v_k \tag{29}$$

$$\bigtriangledown_\beta - \log P(v) = P(h = 1 \mid v_0) - P(h = 1 \mid v_k) \tag{30}$$

In practice, after one time Gibbs sampling the sample v is sufficient to reconstruct the target distribution roughly, but under the effect of burn-in period CD1 usually suffers from the biased approximation, since there is a risk that some samples are not consistent with the stationary distribution. One way to diminish the bias is to use a larger k, because in theory when $k \to \infty$ the sampled distribution is equivalent to the target distribution $p(v)$. It is worthy noting that larger k would bring more accurate approximation but at the same time increase computational burden, so this is a trade-off problem and the specific solution varies among different cases. The CDk algorithm is listed in Algorithm2, where batch size is set up as the same as the data size.

Algorithm 2 Contrastive Divergence - CDk

Input: data, self.W, self.α, self.β
Output: self.W, self.α, self.β

m-w, m-α, m-$\beta \leftarrow$ 0,0,0
$N_B \leftarrow batchnumber$
for epoch in epochs **do**
  for data in DataBatch **do**
    $v_0 \leftarrow data$
    for i in k **do**
      $P(h_i \mid v_i) \leftarrow sigmoid(v_i W + \beta)$
      $h_i \sim P(h_i \mid v_i)$
      $P(v_{i+1} \mid h_i) \leftarrow sigmoid(h_i W^T + \alpha)$
      $v_{i+1} \sim P(v_{i+1} \mid h_i)$
    end for

    $dw = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid v_0^i)v_0^i - P(h = 1 \mid v_k^i)v_k^i)$
    $\mathrm{d}\alpha = \frac{1}{N_B} \sum_i^{N_B} (v_0^i - v_k^i)$
    $\mathrm{d}\beta = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid v_0^i) - P(h = 1 \mid v_k^i))$
  end for
  m-w = momentum * m-w + (1-momentum) * dw
  m-α = momentum * m-α + (1-momentum) * dα
  m-β = momentum * m-β + (1-momentum) * dβ

  self.W += learning-rate * m-w
  self.α += learning-rate * m-α
  self.β += learning-rate * m-β
end for
return self.W, self.α, self.β

4.4 Persistent Contrastive Divergence

Persistent Contrastive Divergence is an extension of Contrastive Divergence for training RBMs. Remind of CDk training process, toward one data batch one Markov chain is established to sample v_k and then abandoned, that is to say during each sampling time the Markov chain starts from the scratch. In PCD the one-time-used Markov chain is cancelled and replaced with a persistent chain. The idea of PCD is to introduce a Markov chain that goes through all the sampling process, and the persistent chain sampled results are used as v_k during each update. With the help of the persistent chain the model could reach more optimal regions.

Generally, PCD outperforms CD since the persistent chain ensures a deep exploration through the target probability distribution and takes advantage of the previous sample information instead of starting at a random state so that prevents the model from being stuck in the local optimum. Most importantly, after long exploration along the chain, the final approximation would turn to be less biased compared to CD. Now PCD has been the primary option rather than CD in most cases.

The PCD-k algorithm is listed in Algorithm3, where batch size is set up as the same as the data size.

Algorithm 3 Persistent Contrastive Divergence - PCD

Input: data, self.W, self.α, self.β
Output: self.W, self.α, self.β

persistent chain = $None$
m-w, m-α, m-β $\leftarrow$ 0,0,0
$N_B \leftarrow batchnumber$
for epoch in epochs **do**
  for data in DataBatch **do**
    if persistent chain = None **then**
      $v_0 \leftarrow data$
    else
      v_0 = **persistent chain**
    end if
    for i in k **do**
      $P(h_i \mid v_i) \leftarrow sigmoid(v_i W + \beta)$
      $h_i \sim P(h_i \mid v_i)$
      $P(v_{i+1} \mid h_i) \leftarrow sigmoid(h_i W^T + \alpha)$
      $v_{i+1} \sim P(v_{i+1} \mid h_i)$
    end for

    $dw = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid data) data - P(h = 1 \mid v_k^i) v_k^i)$
    dα = $\frac{1}{N_B} \sum_i^{N_B} (data - v_k^i)$
    dβ = $\frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid data) - P(h = 1 \mid v_k^i))$

  end for
  m-w = momentum * m-w + (1-momentum) * dw
  m-α = momentum * m-α + (1-momentum) * dα
  m-β = momentum * m-β + (1-momentum) * dβ

  self.W += learning-rate * m-w
  self.α += learning-rate * m-α
  self.β += learning-rate * m-β
  persistent chain = v_k
end for
return self.W, self.α, self.β

4.5 Weighted Contrastive Divergence

Recall the gradient of $\log P(v)$ with θ which comprises two terms of expectation, empirical expectation and model expectation. As the empirical expectation can be computed exactly from the training data, it is only necessary to approximate model expectation to analyze the difference between the exact gradient and the estimated gradient.

In CD algorithm, MCMC is used to sample v_k to estimate the gradient and the corresponding mathematical expression goes as $\sum_{i=1}^{N_B} \frac{1}{N_B} \mathbb{E}_{P(h|v_k^i)}[\frac{\partial E(v_k^i, h)}{\partial \theta}]$, where N_B stands for the volume of one data batch, θ for model parameters and v_k^i for the i^{th} visible unit state after k-step gibbs sampling. In the exact gradient derivation, the gradient is $\sum_v P(v) \mathbb{E}_{P(h|v)}[\frac{\partial E(v,h)}{\partial \theta}]$, where v stands for all of the cases in visible variable space. Comparing these two gradient terms, two main different points can be discovered:

1. CD algorithm only takes the training-data-containing cases of v into account which scales down the exploring space.
2. In exact gradient the expectation, log likelihood of energy under conditional probability $P(h \mid v)$, is assigned with a dynamic term $P(v)$, inversely in CDk algorithm all the expectation is assigned with the same constant $\frac{1}{N_B}$.

Based on these two points, we can understand the gradient approximation in a new perspective. In the available exploring space of v, the approximation is composed of the expectation of loglikelihood gradient under $P(h \mid v)$ and the corresponding $P(v)$, where $P(v)$ could be considered as the assigned weight. According to the second point above, in theory, modifying $\frac{1}{N_B}$ to a reasonable weight $\overline{P}(v)$ can advance the approximation. The model expectation goes as (31).

$$\mathbb{E}_{model}[\frac{\partial E(v,h)}{\partial \theta}] = \sum_{i=1}^{N_B} \overline{P}(v_k^i)\mathbb{E}_{P(h|v_k^i)}[\frac{\partial E(v_k^i,h)}{\partial \theta}] \tag{31}$$

In WCD algorithm, due to the case where we can only use the achieved data to approach to the exact $P(v)$, so $\overline{P}(v_k^i)$ is defined as (32). Notice that although $P(v)$ is utilized in the definition of the weight, the partition function is eliminated in the end, which makes the computation efficient and easily accessible.

$$\overline{P}(v_i) = \frac{P(v_i)}{\sum_j^{N_B} P(v_j)} = \frac{\sum_h exp\{-E(v_i,h)\}}{\sum_j^{N_B}\sum_h \{-E(v_j,h)\}} \tag{32}$$

The WCD algorithm is demonstrated in Algorithm 4, where batch size is set up as the same as the data size.

Algorithm 4 Weighted Contrastive Divergence - WCD

Input: data, self.W, self.α, self.β
Output: self.W, self.α, self.β

m-w, m-α, m-$\beta \leftarrow$ 0,0,0
for epoch in epochs **do**
  for data in DataBatch **do**
    $v_0 \leftarrow data$
    for i in k **do**
      $P(h_i \mid v_i) \leftarrow sigmoid(v_i W + \beta)$
      $h_i \sim P(h_i \mid v_i)$
      $P(v_{i+1} \mid h_i) \leftarrow sigmoid(h_i W^T + \alpha)$
      $v_{i+1} \sim P(v_{i+1} \mid h_i)$
    end for

    $weights = ComputeWeight(v_k, self.W, self.\alpha, self.\beta)$

    $dw = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid v_0^i)v_0^i - weights^{(i)} P(h = 1 \mid v_k^i)v_k^i)$
    $\mathrm{d}\alpha = \frac{1}{N_B} \sum_i^{N_B} (v_0^i - weights^{(i)} v_k^i)$
    $\mathrm{d}\beta = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid v_0^i) - weights^{(i)} P(h = 1 \mid v_k^i))$
  end for

  m-w = momentum * m-w + (1-momentum) * dw
  m-α = momentum * m-α + (1-momentum) * dα
  m-β = momentum * m-β + (1-momentum) * dβ

  self.W += learning-rate * m-w
  self.α += learning-rate * m-α
  self.β += learning-rate * m-β
end for
return self.W, self.α, self.β

Weighted Persistent Contrastive Divergence This sort of algorithm combines the idea of persistent chain and the weight assignment together. The aim is to build a long Markov chain to explore possible space to find better optimum and also assign weights to the corresponding state probability for a better convergence. Therefore, the Markov chain is initialized at the beginning of learning, and update the final state at the end of each batch iteration.

The algorithm is listed in Algorithm 5, where batch size is set up as the same as the data size.

Algorithm 5 Weighted Persistent Contrastive Divergence - WPCD

Input: data, self.W, self.α, self.β
Output: self.W, self.α, self.β

persistent chain = $None$
m-w, m-α, m-β $\leftarrow$ 0,0,0
for epoch in epochs **do**
  for data in DataBatch **do**
    if persistent chain = None **then**
      $v_0 \leftarrow data$
    else
      v_0 = **persistent chain**
    end if
    for i in k **do**
      $P(h_i \mid v_i) \leftarrow sigmoid(v_i W + \beta)$
      $h_i \sim P(h_i \mid v_i)$
      $P(v_{i+1} \mid h_i) \leftarrow sigmoid(h_i W^T + \alpha)$
      $v_{i+1} \sim P(v_{i+1} \mid h_i)$
    end for

    $weights = ComputeWeight(v_k, self.W, self.\alpha, self.\beta)$

    $dw = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid data)data - weights^{(i)} P(h = 1 \mid v_k^i) v_k^i)$
    dα = $\frac{1}{N_B} \sum_i^{N_B} (data - weights^{(i)} v_k^i)$
    dβ = $\frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid data) - weights^{(i)} P(h = 1 \mid v_k^i))$
  end for

  m-w = momentum * m-w + (1-momentum) * dw
  m-α = momentum * m-α + (1-momentum) * dα
  m-β = momentum * m-β + (1-momentum) * dβ

  self.W += learning-rate * m-w
  self.α += learning-rate * m-α
  self.β += learning-rate * m-β
  persistent chain = v_k
end for
return self.W, self.α, self.β

4.6 Parallel Tempering

Parallel Tempering is one of the most powerful sampling techniques, and can be integrated with RBM bringing better performance and enhancement for the sampling efficiency. In the PT algorithm, the concept of 'temperature' is introduced. Based on a sequence of linearly increasing temperature chains, usually from 0 to 1, we establish a sequence of corresponding homogeneous distribution chains, where the start is the uniformed distribution (Temperature=0) and the end is the target distribution (Temperature=1). Hence, the original RBM formula is modified to (33) in PT, where r stands for the index in the temperature sequences, and Tr for the temperature.

$$P_r(v,h) = \frac{1}{Z_r} e^{-\frac{1}{Tr}E(v,h)} \tag{33}$$

The Parallel Tempering has two main advantages: 1.improve sampling and model performance. 2.Faster convergence of model training. As the expression above, the PT chain actually constructs many gradually varying spaces which ensures the model to explore more of the energy landscape as well as avoiding local optimum, because the information from both higher and lower temperature levels is mixed up in the end so to help the model escape poor local minima.

After sampling the RBMs in parallel by Gibbs sampling, we need to compute the probability in exchanging the states of two neighbor Markov chains, with temperature T_r and T_{r-1}. The standard of the probability follows the Metropolis ratio as (34).

$$min\left\{1, \frac{P_r(v_{r-1}, h_{r-1})P_{r-1}(v_r, h_r)}{P_r(v_r, h_r)P_{r-1}(v_{r-1}, h_{r-1})}\right\} \tag{34}$$

With the specific definition of energy function in RBM, the Metropolis ratio is converted to (35).

$$min\left\{1, \exp\left\{(\frac{1}{T_r} - \frac{1}{T_{r-1}})(E(v_r, h_r) - E(v_{r-1}, h_{r-1}))\right\}\right\} \tag{35}$$

As for the regulation of swapping, there are lots of choices to output the ultimate sampling state from the chain sequence. In order to mix the different temperature chains much better, odd-even-order mixing appears to give out the outstanding performance. During each swapping step, first discriminate if exchange the odd-index chains and corresponding next chains, and then move to the even-index chains and check whether or not swap. The swapping time is defined as (36)for efficiency and reliability.

$$SwapTime = max\left\{2, \sqrt{\#TemperatureChains}\right\} \tag{36}$$

Based on all these explanation, the PT algorithm is demonstrated in Algorithm 6, where batch size is set up as the same as the data size.

Algorithm 6 Parallel Tempering- PT

Input: data, self.W, self.α, self.β

Output: self.W, self.α, self.β

$self.ChainNum, self.beta \leftarrow ChianNum, linspace(0.0, 1.0, self.chain_num)$
$self.SwapTime = max(2, \sqrt{self.Chain_Num})$
m-w, m-α, m-$\beta \leftarrow$ 0,0,0
$N_B \leftarrow batchnumber$
for epoch in epochs **do**
  for data in DataBatch **do**
    $v_0 \leftarrow data$
    $v.chain = v.chain * beta$
    for i in k **do**
      $P(h_i \mid v.chain_i) \leftarrow sigmoid(v.chain_i W + \beta)$
      $h_i \sim P(h_i \mid v.chain_i)$
      $P(v.chain_{i+1} \mid h_i) \leftarrow sigmoid(h_i W^T + \alpha)$
      $v.chain_{i+1} \sim P(v.chain_{i+1} \mid h_i)$
    end for

    for j in $self.SwapTime$ **do**
      v.chain = swap-odd-index(v.chain)
      v.chain = swap-even-index(v.chain)
    end for
    $v_k = state - sample(v.chain[:, -1, :])$
    $P(h = 1 \mid v_k) = sigmoid(v_k)$

    $dw = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid v_0^i) v_0^i - P(h = 1 \mid v_k^i) v_k^i)$
    d$\alpha = \frac{1}{N_B} \sum_i^{N_B} (v_0^i - v_k^i)$
    d$\beta = \frac{1}{N_B} \sum_i^{N_B} (P(h = 1 \mid v_0^i) - P(h = 1 \mid v_k^i))$
  end for
  m-w = momentum * m-w + (1-momentum) * dw
  m-α = momentum * m-α + (1-momentum) * dα
  m-β = momentum * m-β + (1-momentum) * dβ

  self.W += learning-rate * m-w
  self.α += learning-rate * m-α
  self.β += learning-rate * m-β
end for
return self.W, self.α, self.β

5 Experimental Analysis

5.1 Datasets

In total, there are 4 datasets used for the experiment. All the datasets are of binary variables 0 & 1 and already flatten into one-dimension array, so there is no need to transform the datasets. Among the datasets, the dataset 'Bars-and-Stripes-3x3' is in a relatively small size hence it is considered as the development dataset for selecting the proper hyperparameter set, rather than dataset "Bars-and-Stripes-4x4", "Labeled-Shifter-4-11" and "Labeled-Shifter-5-13", because the computing cost of RBMs relies on the input dimension to a large extent. Information of these datasets are displayed in Table 1.

Table 1 – Dataset Information

	Size	Dimension	Type
Bars-and-Stripes-3x3	14	9	[0,1]
Bars-and-Stripes-4x4	30	16	[0,1]
Labeled-Shifter-4-11	48	11	[0,1]
Labeled-Shifter-5-13	96	13	[0,1]

5.2 Evaluation Metrics

5.2.1 Kullback–Leibler Divergence and Negative Log-Likelihood

KL Divergence is utilized to quantitatively describe the difference of two distribution. The mathematical definition of KL divergence goes as (37) for discrete variables. It is worth noting that KL divergence is always greater or equal to 0. Specifically, KL divergence is not symmetric which means $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. The lower KL divergence signifies the more similar two distribution are, so in common the target is to minimize the KL divergence between the training data distribution and the target data distribution.

$$D_{KL}(P||Q) = \sum_x P(x) \log(\frac{P(x)}{Q(x)}) \tag{37}$$

Negative Log-Likelihood(NLL) is one of the most frequent metrics used in the machine learning. In model training, the core idea is often to increase the $P(x)$ on training data. In order to diminish the computation cost and fit the optimization schedule, the $P(x)$ optimization is transformed to $-\log P(x)$.

Technically, optimizing KL divergence is equivalent to optimizing NNL. Recall that $NLL = -\frac{1}{N}\sum_n logQ(x_n)$ and $D_{KL}(P||Q) = \sum_x P(x)\log(\frac{P(x)}{Q(x)})$, if we assume $P(x)$ as $1/N$, then the KL divergence is $\sum_n \frac{1}{N}\log(\frac{1}{N}\frac{1}{Q(x_n)})$, after transformation we get $-\frac{1}{N}\sum_n \log N - \frac{1}{N}\sum_n \log Q(X_n)$, which is actually $-\log N + NLL$. In summary, under the condition that each training data sample has the same probability, we can consider that $KL = NLL - \log N$. In this experiment, KL divergence and NLL are equivalent in CD-series algorithms, but the equal distribution hypothesis is not true in WCD-series algorithms and exact gradients, so it is still necessary to compute both of the metrics to supervise the model training process.

5.2.2 Probability Summation of Training Data

The idea of energy based models is to assign lower energy to the desirable data space leading to more stable states, which equally signifies well-trained model would gather the majority of probability on the training data. This is the reason that the probability summation is utilized to evaluate the training process.

5.2.3 Entropy Percentage

Entropy is used to describe how uniform the data distribution is, and the corresponding mathematical expression goes as (38). The exact gradient algorithm is supposed to hold the highest entropy values because there is no gradient bias and the assigned probability is equal among all the data samples, where the entropy upper threshold is $\log \frac{1}{N_B}$. Oppositely, CD-series algorithms are doing the approximation of gradients, so there must be several data points assigned with probability exceeding the threshold, which result in a lower entropy. Farther away from this threshold means there is potentially more bias existing the gradient approximation.

$$Entropy(P(x)) = -\sum_{n} P(x_n) \log P(x_n) \tag{38}$$

In this experiment the upper threshold varies among the datasets because the number of training data is different, so we decide to propose a normalized metric of entropy. This metric is designated as **Entropy Percentage**, and defined as the ratio of the entropy of the approximating distribution sum and the entropy upper threshold of the current dataset(also can be considered as the distribution via learning in exact gradient) (39). Based on that, Entropy Percentage ranges from 0 to 1, where 1 means the uniformed distribution and 0 means extremely biased distribution. In conclusion, this sort of metric is used to evaluate the bias effect on the approximating algorithms.

$$EntropyPercentage = \frac{\sum_T P(x) \log P(x)}{\log \frac{1}{N}} \tag{39}$$

T stands for the training data points and N is the number of examples in the training dataset..

5.3 Hyperparameter Setup

Before conducting the complete analysis on all the algorithms, there are several important hyperparameters to determine. As expressing before, the dataset "Bars-and-Stripes-3x3" would be used as the developing dataset to explore the proper parameter sets, and the basic searching method in this pre-experiment is Grid-search, besides the basic approximation method is Contrastive Divergence(k=1 or k=10) and Weighted Contrastive Divergence. Based on the to-achieve results, the further fine-tuning of the parameters would be done for better convergence.

Variable Type In the beginning of the experiment, the input variable type is studied. As the matter of fact, RBM is designed to deal with the binary input variables so there could be two cases $[0, 1]$ and $[-1, 1]$. When the input variables belong to $[-1, 1]$, with the same energy function configuration, the

inferences are modified as following.

$$P(h \mid v) = \frac{1}{1 + exp\{2 * (-vW - \beta)\}}$$
$$P(v \mid h) = \frac{1}{1 + exp\{2 * (-hW^T - \alpha)\}}$$
$$P(v) = \frac{1}{Z} \exp\left\{\alpha v^T + \log(exp(-vw - \beta) + exp(vw + \beta))\right\}$$

When training RBM with $[-1, 1]$ input, the dataset is scaled from $[0, 1]$ to $[-1, 1]$. We need to figure out which type of the input variable is more reliable and robust. To test these two sorts of RBMs, we want to start from initializing the model weights in Gaussian distribution with different variation. CD1 and WCD1 are selected to update the models with learning rates as 0.01 and 0.1 respectively, because we can manage to train the RBM without training tricks. The training epoch is 100,000 and the corresponding KL values are used to compare the RBMs performance. α and β are initialized as 0, and W is configured to be sampled from a Gaussian distribution with the fixed mean as 0 and multi-level standard errors. In total, 4 levels of standard errors, $1, 0.1, 0.01, 0.001$, are used to evaluate the RBMs. As for the hidden layer dimension, as an important hyperparameter, it will be discussed later, here according to the experiments[9], the hidden layer dimension is set up as 5 times of the visible layer dimension, $5N_v$. In order to explore more cases, we add one more group with $3N_v$. Train two kinds of RBMs for **10** times and the KL averages and standard errors are demonstrated in Table 2 and Table 3.

Table 2 – KL comparison with different σ when the visible layer dimension is $N_h = 3N_v$

Group	$\sigma = 1$	$\sigma = 0.1$	$\sigma = 0.01$	$\sigma = 0.001$
$[0, 1]\&CD1$	0.591(0.052)	0.511(0.036)	0.479(0.029)	0.505(0.052)
$[0, 1]\&WCD1$	0.1416(0.052)	0.130(0.036)	0.137(0.029)	0.154(0.052)
$[-1, 1]\&CD1$	1.709(0.366)	0.101(0.638)	0.066(0.032)	0.083(0.041)
$[-1, 1]\&WCD1$	0.062(0.013)	0.076(0.020)	0.066(0.031)	0.059(0.025)

Table 3 – KL comparison with different σ when the $N_h = 5N_v$

Group	$\sigma = 1$	$\sigma = 0.1$	$\sigma = 0.01$	$\sigma = 0.001$
$[0, 1]\&CD1$	0.658(0.065)	0.548(0.060)	0.548(0.051)	0.551(0.033)
$[0, 1]\&WCD1$	0.153(0.005)	0.162(0.036)	0.160(0.029)	0.150(0.052)
$[-1, 1]\&CD1$	4.702(2.252)	0.087(0.032)	0.091(0.022)	0.071(0.026)
$[-1, 1]\&WCD1$	0.001(0.001)	0.089(0.097)	0.14(0.032)	0.10(0.041)

Observing the table results, apparently RBMs with input $[-1, 1]$ are more sensitive with the initial weight variation. It is proven that the increasing of the weight norm would lead to the decreasing of KL[5]. From this point, it can be seen that RBMs with input $[0, 1]$ are more stable during the training process, and this sort of RBM can deal with a wider range of weights so that it has better capability of avoiding the poor performance. RBMs with input $[-1, 1]$ could give out good performance, especially when σ is limited to 0.01, the KL of CD1 reaches a lower level, but when switching to WCD1, the best KL of [-1,1] input appearing in the group with $\sigma = 1$. Oppositely, the RBMs trained with CD1 when σ=1 give out a relatively large KL value. In contrary, the results of RBMs with input [0,1] remain around 0.5 in all groups, indicating this kind of RBM is more robust. Since RBM with input [-1,1] is affected by the weight initialization significantly and the other kind of RBM performs more stably with different σ, finally [0,1] are chosen as the input variable type for RBM training.

Dimension of hidden layer To determine the dimension number of the hidden layer, we need to create a search space where the dimension $\in [N_v, 2N_v, 3N_v, 4N_v, 5N_v]$ (N_v is the dimension of visible layer) and the learning rate $\in [1e-2, 1e-3, 1e-4]$ according to the anterior work. No learning rate decay or weight decay is involved in the exploring process. The exploration is conducted in CD1 and CD10 algorithms, and the corresponding KL, NLL, Prob Summation would be computed for comparison. Specifically, apply the algorithm for 5 times on each parameter combination, and the average of the metric is recorded.
Based on the achieved results, when dimension of hidden layer is $3N_v$ the KL and Prob Summation reach good values compared to the other dimensions, also $3N_v$ does not bring more computing burden as $5N_v$. The part of results of $3N_v$ are demonstrated in Table 4. The complete record can be seen in the **Appendix**. Therefore, the dimension of hidden layer is chosen to be $3N_v$. Moreover, $1e-3$ seems like a proper learning rate for CDk approximation because of the less oscillation, but we need to determine the exact learning rate according to the situation. The experiment will be repeated **10** times for each set of parameter combinations with random initialization belonging to $N(0, 1)$.
In the following computation, the hidden dimension is fixed as $3N_v$.

Table 4 – Metrics of $3N_v$

Opt	lr	N_h	KL	ProbSum
cd1	0.01	27	0.434(0.048)	0.9206(0.057)
cd1	0.001	27	0.477(0.036)	0.9271(0.018)
cd1	0.0001	27	0.954(0.014)	0.4883(0.017)
cd10	0.01	27	0.0322(0.0028)	0.9917(0.0036)
cd10	0.001	27	0.0462(0.0028)	0.9818(0.0006)
cd10	0.0001	27	0.461(0.0154)	0.6283(0.0118)

Learning Rate Decay and Weight Decay Learning rate decay and weight decay are only set up as non-zero values when conducting PCD, WPCD and PT algorithms in order to reach a stable and better convergence. From the experience of the previous work, the value of weight decay for PCD is better to be set as $2.5e-5$ [11], and this experimental setup is utilized in the experiment. As for the learning rate decay, there are two options: exponential decay and linear decay. The part of results of the exponential decay are demonstrated in Table 5, and the complete records can be seen in **Appendix**. The experiment will be repeated **10** times for each set of parameter combinations with random initialization belonging to $N(0, 1)$, and PCD is selected to be tested.

Table 5 – Metrics of exponential decay

Opt	lr	LrDecay	WeightDecay	KL	ProbSum
PCD	0.01	1.00E-09	2.50E-05	0.00829(0.0005)	0.9405(0.0027)
PCD	0.01	1.00E-10	2.50E-05	0.2814(0.0042)	0.8752(0.1264)
PCD	0.01	1.00E-11	2.50E-05	0.5278(0.0136)	0.8782(0.0311)
PCD	0.001	1.00E-09	2.50E-05	1.7668(0.0335)	0.1850(0.0213)
PCD	0.0001	1.00E-09	2.50E-05	3.5602(0.119)	0.0398(0.0085)

It is easy to discover that the PCD-trained performance is sensitive with the initial parameters. We found out that the KL divergence declines drastically when the learning rate decays from 0.001 to around 0.0003. Following this crucial clue, we decide to implement linear decay where the lower bound is set up as $1e-6$ to leave more space and time to explore the tiny space and also to eliminate a hyperparameter option. The same learning rate decay schedule is applied with WPCD algorithm too.

5.4 Results for all the Datasets

Following the hyperparameter setup in the previous part, the different optimization methods are conducted on the 4 datasets and the experiment results are demonstrated. In the experiment, the learning rate would be modified to adapt to the approximation methods. The training epoch is configured as $1e^5$ which is enough for the convergence for most of the cases. Also as mentioned before, the learning rate decay is utilized only persistent-associated optimization methods, and the type is linear decay from the initial value to $1e^{-6}$ since learning rate around $1e^{-5}$ is a threshold of reaching lower KL convergence values. As for the PT algorithm experiment, the related parameters are set to the lowest level, where the number of temperature-decaying chains is set up as 2 and the learning algorithm is CD. Each experiment is repeated for **30** times and the corresponding metrics are collected up to compute the average and standard error so that we could compare the different algorithm performance.

5.4.1 Bars-and-Stripes-3x3

After hyperparameter setup and the fine-tuning of learning rates, the parameter configuration of different algorithms is listed in Table 6.

Table 6 – Parameter configuration of BS3x3

Opt	sample-type	lr	lr-decay	gibbs-sampling-num	chain-num
CDK	gibbs-sampling	0.005	False	1	None
PCD	gibbs-sampling	0.01	True	1	None
WCD	gibbs-sampling	0.1	False	1	None
$WPCD$	gibbs-sampling	0.03	True	1	None
PT	parallel-tempering	0.005	True	1	2

After a 30-time repetition, the means and standard errors of target metrics are demonstrated in Table 7. Moreover, the evolution of 5 different KL in the first 100,000 epochs and the KL convergence are demonstrated in Figure 3. (* means the corresponding KL is not plotted in KL evolution figure)

Table 7 – Experiment results of BS3x3

	KL	NLL	Prob-sum	Entropy-Percentage
$CD1$	0.45528 (0.0476)	-3.09946 (0.05310)	0.9238 (0.00189)	0.8647 (0.01495)
PCD	0.0809 (0.02586)	-2.71694 (0.01599)	0.96548 (0.00284)	0.94561 (0.00886)
WCD	0.0541(0.0029)	-2.6931(0.0029)	0.9485(0.0029)	0.9671(0.0018)
$WPCD$	0.2315 (0.00678)	-3.06958 (0.00557)	0.8098 (0.00488)	0.8785 (0.0036)
PT	0.0875(0.088)	-2.7666(0.048)	0.9782(0.002)	0.9636(0.0229)
$CD10^*$	0.0501(0.0051)	-2.6823(0.0051)	0.9732(0.0026)	0.9743(0.0024)
$WCD10^*$	0.0595(0.0072)	-2.6996(0.0072)	0.9497(0.0053)	0.9669(0.004)

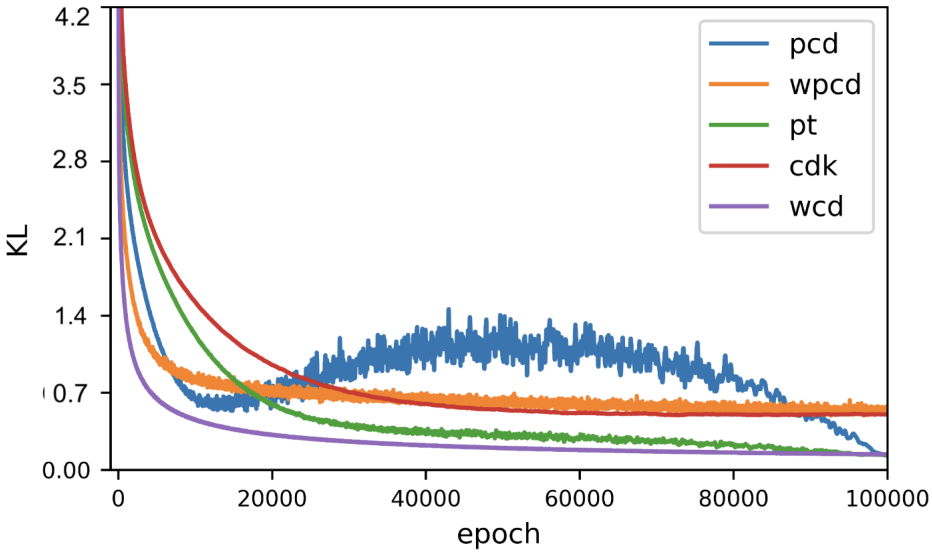


Figure 3 – The KL evolution of 5 approximation algorithms on BS3x3

All the algorithms are configured with gibbs sampling time as 1. According to the results in Table7, CD1 as the standard approximation method has the highest KL, NLL as well as Entropy-percentage, it indicates the modifications on CD are positive to improve the learning. Specially, WCD1 has the fastest and best convergence compared to the other algorithms, but it seems the combination of the persistent chain and WCD does not enhance the results further. The final metric values of PCD are really closed to those of WCD, but the curve of the KL evolution is in an arch shape which goes across most of the training epochs. As mentioned before, this is because PCD is sensitive with the learning rate, so it presents the unstable results. Switching to PT, even though it is set as the lowest standard, there is an obvious performance advance compared to CD1, and PT makes the convergence faster. When increasing the Gibbs sampling time to 10, CD10 could give out the roughly same results as those of WCD1. As for WCD10, it seems bring a little bit fluctuation to the results, and there is no significant improvement.

The probability distributions of the training data set are demonstrated as following figures. Specially, we do extra tests on CD10(the same configuration with CD1) and WCD10(the same configuration with WCD1) to see if adding sampling times alleviates the bias effect, and the distribution figures via CD10 and WCD10 are displayed.

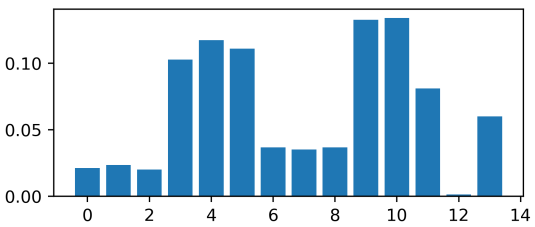


Figure 4 – BS3x3 data distribution via CD1

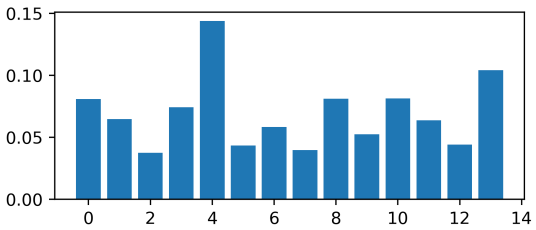


Figure 5 – BS3x3 data distribution via PCD

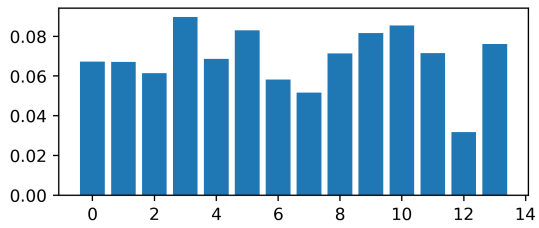


Figure 6 – BS3x3 data distribution via CD10

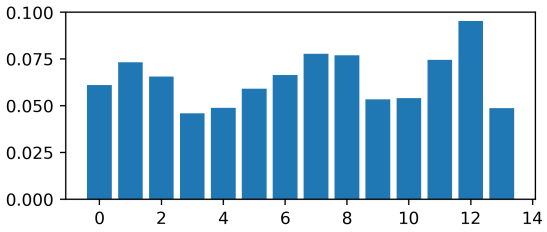


Figure 7 – BS3x3 data distribution via PT

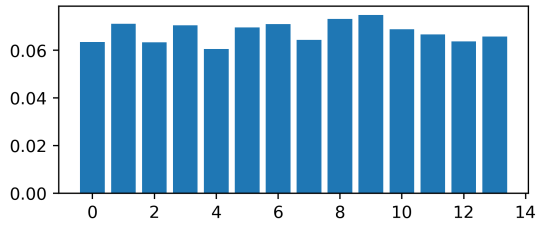


Figure 8 – BS3x3 data distribution via WCD1

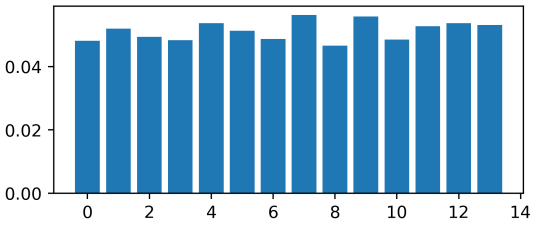


Figure 9 – BS3x3 data distribution via WPCD

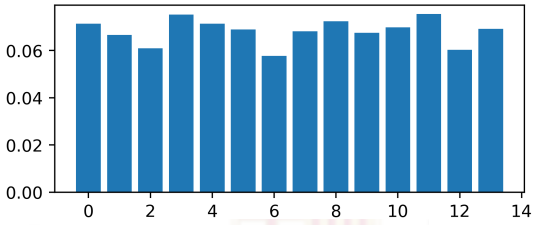


Figure 10 – BS3x3 data distribution via WCD10

The probability distribution of the training data points describe how biased the well-trained results are. There is much bias in the results of CD, PCD and PT, even though the increasing of gibbs sampling time can mitigate this problem, some data points are still designed with small probability. The unbalanced probability distribution would affect the RBM to a situation where the generated results centralize in several training samples. Inversely, WCD and WPCD give out the relatively balanced distribution, which is a significant improvement. Specially, the results of WCD1 and CD10 are really similar, and this is a really ideal standard line for comparison.

5.4.2 Bars-and-Stripes-4x4

After hyperparameter setup and fine-tuning of learning rates, the parameter configuration of different algorithms is listed in Table 8.

Table 8 – Parameter configuration of BS4x4

Opt	sample-type	lr	lr-decay	gibbs-num	chain-num
CDK	gibbs-sampling	0.005	False	1	None
PCD	gibbs-sampling	0.01	True	1	None
WCD	gibbs-sampling	0.1	False	1	None
$WPCD$	gibbs-sampling	0.03	True	1	None
PT	parallel-tempering	0.005	True	1	2

After a 30-time repetition, the means and standard errors of target metrics are demonstrated in Table 9. Also, the evolution of 5 different KL of the approximation algorithm are demonstrated in Figure 11. (* means the corresponding KL is not plotted in KL evolution figure)

Table 9 – Experiment results of BS4x4

	KL	NLL	Prob-sum	Entropy-Percentage
CDK	1.5793(0.0809),	-4.9805(0.0809),	0.908(0.0072),	0.7865(0.0107)
PCD	0.6916(0.2118),	-4.0928(0.2118),	0.9718(0.0071),	0.8023(0.0555)
WCD	0.1018(0.0024),	-3.503(0.0024),	0.9088(0.0029),	0.9327(0.0017)
$WPCD$	0.3727(0.012),	-3.7739(0.012),	0.6907(0.0075),	0.7653(0.0063)
PT	0.2782(0.0691),	-3.6794(0.0691),	0.9072(0.0121),	0.8805(0.0188)
$CD10^*$	0.2743(0.0116)	-3.6755(0.0116)	0.9412(0.0032)	0.913(0.0026)
$WCD10^*$	0.102(0.0149)	-3.5032(0.0149)	0.9101(0.0107)	0.9332(0.0087)

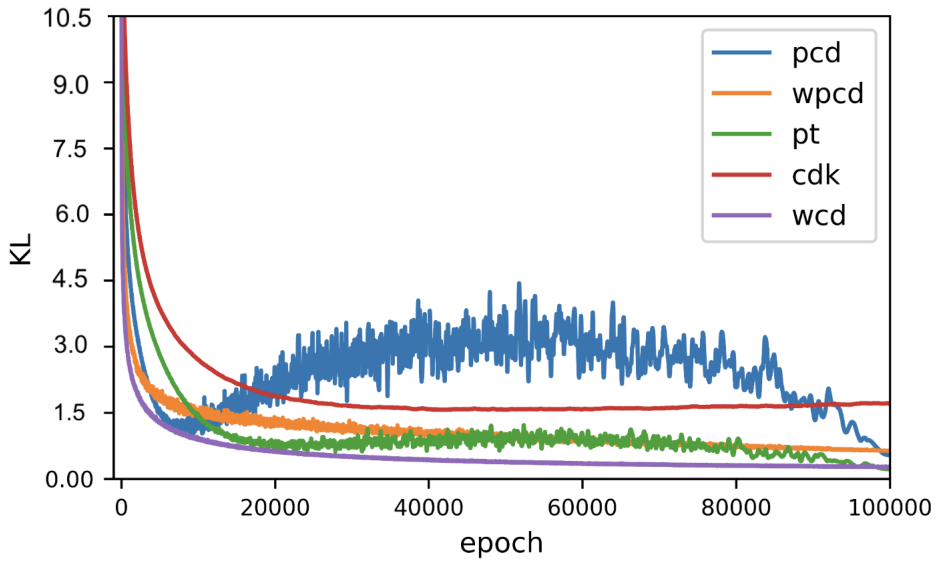


Figure 11 – The KL evolution of 5 approximation algorithms on BS4x4

The size of data is large than BS3x3, and the corresponding results are slightly influenced. The same as the results in BS3x3, WCD achieves the best convergence among the algorithms while CD1 is still the lower bound. Importantly, as for PCD, there is more oscillation of KL in the evolution curve and PCD does not perform as good as before in the smaller dataset. It the similar with WPCD, but it seems not that serious. Although the persistent algorithm has potential to reach better convergence, it is really tough to find the proper learning rate decay schedule. The results of PT are quite stable compared to the other CD-based algorithms, so parallel tempering does improve the RBMs model performance. Specially, as increasing Gibbs sampling time to 10 and the size of dataset, CD10 outperforms WCD1. Switching to WCD10, no significant improvement for the model training.

The probability distributions of the training data set are demonstrated as following figures. As the same, the distribution figures via CD10 and WCD10 are displayed.

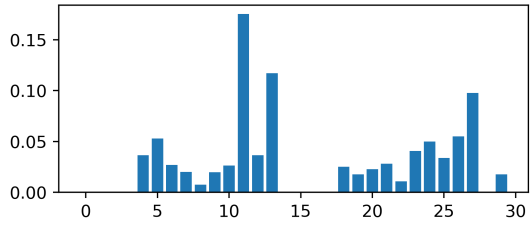


Figure 12 – BS4x4 data distribution via CD1

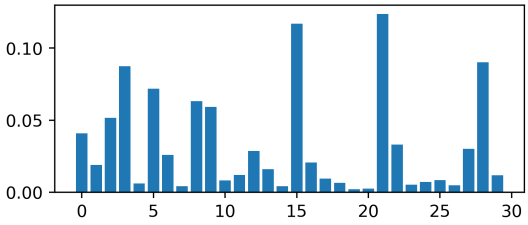


Figure 13 – BS4x4 data distribution via PCD

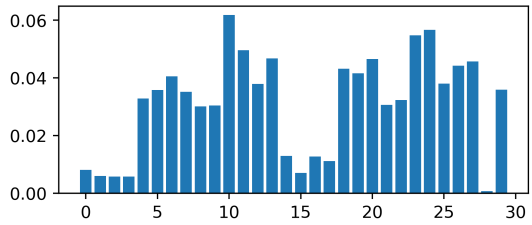


Figure 14 – BS4x4 data distribution via CD10

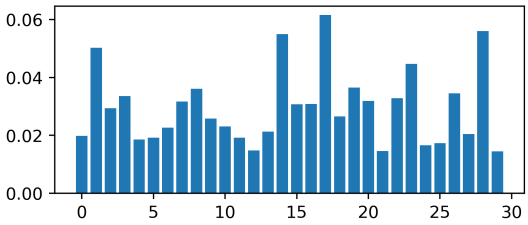


Figure 15 – BS4x4 data distribution via PT

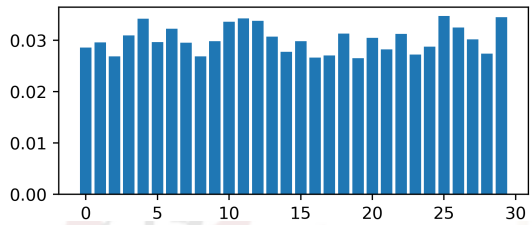


Figure 16 – BS4x4 data distribution via WCD1

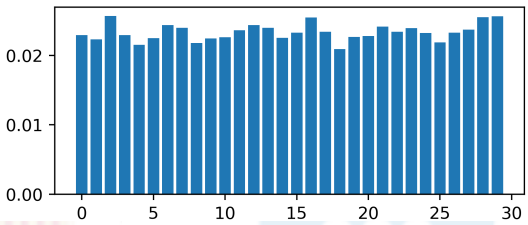


Figure 17 – BS4x4 data distribution via WPCD

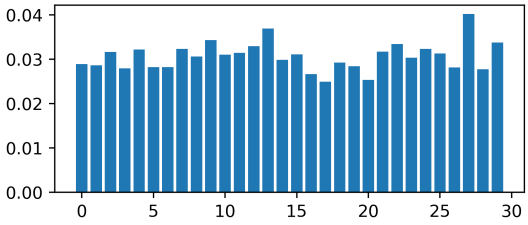


Figure 18 – BS4x4 data distribution via WCD10

As the size of the dataset increasing, the bias becomes more serious and influences the ultimate probability distribution more. Especially, in CD1 distribution several data points are assigned with no probability, and this is unacceptable. Although in PCD, PT and CD10 the bias is alleviated but still there is significant difference among the data points. Switching to WCD and WPCD, the results of these two algorithms are much more reasonable, where all the training data points are permitted with relatively equal probability.

5.4.3 Labeled-Shifter-4-11

After hyperparameter setup and fine-tuning of learning rates, the parameter configuration of different algorithms is listed in Table 10.

After a 30-time repetition, the means and standard errors of target metrics are demonstrated in Table

Table 10 – Parameter configuration of LS4-11

Opt	sample-type	lr	lr-decay	gibbs-num	chain-num
CDK	gibbs-sampling	0.03	False	1	None
PCD	gibbs-sampling	0.05	True	1	None
WCD	gibbs-sampling	0.2	False	1	None
$WPCD$	gibbs-sampling	0.05	True	1	None
PT	parallel-tempering	0.03	True	1	2

11. Also, the evolution of 5 different KL of the approximation algorithm are demonstrated in Figure 19. (* means the corresponding KL is not plotted in KL evolution figure)

Table 11 – Experiment results of LS4-11

	KL	NLL	Prob-sum	Entropy-Percentage
CDK	0.2152(0.0205)	-4.0864(0.0205)	0.948(0.0024)	0.9216(0.0087)
PCD	0.1238(0.0332)	-3.995(0.0332)	0.9609(0.0028)	0.9511(0.0076)
WCD	0.0588(0.0004)	-3.93(0.0004)	0.9436(0.0004)	0.9576(0.0003)
$WPCD$	0.2428(0.0037)	-4.114(0.0037)	0.7865(0.0019)	0.8348(0.0017)
PT	0.099(0.0053)	-3.9702(0.0053)	0.9314(0.0012)	0.9418(0.0018)
$CD10^*$	0.0501(0.0051)	-3.9213(0.0051)	0.9732(0.0026)	0.9743(0.0024)
$WCD10^*$	0.0492(0.0015)	-3.9204(0.0015)	0.9541(0.0015)	0.9651(0.0011)

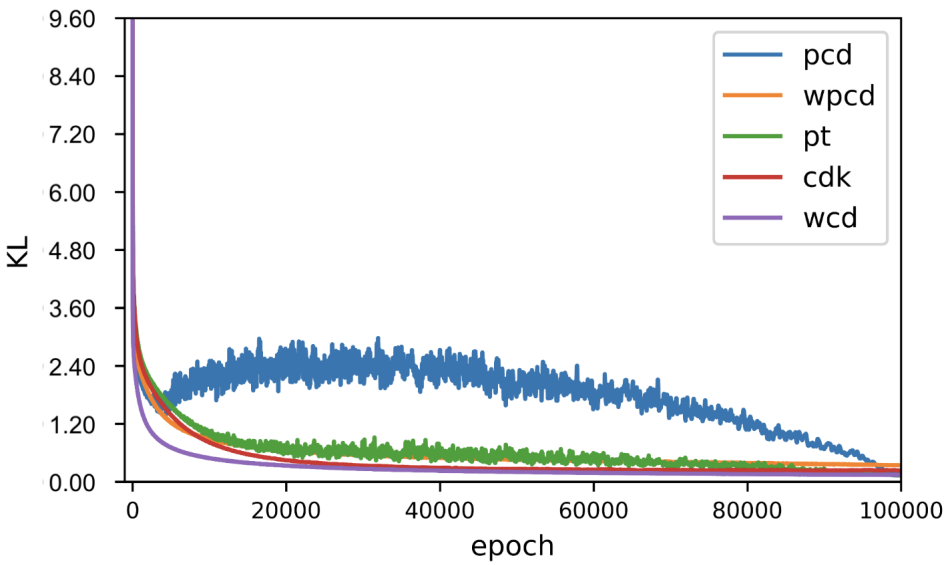


Figure 19 – The KL evolution of 5 approximation algorithms on LS4-11

According to the results in Table11, WCD performs still the best. Similarly, all the algorithms give the similar tendency as before. PCD curve presents as the arch shape, and the KL of other algorithms keep decreasing. CD10 outperforms WCD1 in KL and NLL, and based on this, it can been that as the dimension of the visible layer increases, the efficiency of CD10 would exceed WCD1 gradually. This time WCD10 outperforms CD10 slightly, but these two algorithms can be considered as the same powerful.

The probability distributions of the training data set are demonstrated as following figures. As the same, the distribution figures via CD10 and WCD10 are displayed.

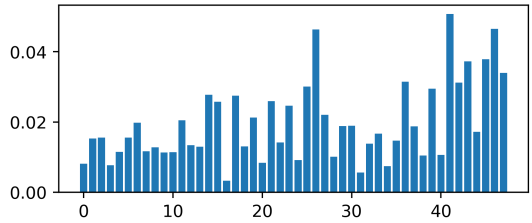


Figure 20 – LS4-11 data distribution via CD1

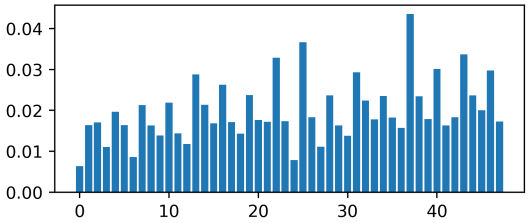


Figure 21 – LS4-11 data distribution via PCD

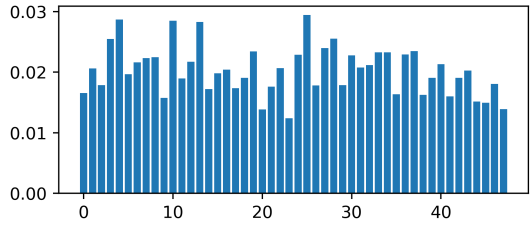


Figure 22 – LS4-11 data distribution via CD10

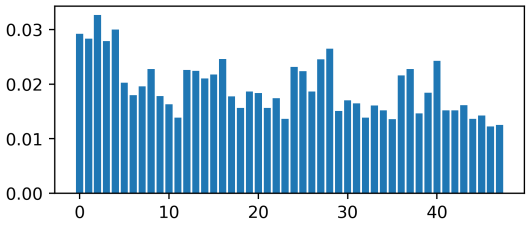


Figure 23 – LS4-11 data distribution via PT

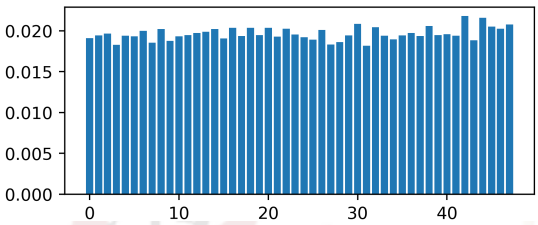


Figure 24 – LS4-11 data distribution via WCD1

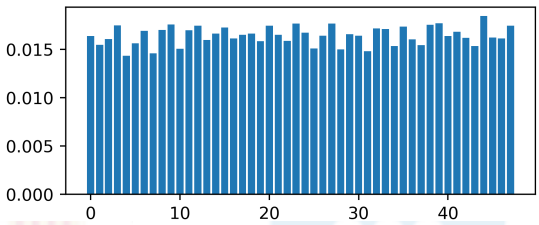


Figure 25 – LS4-11 data distribution via WPCD

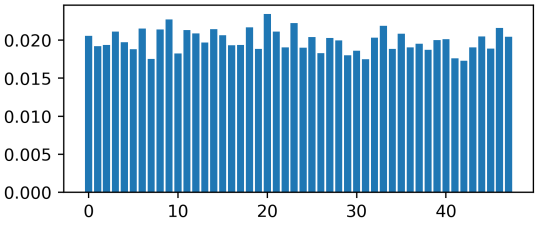


Figure 26 – LS4-11 data distribution via WCD10

According to the results of the training data point distribution, WCD and WPCD still give out the most balanced probability distribution compared to the other algorithms. Increasing the Gibbs sampling time is more significant in mitigating the bias effect.

5.4.4 Labeled-Shifter-5-13

After hyperparameter setup and fine-tuning of learning rates, the parameter configuration of different algorithms is listed in Table 12.

After a 30-time repetition, the means and standard errors of target metrics are demonstrated in Table 13. Also, the evolution of 5 different KL of the approximation algorithm are demonstrated in Figure 27. (* means the corresponding KL is not plotted in KL evolution figure)

Table 12 – Parameter configuration of LS5-13

Opt	sample-type	lr	lr-decay	gibbs-num	chain-num
CDK	gibbs-sampling	0.03	False	1	None
PCD	gibbs-sampling	0.03	True	1	None
WCD	gibbs-sampling	0.2	False	1	None
$WPCD$	gibbs-sampling	0.05	True	1	None
PT	parallel-tempering	0.03	True	1	2

Table 13 – Experiment results of LS5-13

	KL	NLL	Prob-sum	Entropy-Percentage
$CD1$	0.3257(0.0629)	-4.89(0.0629)	0.926(0.0052)	0.8918(0.0134)
PCD	0.1353(0.0165)	-4.6996(0.0165)	0.9135(0.0039)	0.9228(0.005)
WCD	0.0725(0.0001)	-4.6368(0.0001)	0.9307(0.0001)	0.9452(0.0001)
$WPCD$	0.2808(0.0032)	-4.8451(0.0032)	0.7566(0.0023)	0.8025(0.002)
PT	0.1039(0.0333)	-4.6682(0.0333)	0.9652(0.0018)	0.9577(0.0071)
$CD10^*$	0.0717(0.0059)	-4.636(0.0059)	0.9624(0.0024)	0.9636(0.0022)
$WCD10^*$	0.0583(0.003)	-4.6225(0.0005)	0.9488(0.0003)	0.95856(0.0002)

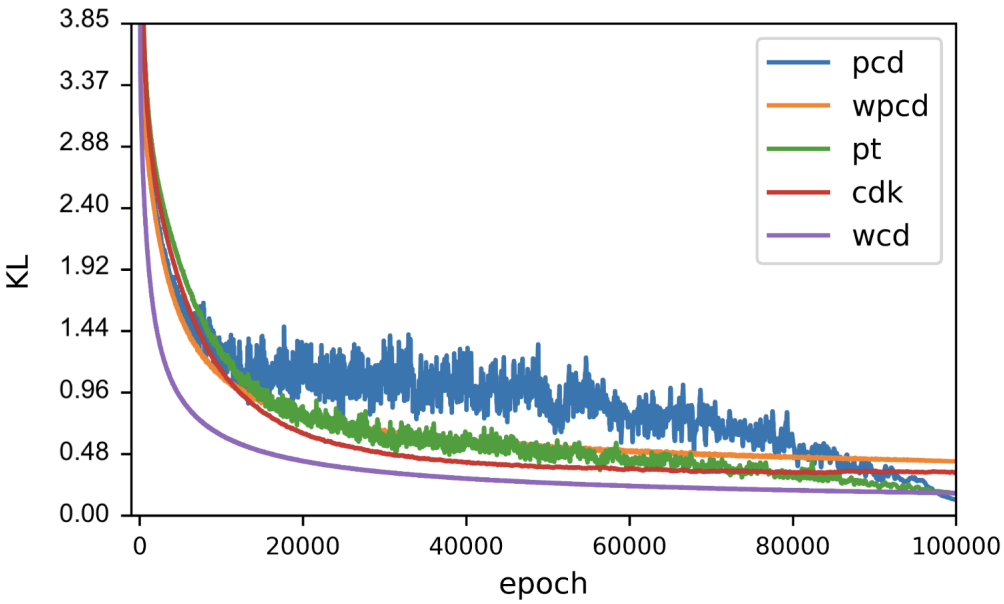


Figure 27 – The KL evolution of 5 approximation algorithms on LS5-13

This time PCD converts to be more effective than its performance in the previous experiments. So it can been seen that PCD is an unstable algorithm, and it requires more attention and effort to find the proper parameters.Similarly, WCD1 has the best and the fasted convergence, and CD10 still outperforms WCD1. As the volume of the dataset increasing, there is a significant improvement brought by WCD10, this algorithm gives out the best performance.

The probability distributions of the training data set are demonstrated as following figures. As the same, the distribution figures via CD10 and WCD10 are displayed.

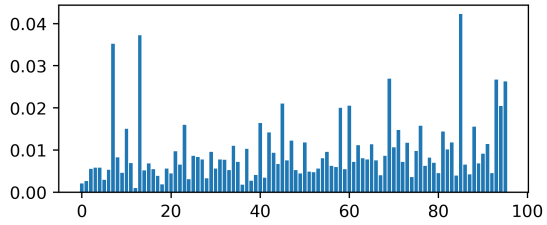


Figure 28 – LS5-13 data distribution via CD1

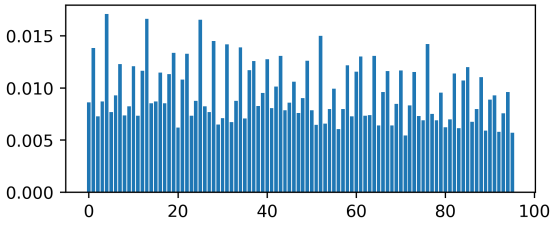


Figure 29 – LS5-13 data distribution via PCD

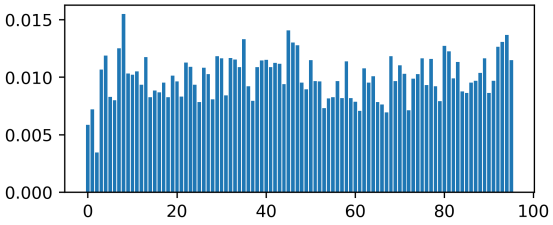


Figure 30 – LS5-13 data distribution via CD10

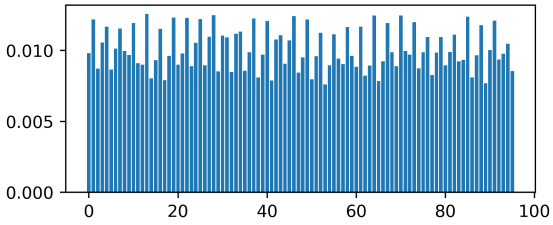


Figure 31 – LS5-13 data distribution via PT

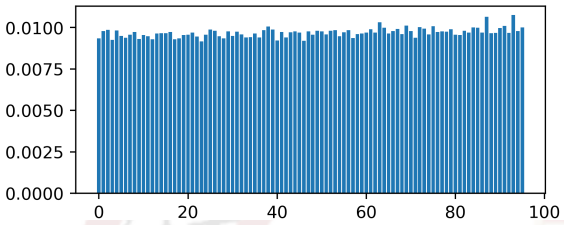


Figure 32 – LS5-13 data distribution via WCD1

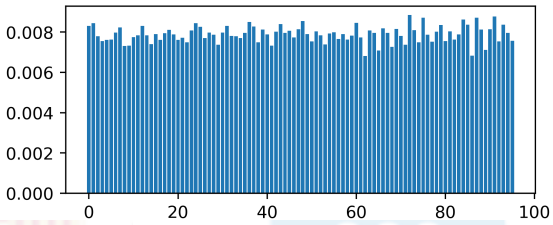


Figure 33 – LS5-13 data distribution via WPCD

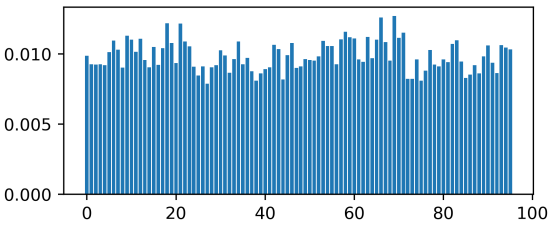


Figure 34 – LS5-13 data distribution via WCD10

This time it is easier to distinguish each algorithm's ability of alleviating the bias effect since the volume of the dataset is the largest. The bias of CD1 is really serious, but adding a persistent chain or increasing the Gibbs sampling time can help to decrease the bias, however still not enough. Switching to WCD and WPCD, the extraordinary performance of these two algorithms is really impressive, and almost in the uniform distribution shape, which is the actual distribution via exact gradient. Especially, when increasing the Gibbs sampling time, it seems more bias is introduced because there is more oscillation existing in the corresponding probability distribution figure, but still better than CD-series algorithms.

5.4.5 PT Parameter Exploration

Since in the experiment PT is configured as the lowest level, it is necessary to explore PT potential by increasing gibbs sampling times and introducing more chains. In the previous test, the gibbs sampling

time is 1 and the number of chains is 2, so this condition is taken as the standard. The searching space of the gibbs sampling time is $[1, 3, 5, 7, 9]$, and the space of the chain number is $[2, 4, 6, 8, 10]$. Utilizing the idea of grid searching, there are in total 25 groups of parameter combination. KL is selected as the metric of the exploration. In each parameter group, the KL would be computed for **5** times and then take the average as the final KL value. The weight initialization is the same as previous tests. The results are demonstrated in Figure 35,36,37,38.

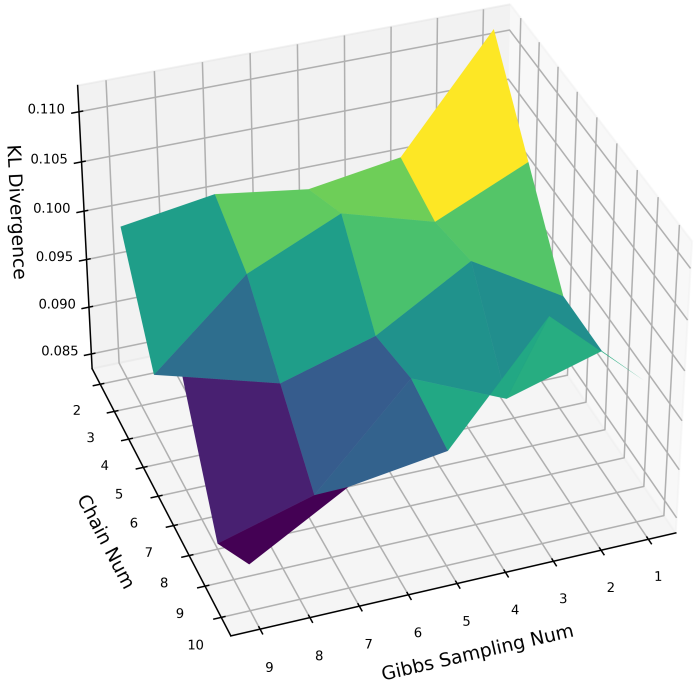


Figure 35 – The KL with different gibbs sampling times and PT chain numbers on BS3x3

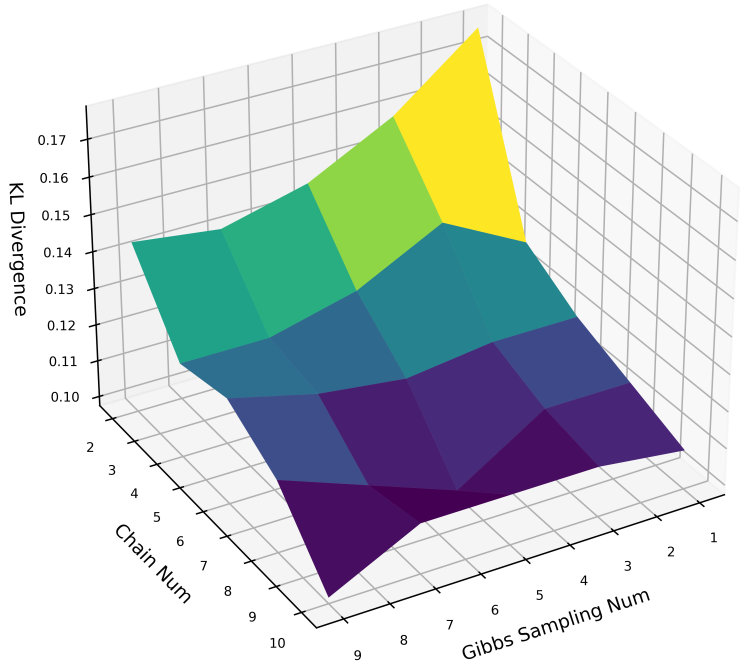


Figure 36 – The KL with different gibbs sampling times and PT chain numbers on BS4x4

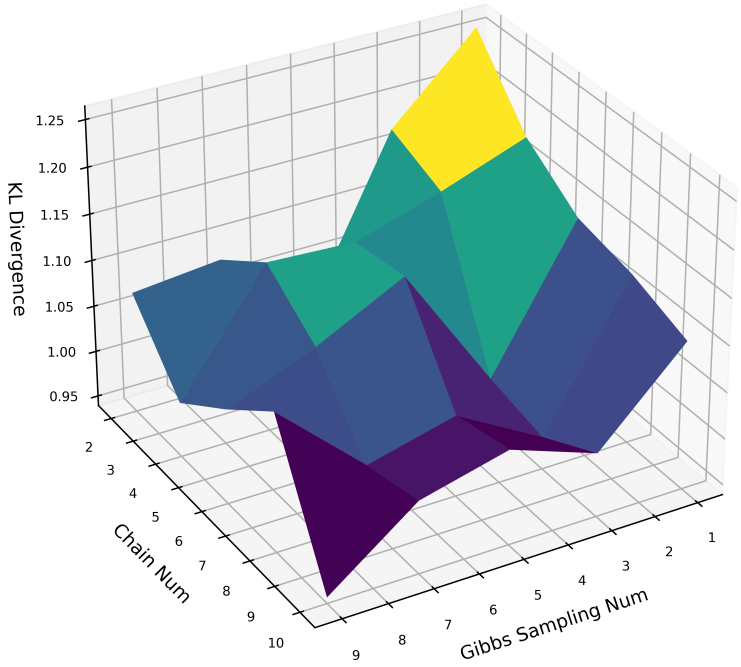


Figure 37 – The KL with different gibbs sampling times and PT chain numbers on LS4-11

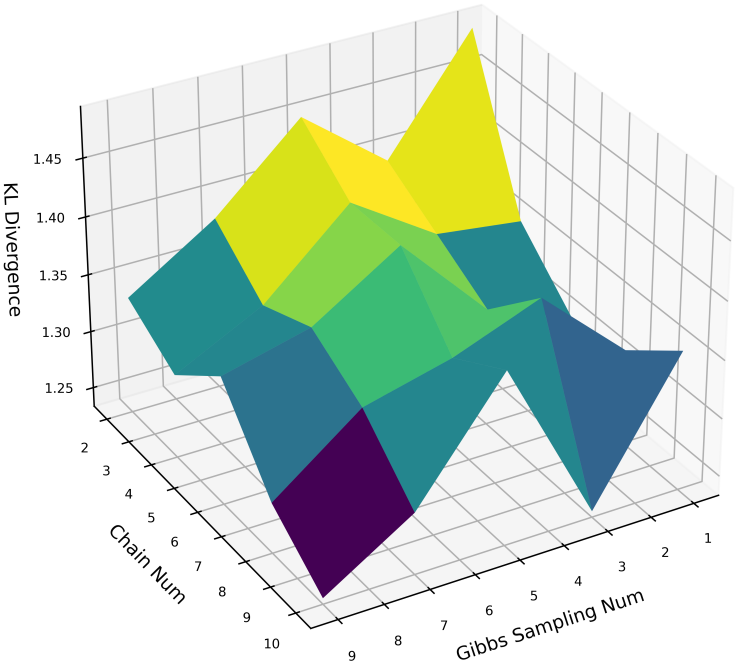


Figure 38 – The KL with different gibbs sampling times and PT chain numbers on LS5-13

As mentioned before, increasing the time of gibbs sampling does improve the model performance significantly. Similarly, increasing the number of chains, in other words adding more temperature spaces for exchanging, is also positive for optimizing the model performance. These two parameters are compatible, which is to say increasing both parameters at the same time can improve the model performance further.

In BS3x3, LS4-11 and LS5-13 figures, the results are not that solid since there is obvious fluctuation on the KL surface, but when switching to BS4x4, the biggest size dataset, the expected trend becomes more spotted. Based on this we can conclude that PT performs better and more stably on larger datasets.

Moreover, increasing the number of chains seems much more effective than increasing the gibbs sampling time, because there exists more decreasing of KL when gibbs sampling num is fixed as 1 compared to when chain number is fixed as 2. Also with chain num as 10, gibbs sampling num as 1, the corresponding KL reaches closer to the KL of the highest level PT(gibbs sampling time = 9, chain num = 10). Hence, when encountering the limitation of computation resource, increasing the chain num could be the priority choice.

5.5 Analysis of Results

5.5.1 Analysis of Convergence

Observing the tables and KL evolution figures, we can discover some significant patterns. Generally, CDK approximation, as the standard method, gives out a standard performance on KL convergence. As for PCD, it is based on CDK adding a persistent chain for successive exploration, this sort of approximation could reach an ideal convergence point, nevertheless, this sort of approach seems really sensitive with the learning rate since in all of the figures there is an arch in the KL evolution curve, and also exists obvious oscillation, which implies this algorithm requires longer training process and more fine tuning process of learning rate.

In this experiment, the PT algorithm is configured with the lowest bound, where gibbs-num = 1 and chain-num = 2. This sort of algorithm could approach to the relatively good convergence, although there exists some slight fluctuation in the evolution curve caused by different temperature exchange and random sampling. Based on the data from the tables, it can be considered as one of the most efficient approximation method. Technically, this is the standard performance of Parallel Tempering algorithm, and increasing the number of gibbs sampling and the number of chains can improve PT performance greatly, and these two hyperparameters are compatible for enhancing the model. According to the results, increasing the chain number is much more efficient for enhancing the model than increasing the Gibbs sampling time. More importantly, PT algorithm would work more stably with increasing of the visible layer dimension shown as 36, but not the data volume.

With the modification of weight assignment in WCD, WCD holds the fastest and best convergence during training process. Similarly, WPCD, adding a persistent chain for successive exploration, still has a fast speed of converging, but degenerates in fitting in the training data distribution, which causes WPCD performance is similar to CDK. Both of the approximation methods need more time to achieve higher probability sum than 0.9, even though at the beginning each metric updates really fast, especially in the end of training, the probability sum grows really slightly at the end of training period.

5.5.2 Analysis of Entropy Percentage&Bias

Switching to the metric Entropy Percentage, this metric reflects how the RBM-trained probability distribution is similar to the ideal trained probability distribution. It is affected by two factors: the assigned probability values and the entropy of training data point probability. The higher entropy-percentage values indicate the better capability of alleviating bias. For example, although less bias is produced in WPCD results, the assigned probability is at a low level, which causes the ultimate performance is similar to CD. Thus the entropy-percentage of WPCD is also similar to that of CD. Especially, WCD gives out the best performance and produces the less bias among the training data points.

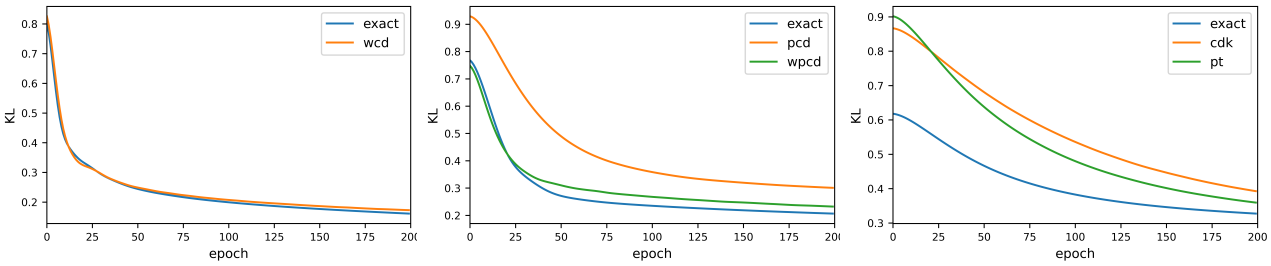
Following WCD and WPCD, PT performs relatively better and holds higher entropy percentage with smaller standard errors, which means PT's result is much more stable compared to PCD. But back to

CDK, the probability is not balanced which implies it is an unbiased learning, and this phenomenon is pointed out as the backward of Contrastive Divergence[1]. Even though there are some methods of alleviating the bias effect, they still have their own shortcomings, for example, adding a persistent chain causes a longer training process, more stricter learning rate setup and unstable results, or enlarging the gibbs sampling time brings heavier computation cost, but the performance is still not good as same as WCD or WPCD in some cases.

For CD series algorithms, increasing K can straightly reduce the bias of estimation and this is theoretically true[1]. But according to the achieved results, it seems not always holding true when using WCD. When the data volume is small, adding the sampling time can even be harmful to the learning, as shown in Figure 3. Even switching to BS4x4, WCD1's performance is similar with WCD10. However, this situation is reversed in the other 2 datasets: LS4-11 and LS5-13. WCD10 begins to outperform WCD1, as well better than CD10. Based on the discoveries, it can be concluded that the visible layer dimension N_v does not affect the WCD significantly, but it is the dataset volume plays a crucial role in enhancing the model performance. The reason is that the assumption of WCD is that only training data points are used for computing the weights to replace the exact $P(v)$, as the data number increases the estimated weights would approach to the exact probability, and the approximation can be more and more accurate. But it seems like increasing the sampling time in WCD would potentially introduce more bias as shown in Figure 34. This point needs further exploration.

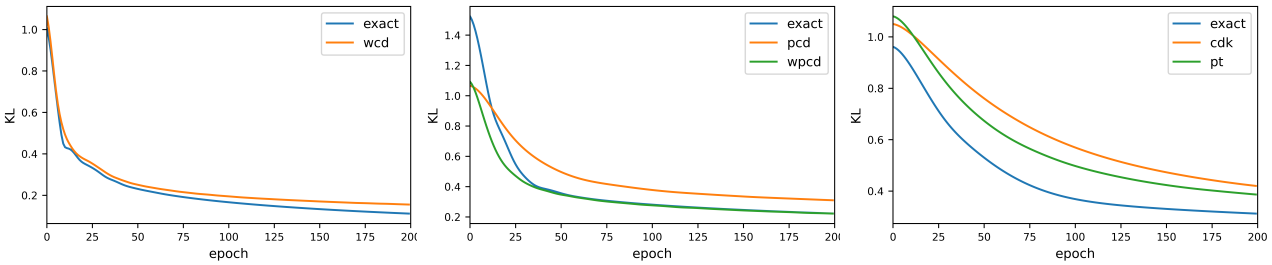
In conclusion, from the perspective of unbiased learning, WCD outperforms the rest algorithms and it should be the proper choices.

Following this conclusion, we conduct the experiment of comparing the KL evolution of the approximation and the exact gradient. According to the parameter configuration, algorithms with similar learning rates are grouped together. Thus, the 5 algorithms are divided into 3 groups: the first group is WCD, the second group contains WPCD, PCD, and the third group contains CDK and PT. Especially, the learning rate of the exact gradient algorithm is set up as the minimum of the corresponding learning rates in the group. Based on the prior experiment experience, the gap between approximated KL and exact KL would emerge anyway, so only the first 200 epochs KL records are collected to check the model fitting. The corresponding hyperparameters and parameters are remained the same as the default configuration for each dataset and each algorithm. In order to avoid some random extreme cases, the algorithm would be repeated for **30** times and the average is taken as the final metric value. The results are demonstrated as Figure 39, 40, 41, 42.



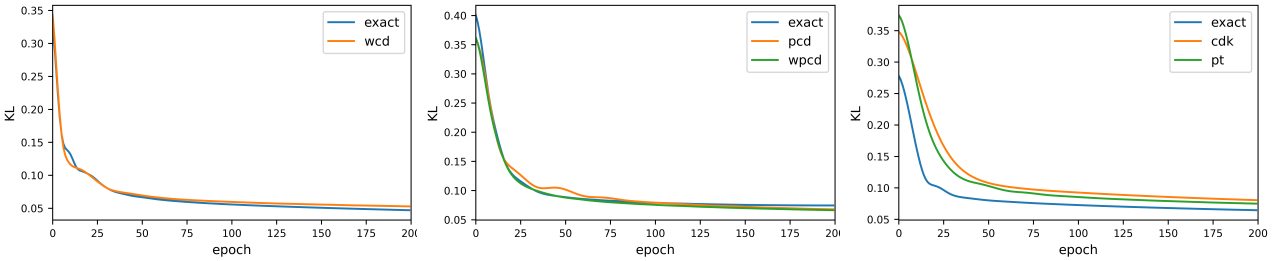
((a)) ExactGrad.&WCD ((b)) ExactGrad.&PCD&WPCD ((c)) ExactGrad.&CDK&PT

Figure 39 – KL evolution comparison of BS3x3



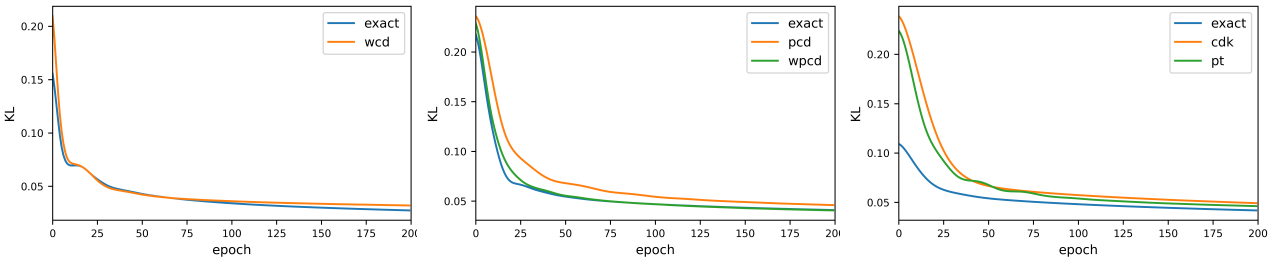
((a)) ExactGrad.&WCD ((b)) ExactGrad.&PCD&WPCD ((c)) ExactGrad.&CDK&PT

Figure 40 – KL evolution comparison of BS4x4



((a)) ExactGrad.&WCD ((b)) ExactGrad.&PCD&WPCD ((c)) ExactGrad.&CDK&PT

Figure 41 – KL evolution comparison of LS4-11



((a)) ExactGrad.&WCD ((b)) ExactGrad.&PCD&WPCD ((c)) ExactGrad.&CDK&PT

Figure 42 – KL evolution comparison of LS5-13

According to the information retrieved from the KL comparison figures, it is easy to discover that weight-assigned approximation is so excellent that the corresponding approximated gradients follow the trace of the exact gradient at the beginning of the training process, while the others, CDK,PCD and PT, become significantly distinguished at the very first beginning. It is worth noting that PCD has potential in approaching to the exact gradient road which benefits from the existence of persistent chain, but not stable enough. In summary, there are two effective ways of pushing approximation to the exact path so that nerf the bias in the approximation: approximating via weight-assignment and adding a persistent chain.

5.5.3 Analysis of Gradient Approximation

Since the previous discoveries are based on the results of approximation, mainly focus on KL, there is a need to explore from the perspective of the gradient. In order to analyze and conduct comparison of the approximated gradients and exact gradients quantitatively, "cosine similarity" is selected as the basic metric to measure how similar the gradients are. As the matter of fact that there exists cases where

the norm of the matrix is 0 sometimes, we add a smooth constant ϵ(ϵ=1e-12) in the cosine similarity definition, and the modified cosine similarity goes as following(40).

$$sim(A, B) = \frac{< A, B > +\epsilon}{\|A\| * \|B\| + \epsilon} \tag{40}$$

According to the definition of the RBM energy function, there are in total 3 parameters to learn from the data: W, α and β, hence the plan is to compute these three parameters' gradients in each epoch with respective to the approximation algorithm and the exact gradient algorithm. In order to get relatively reasonable comparison results, each algorithm would be applied on the same dataset with random initial W, α and β for 30 times. Also the initial weights of the exact gradient algorithm and those of the approximated algorithm are set as default parameters, and the epoch is set up as 3000 (epoch may be extended if not reaches convergence).

The results are demonstrated as following Figure44,45,46, and the legend of 5 algorithms is displayed in Figure 43.

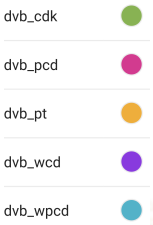
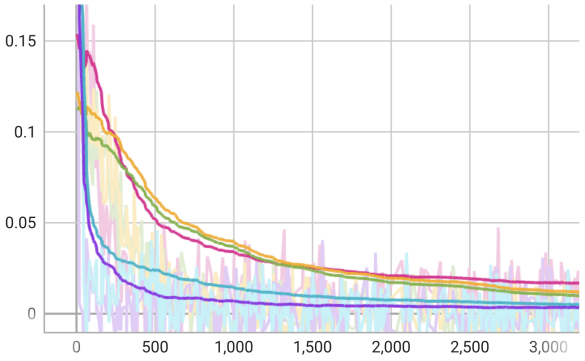
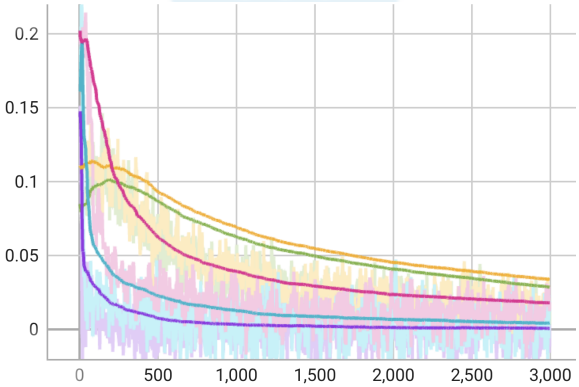


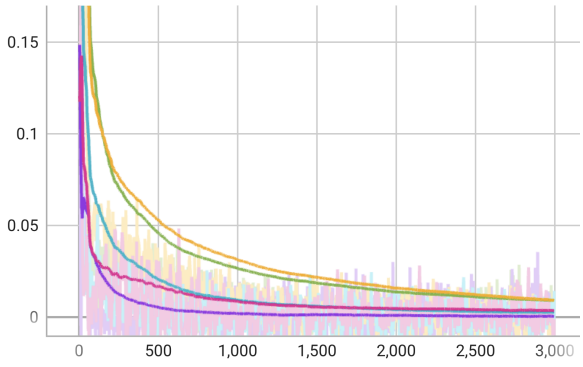
Figure 43 – The legend of 5 algorithms



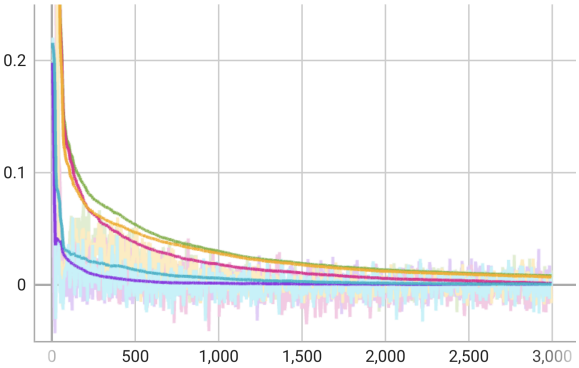
((a)) cos-sim evolution of BS3x3



((b)) cos-sim evolution of BS4x4

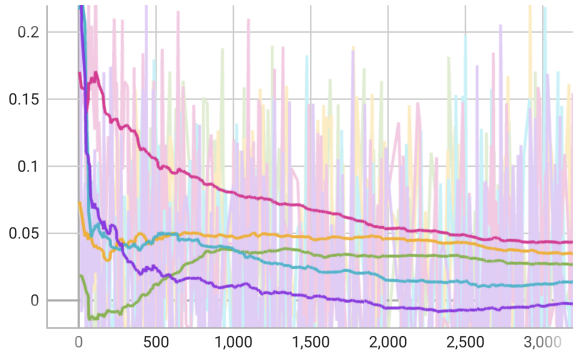


((c)) cos-sim evolution of LS4-11

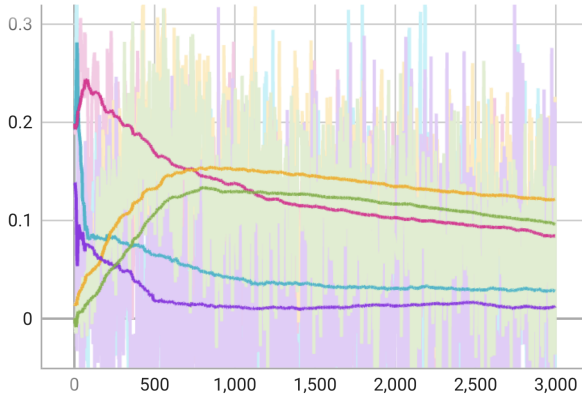


((d)) cos-sim evolution of LS5-13

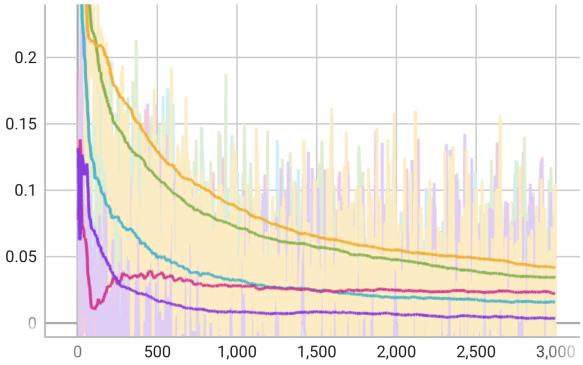
Figure 44 – cosine similarity evolution comparison of W



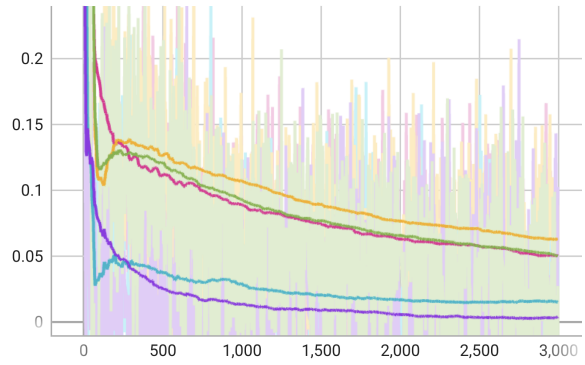
((a)) cos-sim evolution of BS3x3



((b)) cos-sim evolution of BS4x4

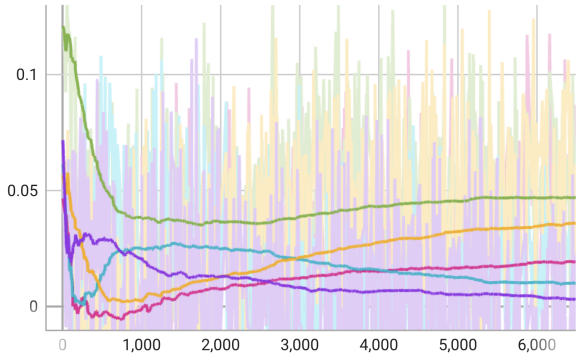


((c)) cos-sim evolution of LS4-11

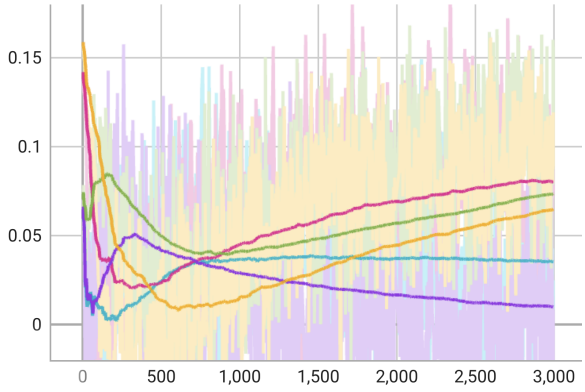


((d)) cos-sim evolution of LS5-13

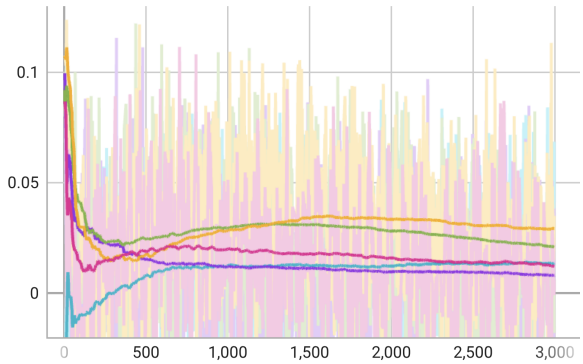
Figure 45 – cosine similarity evolution comparison of α



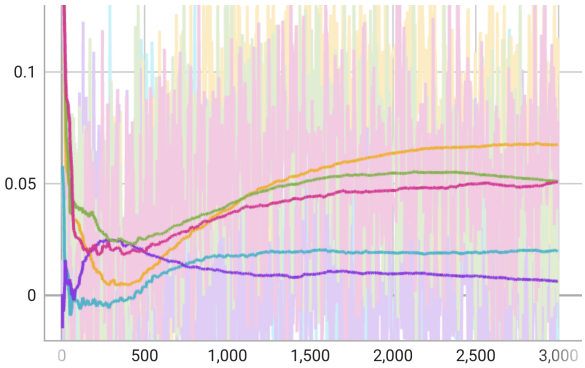
((a)) cos-sim evolution of BS3x3



((b)) cos-sim evolution of BS4x4



((c)) cos-sim evolution of LS4-11



((d)) cos-sim evolution of LS5-13

Figure 46 – cosine similarity evolution comparison of β

Among the visualization results, we find that all of the cosine similarities reach convergence closed to 0, which implies the approximated gradients after 3000 epochs are becoming orthogonal with the exact gradient, in other words they have no relation. It can be understood that the direction of approximation has shifted and rushed to another optimal point. Therefore, none of the estimation is able to track the exact gradient all the way, and there is always estimation shift in the approximation.

It is worth noting that no matter which model parameter is, W, α or β, there is a shared pattern in the evolution of cosine similarity. The curves of WCD and WPCD roughly advance in a similar path, oppositely the curves of CDK and PT are in a totally different path. Following their own paths, all of the algorithms' curves reach the convergence, importantly, the ultimate converging cosine similarity of WCD and of WPCD are all smaller than those of CD and of PCD. Especially in the figures of W, the cosine similarity curves of WCD and WPCD are always below the curves of CD and PT. As for PCD, the curve of PCD seems unpredictable since in LS4-11 PCD roughly follows the trace of WCD and WPCD, but does not perform as well as WCD or WPCD. Reversely, in the other 3 datasets the PCD curve turns to follow the trace of CDK and PT and gives out reasonable results compared to those of CDK and PT. So it can be said that the evolution process of WCD and WPCD is independent with the process of CDK and PT, which means WCD-series algorithms and CD-series algorithms are heterogeneous. Possibly, this is the reason why adding a persistent Markov chain does not improve the WCD performance.

In summary, it can be considered that the way of weight-assignment changes the approximation gradient path, but enhances the model learning.

6 Conclusion

Through the implementation of RBMs, and the conduction of evaluating and exploring experiments, we get the relatively comprehensive understanding of the gradient approximation algorithms: CDK, PCD, PT, WCD and WPCD. According to the achieved results, we can draw some general conclusions about RBMs and RBM approximating algorithms.

First, we compare two types of input variable for RBMs, $[0, 1]$ and $[-1, 1]$. Between these two sorts of input, RBMs with $[0, 1]$ present more robustness to the random initialized weights and produce more stable results. Since the target of this project is to compare different gradient approximation algorithms, the based RBMs are supposed to be resilient and enduring and capable of fitting in different scale of data. Therefore, $[0, 1]$ as the input type is selected.

If the Gibbs sampling time is set as 1 and no limitation of using training proper techniques, Weighted Contrastive Divergence has the fastest and best convergence compared to the other gradient approximation algorithms, where the chain num is set as 2 for PT. Specifically, the results of WCD1 can be the same good as those of CD10. According to the experiment records, WCD1 outperforms CD10 when the dimension of the visible layer is small, roughly estimating the threshold is 10. Moreover, increasing the Gibbs sampling time in WCD does not enhance the RBM performance straightly, but in the situation where when the data volume is large, WCD10 would outperform WCD1 and CD10. The reason is that more training data points could cover more possible visible units, whose total number is 2^{N_V}, so that the estimated weights can be more similar with $P(v)$ in the exact gradient.

As for the bias in approximation, there is the least bias in the approximation via WCD according to the experiment results. Although WPCD can produce relatively equal probability, the probability sum does not reach the CDK standard line, only around 0.75. The persistent chain seems to suppress WCD power, and with increasing of the dataset size it is more obvious. Specially, in WCD10 increasing the Gibbs sampling time seems to introduce more bias than WCD1. Oppositely, the persistent chain can improve the performance of CDK significantly and also alleviates the bias effect, but not good as WCD. The shortcoming of PCD is that the corresponding results are not stable enough and really sensitive with the learning rate. Switching to PT, even set as the lowest standard(chain num is 2), PT still gives out a significant improvement of model performance compared to CD1, and also mitigates the effect of bias, better than PCD but less than WCD.

We also have drawn general conclusions of hyperparameter configuration in PT, the Gibbs sampling time and the number of Temperature chains. Based on the results of searching in different hyperparameter groups, we find that PT works much more stably with increasing of the visible layer dimension rather than the dataset volume. Additionally, enlarging the number of chains and the sampling time are compatible of enhancing the model performance. More specifically, the increasing of the chain number is more effective and efficient than increasing the Gibbs sampling time. Therefore, adding the chain number should be the prior choice when using PT in order to improve the model performance.

Through observing the KL evolution comparison between the exact gradient algorithm and the approximation gradient algorithms, we find that in the first 200 epochs the KL curves of WCD and WPCD almost follow the trace of the exact KL curve. It must be pointed out that the most dramatic decline of KL happens in the first hundreds of training epochs, so this is also a potential evidence why WCD is the best approximation algorithm. Afterwards, the cosine similarity between the approximated gradient and the exact gradient are compared, we conclude that the WCD-series algorithms and CD-series algorithms are heterogeneous.

In summary, considering the model performance and least bias effect, WCD is the prior choice for RBMs.

References

[1] Yoshua Bengio **and** Olivier Delalleau. "Justifying and generalizing contrastive divergence". **in***Neural computation*: 21.6 (2009), **pages** 1601–1621.

[2] Guillaume Desjardins **andothers**. "Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines". **in***Proceedings of the thirteenth international conference on artificial intelligence and statistics*: JMLR Workshop **and** Conference Proceedings. 2010, **pages** 145–152.

[3] David J Earl **and** Michael W Deem. "Parallel tempering: Theory, applications, and new perspectives". **in***Physical Chemistry Chemical Physics*: 7.23 (2005), **pages** 3910–3916.

[4] Richard P Feynman. "Simulating physics with computers". **in***Feynman and computation*: CRC Press, 2018, **pages** 133–153.

[5] Asja Fischer **and** Christian Igel. "Bounding the bias of contrastive divergence learning". **in***Neural computation*: 23.3 (2011), **pages** 664–673.

[6] Asja Fischer **and** Christian Igel. "Empirical Analysis of the Divergence of Gibbs Sampling Based Learning Algorithms for Restricted Boltzmann Machines." **in***ICANN (3)*: 6354 (2010), **pages** 208–217.

[7] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". **in***Neural computation*: 14.8 (2002), **pages** 1771–1800.

[8] Renjie Liao **andothers**. "Gaussian-Bernoulli RBMs Without Tears". **in***arXiv preprint arXiv:2210.10318*: (2022).

[9] Enrique Romero **andothers**. "Weighted contrastive divergence". **in***Neural Networks*: 114 (2019), **pages** 147–156.

[10] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory.* techreport. Colorado Univ at Boulder Dept of Computer Science, 1986.

[11] Tijmen Tieleman. "Training restricted Boltzmann machines using approximations to the likelihood gradient". **in***Proceedings of the 25th international conference on Machine learning*: 2008, **pages** 1064–1071.

[12] Ashish Vaswani **andothers**. "Attention is all you need". **in***Advances in neural information processing systems*: 30 (2017).

Appendix.

The Project Repository The RBMs and all the gradient approximation algorithms are implemented in Python, and also established from the scratch by Numpy. The codes are stored in the Git-Hub repository `https://github.com/lihongji98/Restricted-Boltzmann-Machine-RBM`.

The records of Hyperparameter Searching via CD1

Algorithm	LearningRate	HiddenDimension	KL	NLL	ProbSum
CD1	0.01	9	0.4844(0.006)	-3.1260(0.072)	0.8709(0.016)
CD1	0.01	18	0.4312(0.006)	-3.0733(0.042)	0.9426(0.031)
CD1	0.01	27	0.434(0.007)	-3.0796(0.058)	0.9206(0.057)
CD1	0.01	36	0.4466(0.003)	-3.0857(0.026)	0.9139(0.018)
CD1	0.01	45	0.476(0.004)	-3.1448(0.048)	0.9277(0.009)
CD1	0.001	9	0.4522(0.002)	-3.0909(0.003)	0.8847(0.028)
CD1	0.001	18	0.4648(0.002)	-3.1065(0.023)	0.9036(0.032)
CD1	0.001	27	0.4774(0.004)	-3.1299(0.118)	0.9271(0.018)
CD1	0.001	36	0.546(0.004)	-3.1741(0.096)	0.9155(0.018)
CD1	0.001	45	0.5166(0.001)	-3.1553(0.012)	0.9166(0.018)
CD1	0.0001	9	1.7514(0.011)	-4.3907(0.157)	0.2090(0.021)
CD1	0.0001	18	1.1718(0.001)	-3.8103(0.014)	0.3793(0.011)
CD1	0.0001	27	0.945(0.001)	-3.5834(0.023)	0.4883(0.017)
CD1	0.0001	36	0.8806(0.002)	-3.5209(0.030)	0.5467(0.031)
CD1	0.0001	45	0.854(0.001)	-3.4919(0.015)	0.5798(0.010)

The records of Hyperparameter Searching via CD10

Algorithm	LearningRate	HiddenDimension	KL	NLL	ProbSum
CD10	0.01	9	0.035(0.0023)	-2.6732(0.0160)	0.9925(0.0018)
CD10	0.01	18	0.0364(0.0092)	-2.6757(0.0071)	0.9877(0.0044)
CD10	0.01	27	0.0322(0.0046)	-2.6707(0.0014)	0.9917(0.0036)
CD10	0.01	36	0.0308(0.0023)	-2.6702(0.0011)	0.9874(0.0057)
CD10	0.01	45	0.0434(0.0115)	-2.6817(0.0060)	0.9872(0.0052)
CD10	0.001	9	0.0602(0.0092)	-2.6997(0.0058)	0.957(0.0023)
CD10	0.001	18	0.0476(0.0115)	-2.6871(0.0076)	0.9771(0.0052)
CD10	0.001	27	0.0462(0.0046)	-2.6859(0.0019)	0.9818(0.0006)
CD10	0.001	36	0.0504(0.0069)	-2.6916(0.0039)	0.979(0.003)
CD10	0.001	45	0.056(0.0046)	-2.6942(0.0046)	0.9814(0.003)
CD10	0.0001	9	1.5526(0.1035)	-4.1915(0.0481)	0.2202(0.0102)
CD10	0.0001	18	0.7182(0.0483)	-3.2909(0.0277)	0.4918(0.0135)
CD10	0.0001	27	0.4606(0.0253)	-3.0995(0.0203)	0.6283(0.0118)
CD10	0.0001	36	0.35(0.0276)	-2.9894(0.0208)	0.7073(0.0147)
CD10	0.0001	45	0.287(0.0368)	-2.9262(0.0199)	0.7645(0.0130)

The records of Learning Rate Decay Searching via PCD

Algorithm	LR	LRDecay	WeightDecay	KL	NLL	ProbSum
PCD	0.0001	1.00E-11	2.50E-05	2.062(0.108)	-4.702(0.144)	0.148(0.013)
PCD	0.0001	1.00E-10	2.50E-05	2.482(0.154)	-5.281(0.243)	0.088(0.028)
PCD	0.0001	1.00E-09	2.50E-05	3.560(0.391)	-6.060(0.104)	0.0399(0.009)
PCD	0.01	1.00E-11	2.50E-05	0.527(0.174)	-3.121(0.109)	0.878(0.031)
PCD	0.01	1.00E-10	2.50E-05	0.281(0.276)	-2.947(0.109)	0.875(0.126)
PCD	0.01	1.00E-09	2.50E-05	0.068(0.012)	-2.705(0.006)	0.941(0.003)
PCD	0.001	1.00E-11	2.50E-05	0.162(0.025)	-2.801(0.007)	0.878(0.012)
PCD	0.001	1.00E-10	2.50E-05	0.414(0.078)	-3.046(0.065)	0.655(0.032)
PCD	0.001	1.00E-09	2.50E-05	1.766(0.195)	-4.404(0.0101)	0.185(0.021)
PCD	0.0025	1.00E-11	2.50E-05	0.215(0.285)	-2.944(0.189)	0.938(0.053)
PCD	0.0025	1.00E-10	2.50E-05	0.101(0.004)	-2.741(0.005)	0.915(0.002)
PCD	0.0025	1.00E-09	2.50E-05	0.662(0.076)	-3.245(0.069)	0.523(0.016)