UNIVERSITY OF CALIFORNIA, SAN DIEGO

Power Efficient Image Classification and Generation using Fixed Point Gibbs Sampling

A thesis submitted in partial satisfaction of the
requirements for the degree
Masters of Science

in

Electrical Engineering
(Intelligent Systems, Robotics and Control)

by

Chih-yin Kan

Committee in charge:

Professor Ken Kreutz-Delgado, Chair
Professor Truong Nguyen
Professor Piya Pal

2018

The thesis of Chih-yin Kan is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2018

EPIGRAPH

Torture the data, and it will confess to anything.
— Ronald Coase

No computer has ever been designed that is ever aware of what its doing;
but most of the time, we aren't either.
— Marvin Minsky

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Ken Kreutz-Delgado for the continuous support of my Masters thesis and research, for his motivation, enthusiasm, and immense knowledge in the field. With his guidance, I learned a great deal in the research and the writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Truong Nguyen, and Prof. Piya Pal, for their support, insightful comments, and challenging questions.

My sincere thanks also goes to my mentor, Srinjoy Das, for offering me the research opportunities and encouraging me to take on the thesis project. He was very helpful, approachable and knowledgeable and guided me through the completion of my thesis. I also appreciate my fellow labmates in UCSD research Group: Xinyu Zhang, Ian Colbert and Ojash Neopane, for their valuable feedback and insights during meetings and discussions.

I would like to thank Jonas Chen and Professor Andrew Kahn for providing access to VLSI tools, scripts, libraries and other support infrastructure. I was able to conduct my research at the Calit2 Qualcomm Institute Pattern Recognition Laboratory (PRLab), and in particular, I would like to thank John Graham and Tom DeFanti of the PRLab for providing the computing infrastructure for the thesis project.

Last, but not least, I would like to thank my family and friends who supported me in one way or another throughout my Masters study.

VITA

2016	B.S. in Electrical and Computer Engineering, University of California, San Diego
2018	M.S. in Electrical and Computer Engineering, University of California, San Diego

PUBLICATIONS

Arnav Acharyya, Dustin Hudson, Ka Wai Chen, Tianjia Feng, Chih-yin Kan, Truong Nguyen, "Depth Estimation from Focus and Disparity", *IEEE (ICIP)*, 2016.

Abstract of the Thesis

Power Efficient Image Classification and Generation using Fixed Point Gibbs Sampling

by

Chih-yin Kan
Masters of Science in Electrical Engineering
(Intelligent Systems, Robotics and Control)

University of California San Diego, 2018

Professor Ken Kreutz-Delgado, Chair

Machine learning-based algorithms are essential tools used to extract and analyze information for applications such as classification, pattern recognition, denoising and reconstruction. It has become commonplace that smart, connected, and Internet-of-Things (IoT) based devices, perform machine learning algorithms in the cloud. However, with the increase in the number of connected devices, processing information on the cloud can encounter privacy, latency, and reliability problems. As a result, hardware for machine learning on the "edge" of the cloud is gaining popularity, since having a real-time processor that is locally embedded onto devices and sensors can ameliorate these issues, as well as decrease the power requirements associated with continuous, sustained connectivity to the cloud. In the designing of the locally embedded hardware algorithms, there is a requirement for maintaining accuracy close to cloud-based algorithms (which serves as ideal benchmarks), while addressing limitations due to hardware cost, size, and

power consumption. Thus, it can be challenging to design a hardware for machine learning purposes. In this thesis, we study the power-at-performance efficiency of Restricted Boltzmann Machine-based machine learning algorithms using Gibbs samplers implemented with fixed point approximations and functional approximations of sigmoid activation functions. We discuss how hardware designers can determine the trade-offs between using the different styles of activation function approximations and different levels of bitwidth quantization, and develop a design methodology which we implement and verify on a Verilog environment. Metrics are developed for comparing the performance of both Discriminative and Generative RBM, various choices of bitwidth level and style of activation approximations are explored to obtain a good performance-power trade-off.

Chapter 1

Introduction

The use of Big Data is exploding and its use to a large extent a result of the increase in the usage of sensors and the Internet of things (IoT). The number of objects that will be connected to the internet is expected reach 50 billion by 2020 [1]. This explosive growth is predicted to generate approximately 507.5 Zettabytes of data annually by 2020 [2]. In many applications, machine learning algorithms are utilized to process and extract useful information from these data. Because of the high processing requirements, systems often have to be connected to the cloud in order to have access to high-performance machine learning algorithms, but doing these computations on the cloud can be problematic, due to issues such as the high volume of data flow, poor latency rate, and the lack of privacy. A more efficient method is to train the models on the cloud, and have a real-time processor that is embedded onto devices and sensors at the "edge of the cloud" to locally process the data using the trained model.

However, a major challenge with implementing machine learning algorithms onto local hardware is power consumption. Due to the constraint in cost and size of the hardware and local power supplies (such as batteries), minimizing the power consumption on the chip, while keeping the performance of the model reasonably high is essential. In this thesis, we propose a method to increase the power efficiency of the local hardware implementation by using fixed point Markov Chain Monte Carlos (MCMC) Gibbs sampling and functional approximation of activation functions. By limiting the number of bits that are used and the functional approxi-

mations made in the implemented, trained algorithm, the number of registers used in the local hardware can be reduced significantly, consequently reducing power consumption. To test our methodology, the model is quantized and observed at 5 different bit widths: 4-bit, 8-bit, 12-bit, 16-bit, and 64-bit, and the effects of the fixed point Gibbs samplers are realized on both the discriminative and generative model of a Restricted Boltzmann Machine (RBM) for the purpose of demonstration. The efficiency of each quantized model is computed using test metrics devised for this purpose and compared to identify a quantization level that is suitable for the application. A Verilog model of the resulting hardware implementation is used to ascertain the efficacy of the proposed procedure and the ability to attained a desired level of power-at-performance. See Figure 1.1 for a graphical view of the various aspects of the proposed methodology.

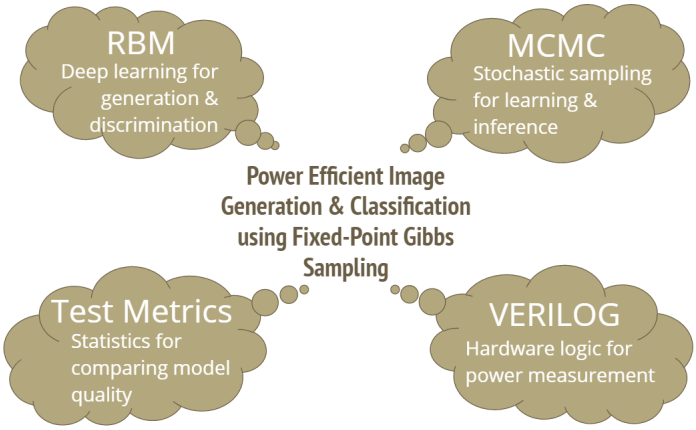

Figure 1.1: Overview of the various aspects of hardware design which are addressed and utilized in this thesis

The remainder of the thesis is structured as follows. Chapter 2 introduces the Markov Chain Monte Carlo (MCMC) method and one of its most widely used algorithms - the Gibbs sampler. The quantization scheme is also described in this chapter. Gibbs sampling with limited bitwidth is used on the RBM, and the corresponding kernel is approximated using two different methods. Chapter 3 will introduce the RBM and talk about the difference in implementation and application of the generative model version and the discriminative model version of the RBM. Two different quantized activation function approximations are proposed

and their effects are demonstrated on the RBM models. Test metrics are developed for comparing the quality of the approximated models. Experiments and Verilog hardware implementations are described and the results are shown and analyzed in Chapter 4 and 5, for the discriminative model and the generative model respectively. Lastly, Chapter 6 provides conclusions and suggests possible future extensions to the work described in this thesis.

Chapter 2

Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a powerful inference technique for sampling from high dimensional probability distributions. The term *Monte Carlo* in MCMC refers to performing a stochastic sampling, which means that we want to draw from a specified distribution, while the method that is used to perform this stochastic sampling is based on drawing repeatedly from the transition probabilities describing a *Markov chain* (Figure 2.1). See reference [3] for a detailed discussion of MCMC, the Gibbs sampler, and applications to the Restricted Boltzmann Machine (RBM).

To perform MCMC, we need to sample from a specified distribution $P(x)$ through the construction of a Markov Chain that enables us to perform a random walk on the state space $\mathcal{X}$. The time spent in each state should be proportional to the target density $P(x)$, and to ensure that this occurs requires that the state-transition probability $P(x_i|x_{i-1})$.

$$x_i|x_{i-1}, ..., x_0 \sim P(x_i|x_{i-1}, ..., x_0) = P(x_i|x_{i-1})$$

With the use of Markov Chain Monte Carlo, the computation of the value of each neuron is greatly simplified.

Once we have properly constructed the Markov Chain, if it is run for a sufficient large number of transition steps, the state values attain their steady state behavior, which corresponds to being drawn from the target distribution $P(x)$. The process of settling down to the steady-state behavior is known as *burn-in*. Suppose we

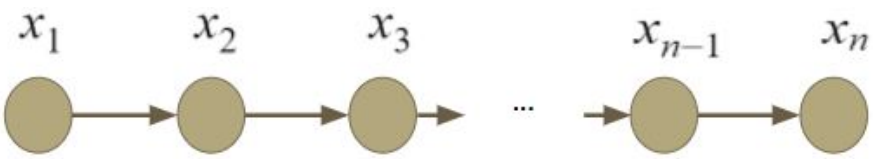

Figure 2.1: Markov Chain

have a function $f(x)$. The expected value of the function is

$$E[f(x)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

and, as a consequence of the law of large numbers, equality in some stochastic sense as $N \to \infty$. Thus, according to the law of large numbers [4], if N is a large number, i.e. $N \gg 1$, then burn-in can be assumed and the approximation is valid. Likewise, the burn-in property accounts for the assumption that the dependency of the next state of the neuron x_i on the neurons further away than the current is insignificant and thus Markov Chains often can be assumed to be almost memoryless (i.e., the correlation time-length often can be viewed as relatively inconsequential for $N \gg 1$).

2.1 Gibbs Sampling

Gibbs sampling is an MCMC technique that is commonly used to generate sequential stochastic samples from a transition probability constructed from a conditional distribution derived from the target static distribution $P(x)$ of an RBM neural network, where x denotes the instantaneous state of the RBM [3].

For example, given a joint sample $\mathbb{x}^S$ of all the RBM variables at time ("stage") S, we are able to generate a new sample $\mathbb{x}^{S+1}$ by sampling each component with respect to the most recent values of the other variables, i.e.

$$x_i^{S+1} \sim p(x_i|\mathbb{x}_{\setminus i})$$

where $\mathbb{x}_{\setminus i}$ is the set of the most recent (step S) variables other than x_i, and $p(x_i|\mathbb{x}_{\setminus i})$ is the fully conditional for the variable i.

In this thesis, we use the Modified National Institute of Standards and Technology (MNIST) dataset as the data to which we wish to fit an RBM to serve as a generative model. MNIST is a large dataset consisting hand-written digits from 0 to 9. The entire dataset has 60000 training images and 10000 testing images. For the interest of process time, the first 5000 images in the MNIST training dataset are used for training and the first 1000 images in the MNIST testing dataset are used for testing in this thesis.

Each MNIST image is made up of 28×28 pixels, giving us 784 neurons, one per pixel. Here, our random vector $\mathbf{X}$ is 784-dimensional, consisting of random variables X_i, taking realization values x_i, for $i = 1, 2, ..., 784$, i.e. $X_1, X_2, ..., X_{784}$ ϵ $\mathbf{X}$. Suppose the conditional distribution of X_i, given the values of all the other other components $X_{\setminus i}$ of $\mathbf{X}$, is known. To implement Gibbs sampling, the conditional distribution for each component of $\mathbf{x}$ is computed, and the new sample $\mathbb{x}^{S+1}$ is generated as follows [3]:

$$
\begin{aligned}
x_1^{S+1} &\sim p(x_1|x_2^S, x_3^S, ..., x_{784}^S) \\
x_2^{S+1} &\sim p(x_2|x_1^{S+1}, x_3^S, ..., x_{784}^S) \\
&\vdots \\
x_i^{S+1} &\sim p(x_i|x_1^{S+1}, ..., x_{i-1}^{S+1}, x_{i+1}^S, ..., x_{784}^S) \\
&\vdots \\
x_{784}^{S+1} &\sim p(x_{784}|x_1^{S+1}, x_2^{S+1}, ..., x_{783}^{S+1})
\end{aligned}
$$

As shown, Gibbs sampling simplifies the process of sampling in a high dimensional space (here, $N = 784$) by using a uni-dimensional sampling cyclically repeated to eventually sample from the overall dimension space. This complexity simplifying property of the Gibbs samplers is the reason why this particular MCMC technique is favored to training RBMs.

2.2 Quantization

The main approach to power saving in the thesis is through the quantization of the parameters of the RBM models and Gibbs samplers. In order to do so, the number of bits in each model is limited at 5 different levels of quantization: 4-bit, 8-bit, 12-bit, 16-bit and 64-bit. Within each level of quantization, the number of bits before the decimal point m and the number of bits after the decimal point n can also be essential to the quality of the model. To fully utilize the limited number of bits, an iterative process of accuracy computation was done and the most optimal $m.n$ for each bitwidth was determined.

Table 2.1: Quantization structure

	number of bits		
Bitwidth	**sign**	**m**	**n**
4-bit	1	2	1
8-bit	1	3	4
12-bit	1	7	4
16-bit	1	7	8
64-bit	1	55	8

The model is then quantized at the five different bitwidths, according to the $m.n$ structure in Table 2.1. These five levels of quantizations are performed on the model which are used in the experiments throughout the thesis.

Chapter 3

Overview of the Restricted Boltzmann Machine

3.1 What is an RBM?

A Restricted Boltzmann Machine (RBM) and a Deep Belief Network (DBN) are stochastic neural networks that have been used for a variety of discriminative and generative tasks, such as image classification, sequence completion, motion synthesis and speech recognition.

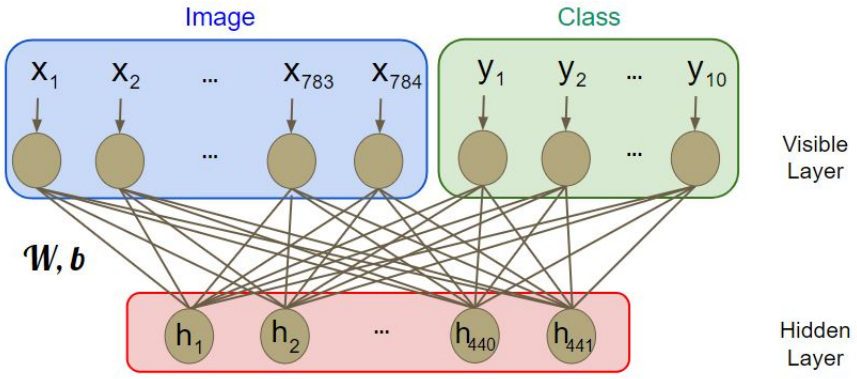

Figure 3.1: Training of an RBM

An RBM can be trained to learn a probability distribution over a set of input data. The structure of an RBM consists of two layers of neuron-like units: a visible

layer and a hidden layer. The neurons are interconnected between the two layers and there are no connection within each single layer, meaning that all visible-to-visible and hidden-to-hidden communications are restricted to be nonexistent, as shown in Figure 3.1. Since the visible and hidden layers are such that no two neurons within the same layer are adjacent in the graph, the resulting graphical structure is a bipartite graph. The bipartite property of the RBM also allows Gibbs sampling to be done in parallel, reducing the computation time.

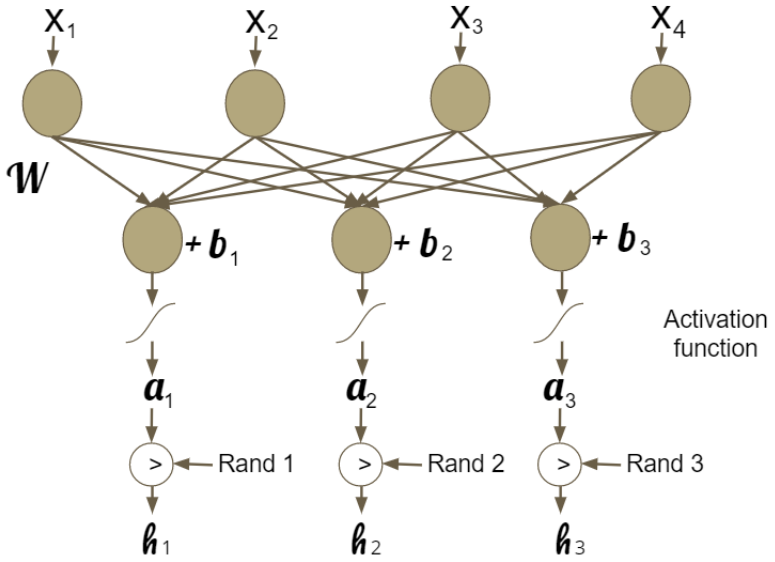

Figure 3.2: Single Layer Neural Network

In Figure 3.2, we have 4 units of visible neurons on the top layer, and 3 units of hidden neurons on the bottom layer. The notation X_i denotes each input feature. Each of the features is fed into a visible unit. Now, each unit of the visible layer represents a single feature of the input data. A hidden layer is then used to model a distribution over the visible units. The value of the units in the hidden layer depends on the inputs into the visible layer and their weights and biases. The matrix W in the figure has a dimension of 4×3, where each element w_{ji} of the matrix represents the weights between the visible neuron i and the hidden neuron j, and b_j is the value of the bias at the hidden neuron. The output of the hidden neuron z_j can be calculated as follows:

$$z_j = \sum_i w_{ji} x_i + b_j$$

The output of the hidden neurons are then fed into an activation function. The notation a_j in Figure 3.2 denotes the output of the activation function. The probability of activation is:

$$a_j = P(z_j = 1|x) = \sigma(\sum_i w_{ji} x_i + b_j)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ in the case of an RBM. Since the RBM is used with Gibbs sampling in this research, the activation function will be the logistic sigmoid function which will be further elaborated in Section 3.4. In the binary RBM, each output of the activation function is then compared to a number between 0 and 1, generated randomly from a uniform distribution. In the hardware implementation, the random number is generated using a pseudo-random number generator (PRNG). If the output of the activation function is greater than the random number, the hidden unit is activated.

3.2 Deep Belief Network (DBN)

The Deep Belief Network (DBN) can be formed by stacking multiple layers of hidden units on an RBM. Similar to an RBM, the neurons are inter-connected between the layers but not intra-connected within a single layer.

Using the output of the RBM in Figure 3.1, we can stack a second hidden layer $z^{(2)}$, and feed the outputs from the first layer $a_1^{(1)}$, $a_2^{(1)}$ and $a_3^{(1)}$ into the second layer of hidden units, as shown in Figure 3.3. The output of the second layer can be computed as below:

$$z_k^{(2)} = \sum_j w_{kj}^{(2)} a_j^{(1)} + b_k^{(2)}$$

The stacking of the hidden layers can be repeated in this manner in order to form a deeper neural network.

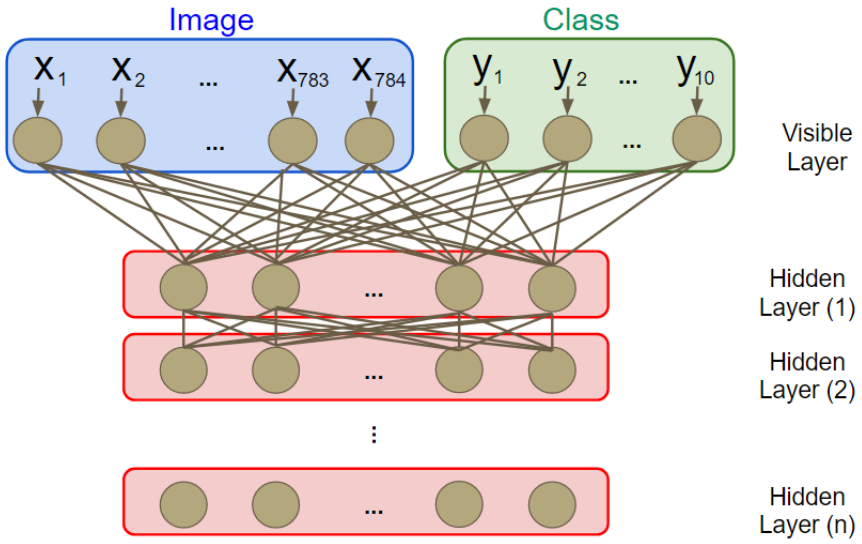

Figure 3.3: Stacking of hidden layers to form DBN

In this thesis, all the effects of each quantized sigmoid function approximation are demonstrated on an RBM model for simplicity during evaluation and comparison. However, similar trends in behavior are expected to be observed when the effects of the quantized sigmoid function approximations are applied to a DBN model.

3.3 Training a Boltzmann Machine

The training of a Boltzmann Machine is done using Gibbs sampling in MCMC. Each neuron is sampled based on the total input from other connected neurons with a sigmoidal activation function.

The Boltzmann Machine is trained usually by using the gradient maximization of the log-likelihood given the training data. The likelihood function of the Boltzmann Machine is:

$$L(\Theta; v, h) = P(v, h; \Theta) = \frac{e^{-E(v,h;\Theta)}}{Z(\Theta)}$$

where $E(v, h; \Theta)$ is the energy function of the RBM, and parameters $\Theta = b, c, W, U, V$, where b and c are the biases of the hidden and visible units respectively, and W,

U and V are the weight matrices of the visible-to-hidden connections, visible-to-visible connections, and hidden-to-hidden connections respectively. The energy function of a general Boltzmann Machine is:

$$E(v, h; \Theta) = -b'v - c'h - h'Wv - v'Uv - h'Vh$$

An RBM feeds the input data into the visible layer, and connects the visible neurons to the hidden neurons, as shown in Figure 3.1. To train an RBM means to learn the weights of each connection W and the bias of the neurons b. In an RBM, the inter-layer connections are restricted to be nonexistent, and therefore the last two terms of the energy function will be zero:

$$E(v, h; \Theta) = -b'v - c'h - h'Wv$$

In order to learn the parameters W, b, and c, we attempt to obtain the maximum log-likelihood estimate of the parameters Θ, which requires computation of the gradient of the log-likelihood function:

$$\frac{\partial\ logP(v, h; \Theta)}{\partial \Theta} = -\sum_{h} P(h|v; \Theta) \frac{\partial\, E(v, h; \Theta)}{\partial \Theta} + \sum_{\widetilde{v}, \widetilde{h}} P(\widetilde{v}, \widetilde{h}; \Theta) \frac{\partial\, E(\widetilde{v}, \widetilde{h}; \Theta)}{\partial \Theta}$$

where $\Theta = b, c, W$.

According to the Contrastive Divergence (CD) method proposed in [7], maximizing the log-likelihood is equivalent to minimizing the Kullback-Leibler (KL) divergence between the data distribution P^0 and equilibrium distribution of the model over visible variables P_Θ^{final}. By replacing the term

$$\frac{\partial\ logP(v)}{\partial \Theta} \text{ with } \frac{\partial\ (\mathbb{KL}(P^0 || P_\Theta^{final)}) - \mathbb{KL}(P_\Theta^1 || P_\Theta^{final}))}{\partial \Theta}$$

where P_Θ^1 is the distribution over the reconstructions of the data vectors after one step of Gibbs sampling. The maximization log-likelihood w.r.t the weight w_{ji} can be rewritten as:

$$-\frac{\partial\ (\mathbb{KL}(P^0 || P_\Theta^{final}) - \mathbb{KL}(P_\Theta^1 || P_\Theta^{final}))}{\partial w_{ji}} = < h_j v_i >_{P^0} - < h_j v_i >_{P_\Theta^1} + residual$$

where $< h_j v_i >_{P^0}$ is the positive (wake) phase with the visible unit clamped to the input data, and $< h_j v_i >_{P_\Theta^1}$ is the negative (sleep) phase where both hidden

and visible neurons are running freely to attempt to reproduce the positive phase. The residual term is problematic to compute, and is shown in [7] that its effect is small and can be ignored.

$$-\frac{\partial\ (\mathbb{KL}(P^0||P_\Theta^{final}) - \mathbb{KL}(P_\Theta^1||P_\Theta^{final}))}{\partial w_{ji}} = < h_j v_i >_{P^0} - < h_j v_i >_{P_\Theta^1}$$

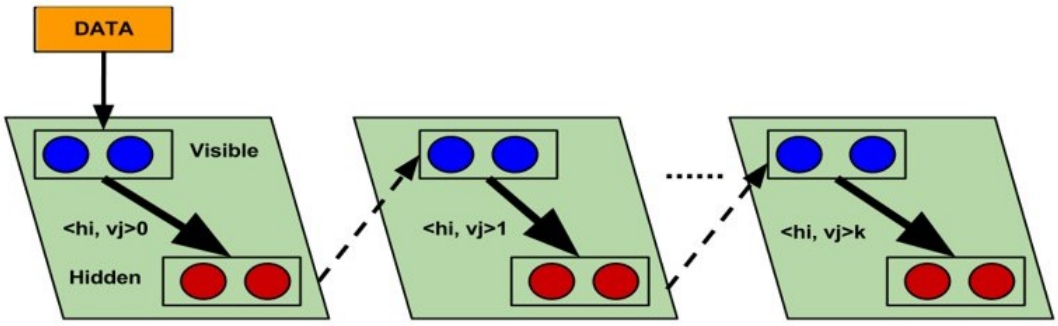

Figure 3.4: Visible-hidden-reconstruction over MCMC steps

The visible-hidden-reconstruction can iterate over multiple MCMC steps for the purpose of training and inference as shown in Figure 3.4. Using CD, we can force the weight update to be calculated with a specified number of steps, instead of waiting infinitely for convergence. For training of a Discriminative RBM, it is found that performing one MCMC step is sufficient to train the model. As a result, CD-1 is used for the training of the Discriminative RBM:

$$CD{-}1: \quad < h_j v_i >_{P^0} - < h_j v_i >_{P_\Theta^1}$$

The generative model, on the other hand, requires prolonged Gibbs sampling to be trained. CD-k is used for the training of the Generative RBM:

$$CD{-}k: \quad < h_j v_i >_{P^0} - < h_j v_i >_{P_\Theta^k}$$

The change in weight is defined as:

$$\Delta w_{ji} = \lambda(< h_j v_i >_{P^0} - < h_j v_i >_{P_\Theta^k})$$

where λ is the pre-determined learning rate. The weight can then be updated as:

$$w_{ji}^{(\tau+1)} = w_{ji}^{(\tau)} + \Delta w_{ji}$$

Similarly, the other parameters in Θ can be updated using the same method.

The training of a DBN can be done greedily. It can be trained layer-wise, similar to training a RBM. Using the same stochastic sampling procedure, the values of each layer of neurons in the DBN can be inferred [7].

3.3.1 Supervised Training

The first 5000 images and labels from the MNIST dataset are used to train the Discriminative RBM model. Since RBM is a bipartite graph, we notice that for any visible image neuron to communicate with any visible class neuron, it will have to go through a hidden neuron. Therefore, the position of the neurons in Figure 3.1 can be adjusted without changing any connections to show the training process described in Figure 3.5. Each image in the training dataset and its corresponding label are fed into the RBM model, forming the visible layers. The first 784 visible image units represents each pixel of the MNIST image and 10 visible class units represents the class labels of the image. These visible class unit corresponding to the class of that image will be 1, and the other 9 units will be zero. From the figure, we can tell that there are 5 parameters here that we have to learn: W, b, c, W_c, b_c.

The RBM undergoes an iterative process through all the 5000 trained images to adjust these weights and biases according to $CD-1$.

3.3.2 Unsupervised Training

The first 5000 images from MNIST dataset are used to train the Generative RBM model, as shown in Figure 3.6. Note that no class label is used, so the training is unsupervised in this case. Each image in the training dataset is fed into the RBM model, forming the visible layer. The first 784 visible image units represents each pixel of the MNIST image, and is fed into the 441 hidden units. The hidden units then fed back to the visible units. The output from the visible unit then tries to reproduce the original input, and the weights and biases are updated each step. From 3.6, we can tell that there are 3 parameters here that we have to learn at each MCMC step: $W^{(t)}, b^{(t)}, c^{(t)}$.

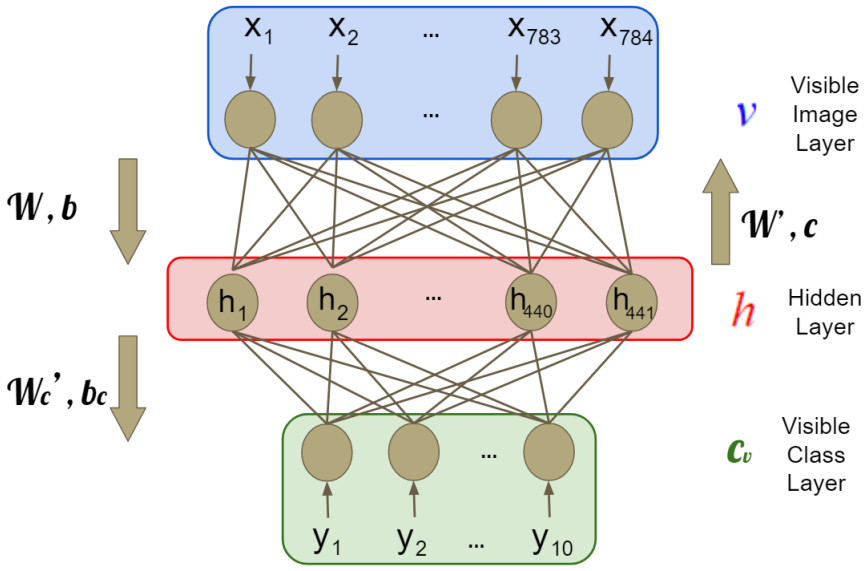

Figure 3.5: Supervised Training of an RBM

The RBM undergoes an iterative process through all the 5000 trained images to adjust these weights and biases according to $CD-k$, where t ranges from 1 to k=50.

3.4 Gibbs Sampling for the RBM

The Gibbs sampling MCMC method uses a state transition probability, often referred to as a kernel. In a binary Restricted Boltzmann Machine (RBM), the Gibbs sampling kernel is the logistic sigmoid function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

and the probability of activation of unit j is

$$p(x_j = 1|x_i) = \sigma(\sum_j w_{ji}x_i + b_j)$$

where w_{ji} is the weight from unit i to unit j and b_j is the bias at unit j.

In order to perform hardware implementation, we cannot use an ideal sigmoid function as our activation function, because it is impossible to implement the function $\sigma(\cdot)$ to infinite precision (in finite time) on the finite-precision hardware

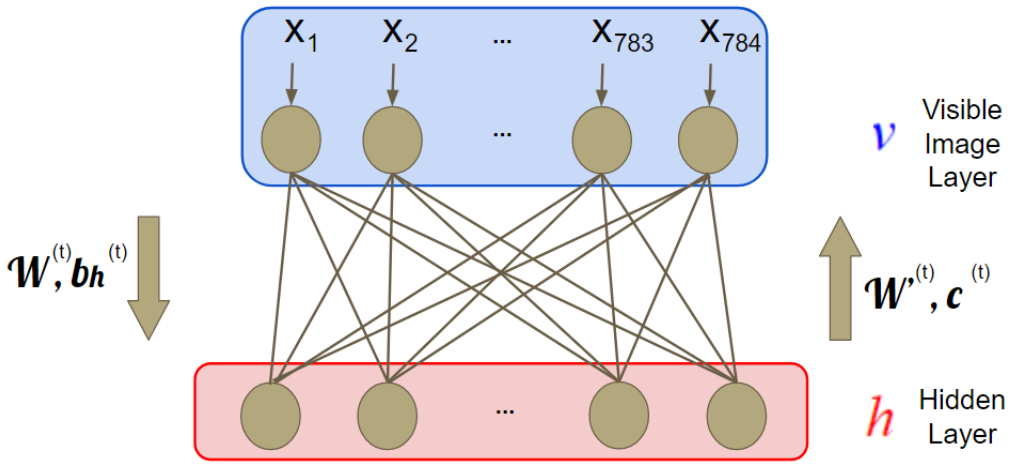

Figure 3.6: Unsupervised Training of an RBM

we are interested in. As a result, we have to approximate on the sigmoid function. In this thesis, we explore two types of approximations for the sigmoid activation function - a "spatial" approximation and a "temporal" approximation.

3.4.1 Spatial Approximation to the Activation Function

The spatial approximation is a piece-wise approximation of a sigmoid function over its input domain. As shown in Figure 3.7, the ideal sigmoid function in blue can be broken down into segments, and a linear approximation of each segment can then be made. The green and red plots are examples of the spatial approximations made to approximate the ideal sigmoid function. From the figure, it can be observed that the green plot is a better approximation of the sigmoid function than the red plot, indicating that the higher the order of approximation of the sigmoid, the closer the approximate would be to the ideal sigmoid function.

In this research, a three-segment (in red) and a five-segment (in green) approximations of the sigmoid function are used. The three-segment spatial approximation is the first-order (simplest) spatial piece-wise approximation of the sigmoid function. It is the simplest to implement since the sigmoid function is only broken down into the minimal number of segments. This minimal estimation of the sigmoid function is not only simple to implement, but also proven to be sufficient to

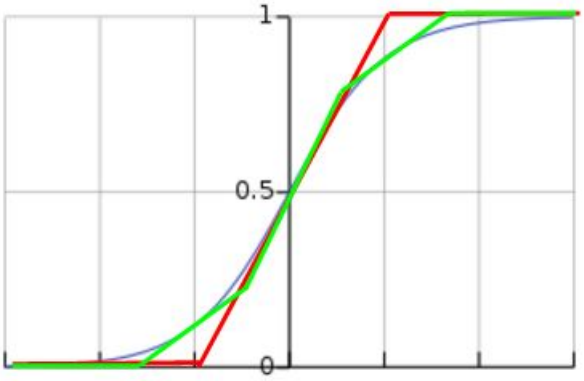

Figure 3.7: Spatial sigmoid approximation

be used in the discriminative model [9].

The activation output from the spatial approximation is then compared to a value generated by an on-chip pseudo random number generator (PRNG) to determine the spiking of the output neurons. If the value from the PRNG is larger than the value from the output of the spatial approximation, the output of the neuron is spiked to 1.

3.4.2 Temporal Approximation to the Activation Function

The temporal approximation is a piece-wise approximation of a sigmoid function in the temporal domain, originally proposed in [10]. It uses 4 parameters: 1) Tw, the number of discrete time steps, 2) Vt, a fixed threshold value, 3) TM, the number of bits allowed for a stochastic threshold variable, and 4) the stochastic *leak*.

The algorithm for realizing the temporal sigmoidal sampling rule to perform MCMC sampling in RBMs is [10]:

Input: $v_i = scale * (\sum_j w_{ij}x_j + b_i), V = V_{initial} = v_i$
repeat
  $\mathbf{V = V + leak * (B(0.5))}$, **B is Bernoulli**
  $\mathbf{Vt\_rand = Vt + floor(U(0, 2^{TM} - 1))}$
  $\mathbf{spiked(V \geq Vt\_rand) = 1}$
until Tw steps;

In the temporal approximation implementation, Vt_rand represents the threshold and V represents the membrane potential. The spikes are determined by comparing the value of V and Vt_rand. If V is larger than Vt_rand, the neuron is spiked. If a spike occurs in any of the preset number of steps Tw, the value of the neuron is set to 1. This method to approximate the sigmoid function uses the stochastic properties and spike synchronization at a fixed time steps to reproduce a sigmoidal probability curve. The values of the random threshold Vt_rand and $leak$ are realized using the on-chip PRNG.

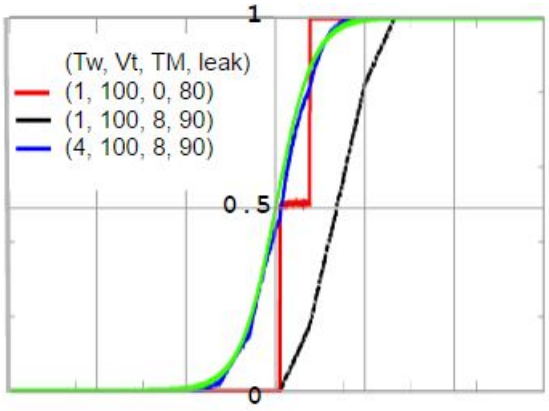

Figure 3.8: Temporal sigmoid approximation

These four parameters are non-linear and therefore, the approximation of the sigmoid function cannot be done by varying a single parameter alone. Figure 3.8 is an attempt to compare the different parameter tuning to implement the temporal approximation of the ideal sigmoid function (in green).

Tuning the Temporal Approximation

In this thesis, we focus on two ways to vary the parameters for the temporal approximation: varying the bitwidth and changing the number of discrete time steps Tw. The values of the other parameters are tuned accordingly to attempt to approximate the shape of an ideal sigmoid function. This method is used in order to match the quantization structure specified in Table 2.1. To do so, the scaling factor is set so that the input values are scaled by the number of bits after

the decimal point, denoted by n in Table 2.1. The total number of bits are also limited to $m + n + 1$. The value of Tw are set to be 1 and 16 to demonstrate the effect of increasing the number of time steps. For each of the Tw and scaling factor combination, the rest of the parameters are tuned.

The sets of parameters used for the first order ($Tw = 1$) temporal approximations are determined and shown in Table 3.1.

Table 3.1: Parameters for Temporal Approximation for $Tw = 1$

	Parameters (Tw = 1)	
Bitwidth	**Scaling Factor**	**(Vt, TM, leak)**
4-bit	2	-4, 3, 0
8-bit	16	-65, 7, 0
12-bit	16	-65, 7, 0
16-bit	256	-510, 10, 0
64-bit	256	-510, 10, 0

To find the parameters for other values of Tw, the parameters of the temporal approximation are systematically adjusted. For example, using the 64-bit model and $Tw = 6$, the tuning of the parameters are shown step by step in Figure 3.9. First, set Tw to 6, and the other three parameters to zero. Then adjust TM such that the curve of the upper corner becomes a good estimate of the ideal sigmoid function. The value of *leak* can then be changed so that the shape of the approximated function follows the shape of the ideal function. Lastly, adjust the value of Vt to center the approximated function to the origin. The tuned temporal approximation (in green) is a good estimation of the ideal sigmoid function (in red dashes).

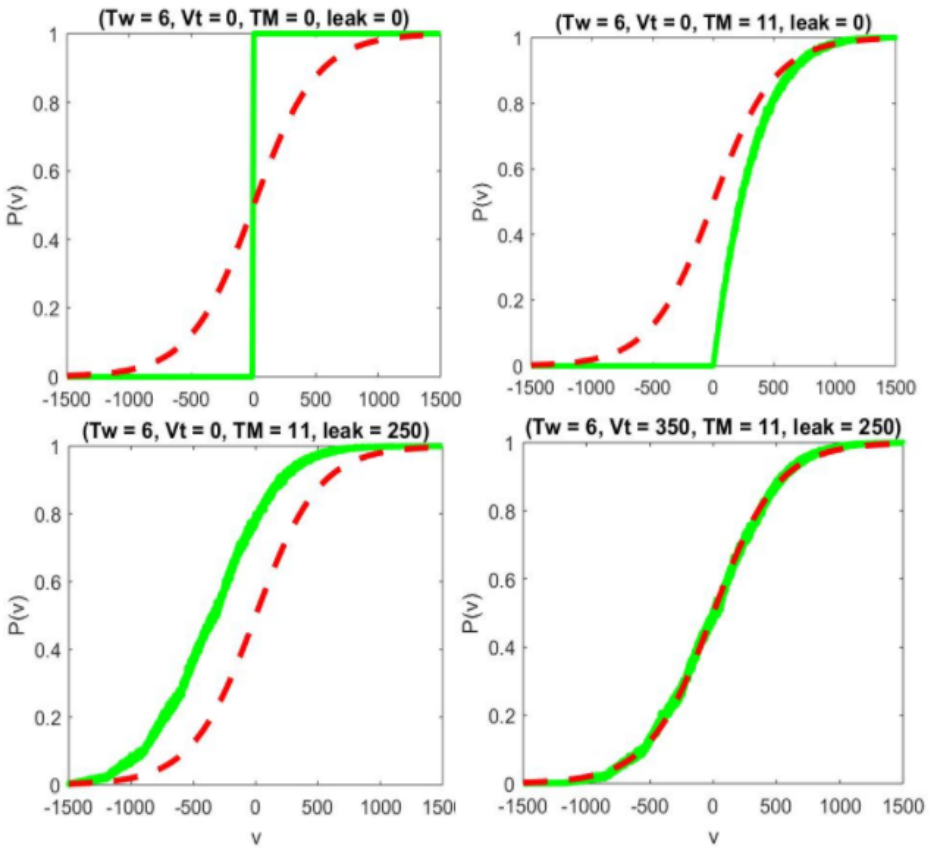

Figure 3.9: Tuning the Temporal Approximation of the Sigmoid function

Chapter 4

Discriminative RBM Model with Finite Precision (FP)

In this thesis, the MNIST dataset is used as the proof-of-concept test data. The Discriminative RBM model was trained using 5000 images and their corresponding labels as described in Section 3.3.1. The ideal values used for hidden layer weights, W, and biases, b, and the classification layer weights, W_c, and biases, b_c, are the optimized values for the weights and biases obtained from the training process which is assumed to have occurred on the cloud. The ideal values can then be systematically adjusted to finite-precision values for local hardware implementation.

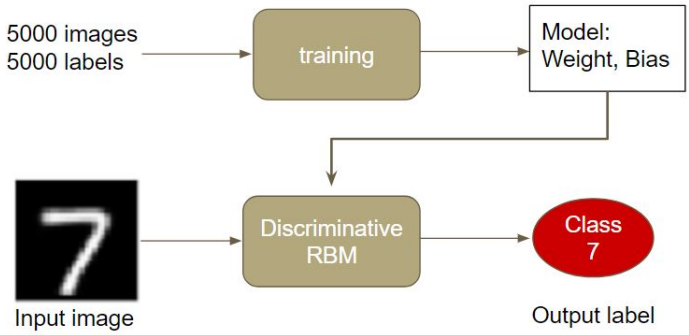

Figure 4.1: Discriminative model flow chart

The trained model is then used to classify the 1000 test images. Each image

is 28×28, giving it 784 distinct pixels. The input image is quantized to $m.n$ bits (following the infrastructure shown in Table 2.1), and fed into the visible layer, where each of the 784 visible units corresponds to a single pixel in a MNIST image. To get the value of each of the 441 hidden units j, each visible unit i is multiplied by the weight w_{ji}, and the bias b_j is then added to the sum of these products over all the visible units. Here, the values of the weights and biases were also quantized to $m.n$ bits.

The hidden unit is then fed into the activation function, which is simulated by an approximation of the sigmoid activation function for hardware implementation and the output from the approximation is compared to a random number generated from the PRNG. If the output of the sigmoid approximation is greater than the random number, the hidden unit is activated. The binary output from the 441 hidden units are then fed into the classification layer which consists of 10 neurons. Similarly, the output is fed into the approximate sigmoid function and compared to a random number generated by the PRNG to determine the activation. If the neuron is activated, it is spiked. The process is repeated over 1000 iterations and the argument of the most spiked classification neuron corresponds to the output classification label for the input image. The entire process to perform discrimination is summarized in the flow chart in Figure 4.1.

As mentioned in Section 3.4, two different kernels (spatial and temporal) were used to approximate the sigmoidal activation function. Classification using the Discriminative model was performed using both types of approximations and their respective results are shown and compared in terms of efficiency.

4.1 Efficiency of Discriminative RBM with FP

The term efficiency here is defined loosely, and can be modified depending on the preference of the designer. For the purpose of this paper, efficiency e for the discriminative case is defined to be:

$$e = \frac{(Acc - Acc_{req})}{P_{consumption}}$$

where Acc is the accuracy value of the model in percentage, Acc_{req} is the minimum accuracy requirement for the system, and $P_{consumption}$ is the total power consumed by the model. In this demonstration, Acc_{req} is set to 90%, meaning that we expect our model to be at least 90% accurate when performing classification. Any model that is less than Acc_{req}% accurate will result in a negative efficiency and would be rejected by the hardware design.

4.1.1 Accuracy of FP Discriminative RBM

The accuracy of the Discriminative RBM is computed using the percentage of correct predictions.

$$Acc = \frac{\text{\# of correct classifications}}{\text{total \# of classifications}} \times 100\%$$

The argument of the most spiked neuron in the classification layer after 1000 iterations is compared to the ground truth, which is the label of the test data. If the position of the most spiked neuron matches the test image label, the classification is accurate. The process is repeated over the 1000 test images and the average accuracy can be obtained.

4.1.2 Power Consumption of FP Discriminative RBM

A hardware implementation of the Discriminative RBM is implemented in Verilog and the block diagram of the Verilog code is shown in Figure 4.2. The power consumption of the Discriminative RBM is measured using this Verilog implementation.

In the Verilog design, there is a main block which consists of two big blocks. The first block (in light blue) simulates the flow from the visible layer to the hidden layer, and the second block (in light green) simulates the flow from the hidden layer to the class layer. Within each of the two blocks, there is an accumulator, a sigmoid function, a Random Number Generator (PRNG) and a comparator. The entire design is quantized to j bits, where $j = 4, 8, 12, 16, 64$. The PRNG is set to $k = 16$ bits to allow a reasonable amount of variation in the numbers generated.

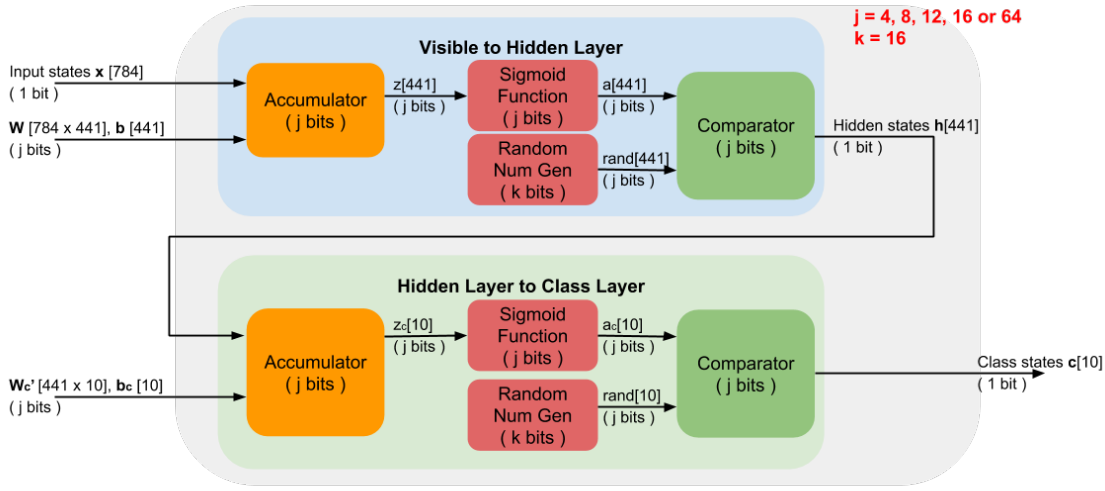

Figure 4.2: Block Diagram for Verilog implementation of Discriminative RBM

The quantization of the model is set to follow the $m.n$ infrastructure described in Table 2.1.

The input states x represents the 784 pixels in an input image, and the quantized j-bit weight W and bias b are fed into the accumulator. The accumulator computes the sum of the product of the weights and the corresponding input states, and adds the sum to the respective biases of the hidden units. The output z_h is then fed into the approximated sigmoid function. The output of the sigmoid function a_h is compared to the output of the PRNG. The PRNG generates a random number of a fixed bitwidth of $k = 16$ bits and the number has to be zero-padded or truncated to j bits in order to be compared to the output of the sigmoid function. If $a_h > rand$, then the hidden state is spiked to 1. Otherwise, the hidden state is 0.

The 441 hidden states, the quantized j-bit weights from the hidden neurons to the class neurons W_c', and bias of the class layer b_c are fed into the accumulator in the second (light green) block. The process is similar to that of the top (light blue) block and the class states are spiked accordingly. The position of the spiked class state is the class which the model classified the input image to be in.

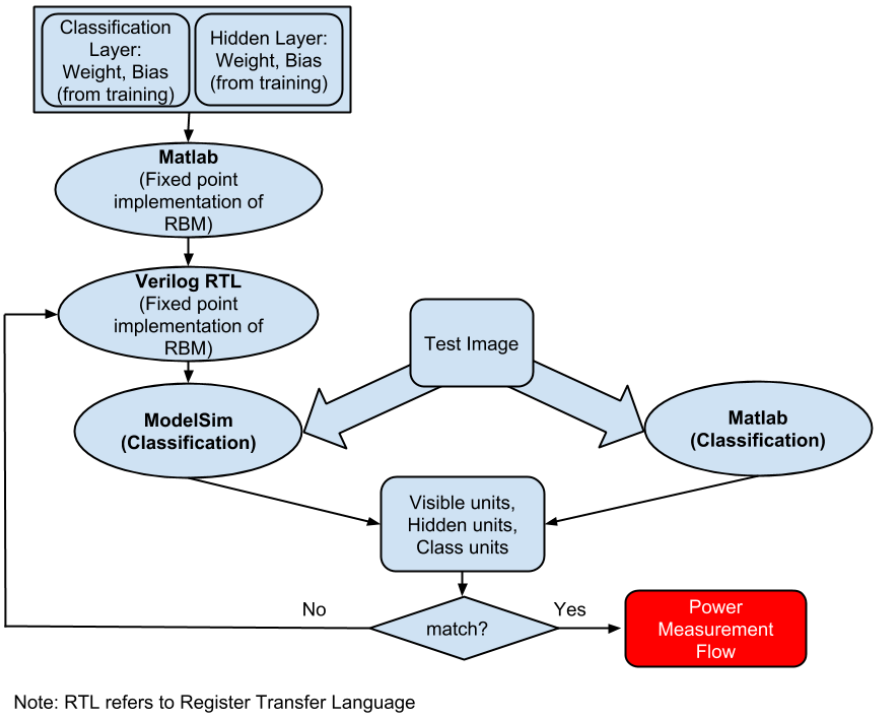

Figure 4.3: Procedure for Verilog implementation and verification of Discriminative RBM

The neurons, intermediate states and computations are programmed using registers, wires, and digital logics in Verilog, following the procedure described in Figure 4.3. The values of the visible, hidden and class units from the ModelSim are compared with the values from Matlab simulation to ensure consistency between Verilog and Matlab simulations. If the values matches, we proceed to measure the power consumed by the model. Power in the hardware are consumed through dynamic activities such as the switching activities of the registers in the digital logic and through static events such as the current leakage in the transistors. Thus, changing the number of bits of the model, would significantly affect the power consumed by it. The power measured in the Verilog simulation will then be an essential component to determine the efficiency of the model. The procedure used for power measurement is shown in Figure 4.4.

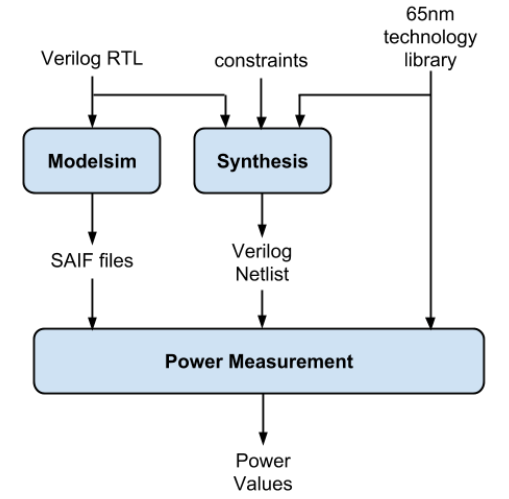

Figure 4.4: Power Measurement Flow

4.2 Results on the Discriminative RBM

To compare the results of the experiment in this thesis, the efficiency of the Discriminative RBM using the two different methods of sigmoid approximations (spatial and temporal) are obtained across the 5 quantization levels in Table 2.1. The efficiency is dependent on the classification accuracy and the power consumption of the model.

4.2.1 Accuracy Results

For each bitwidth of the model, the values of the weights and biases are quantized according to the $m.n$ structure in Table 2.1, and the accuracy of the model using the spatial approximation and the temporal approximation are shown in the second and third columns in Table 4.1 respectively. In the table, the notation **DS** represents using the *Discriminative RBM model with spatial approximation* (3-segment), and notation **DT** refers to using the *Discriminative RBM model with temporal approximation*($Tw = 1$). The accuracy is also plotted in Figure 4.6.

From Table 4.1, the accuracy of the model increases in general, as the number of bitwidth increases. In the spatial approximation example, the accuracy is low

Table 4.1: Accuracy Results

	Classification Accuracy(%)	
Bitwidth	**DS**	**DT**
4-bit	86.4	92.2
8-bit	94.2	94.1
12-bit	93.8	94.0
16-bit	94.4	94.3
64-bit	94	94.8

(about 86.4% on average) at 4 bits. As the number of bits increase to 8, the accuracy increased to about 94.2% on average. However, the accuracy saturates at around 8 bits. As we can see from table 4.1, the accuracy is not significantly changing after it saturates at 8 bits. The slight fluctuations in accuracy rate can be due to the uncertainty caused by the random number generated from the PRNG, slightly affecting the spiking of the neurons.

In the case of a temporal approximation example, the accuracy is relatively high (about 92.2% on average) even at 4 bits. As the number of bits increase to 8, the accuracy increased to about 94.1% on average. However, the accuracy also saturates at around 8 bits. According to the table 4.1, the increase in accuracy is, again, not significant as the number of bits continue to increase.

By comparing the accuracy performance between the the Discriminative RBM model using the two different approximations, the performance of the temporal approximation is better than the spatial approximation across bit widths on average. The difference in accuracy is negligible for models that are 8-bit and above, but is significant at 4-bit.

4.2.2 Power Consumption Results

The power consumption of the models were first reduced through efficient clock gating methods, by activating the blocks only when necessary. As a result, for each simulation, the RBM layer (connecting 784 visible units to 441 hidden units) consumes more than 90% of the power on average across all the bitwidth, while

the Classification layer (connecting 441 hidden units to 10 class units) consumes only less than 10% of the power on average across all the bit width. An example of the power breakdown is shown in Figure 4.5, demonstrating the efficient clock gating methods which leads to the reduction in switching activities [12].

Dynamic Power Units = 1mW

Hierarchy	Total Power	%
Main	7.469	100.0
classi (ClassiLayer)	0.568	7.6
adder_only (ap_adder_1)	3.04e-02	0.4
add_13 (ap_adder_1_DW01_add_0)	2.32e-02	0.3
sg (sigmoid_1)	0.426	5.7
add_54 (sigmoid_1_DW01_inc_1)	1.66e-03	0.0
add_68 (sigmoid_1_DW01_add_2)	2.40e-03	0.0
rnd (RandomGenerator_1)	0.116	1.6
rbm (RBMLayer)	6.847	91.7
adder_only (ap_adder_0)	0.206	2.8
add_13 (ap_adder_0_DW01_add_0)	0.161	2.2
sg (sigmoid_0)	7.13e-02	1.0
add_54 (sigmoid_0_DW01_inc_1)	1.66e-03	0.0
add_68 (sigmoid_0_DW01_add_2)	4.87e-03	0.1
rnd (RandomGenerator_0)	1.33e-02	0.2

Figure 4.5: Power Breakdown for Discriminative RBM

The power consumption for the discriminative models are then measured at different bitwidths, using the spatial approximation and the temporal approximation. The results are plotted and compared in Figure 4.6.

The power consumption trends shown in Figure 4.6 are expected. As the number of bits used in the model is reduced, the area of the design is scaled down and the number of switching activities is also cut down due to the fewer number of registers used across the entire model. As a result, the power consumption is significantly lowered as the bitwidth of the model is reduced [12].

4.2.3 Efficiency

With the results from the sections 4.2.1 and 4.2.2, the efficiency of both the spatial approximation and the temporal approximation used on the Discriminative RBM are computed. Figure 4.6 shows the combined total power versus accuracy plots for the Discriminative RBM model using the spatial approximation and the

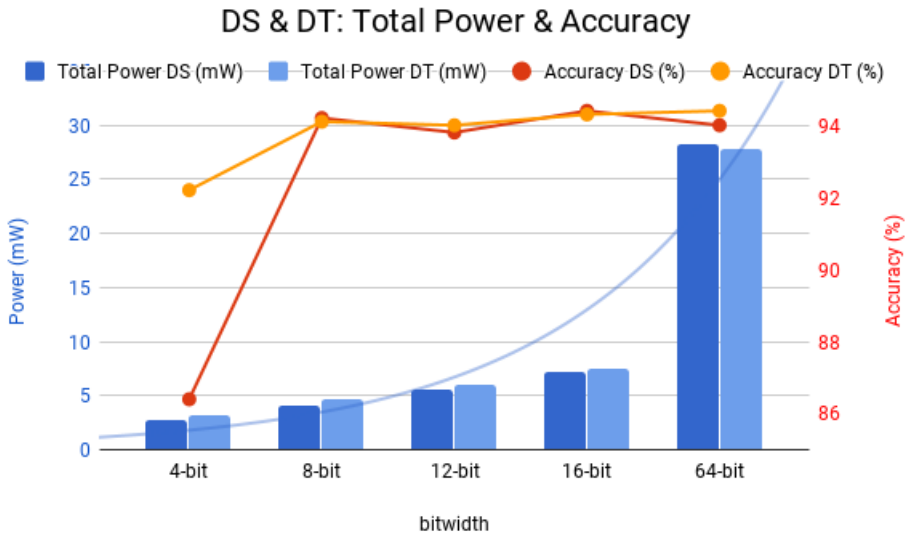

Figure 4.6: Accuracy (right axis) and Power Consumption (left axis) using Spatial vs Temporal Approximations

temporal approximation respectively. In general, as the number of bits increases, the power consumed by the model increases significantly, while the increase in accuracy saturates at around 8 bits. Using our efficiency definition, the efficiency across all bitwidths for both approximations are computed and plotted in Figure 4.7.

Figure 4.7 shows that the efficiency of both spatial and temporal approximation used on the Discriminative model to be highest at 8-bit quantization. However, the performance of the model using the temporal approximations is significantly better at low bitwidth, specifically at 4-bit quantization. At higher bitwidths, the performances of the two approximations in terms of efficiency are very similar.

4.3 Evaluation

The performance of the spatial and temporal approximations are very similar at higher bitwidths. As a result, the performance for a 8-bit or above model would be identical whichever method of sigmoid approximation the designer chooses to use. However, a designer should choose the 8-bit quantization over the higher

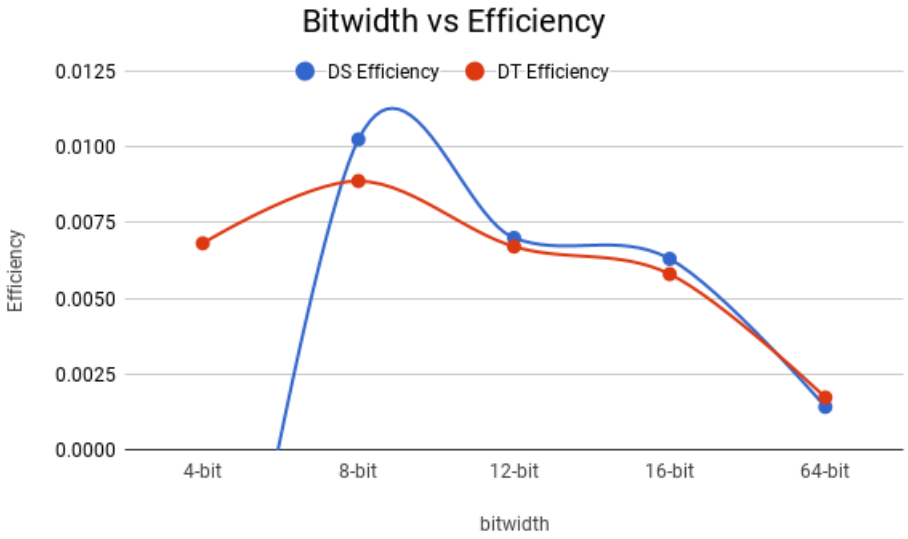

Figure 4.7: Efficiency for Discriminative model

bitwidth models since it performs as well in terms of accuracy, and yet uses the least power.

For specific applications with very low power allowance (below 8-bit implementation), using the temporal approximation can assure a significantly better performing model. This is because the performance in terms of efficiency and accuracy at 4-bit quantization is significantly higher when using the temporal sigmoid approximation.

On a side note, the temporal approximation is easier to tune and can be more flexible to changes to the order of approximation of the sigmoid function. To increase the order of approximation temporally, we are only required to change the number of time steps, and then adjust the other three parameters accordingly; Increasing the order of approximation spatially requires the hard-coding of the new cut-offs for each segment of the sigmoid function.

Chapter 5

Generative RBM Model with Finite Precision

The Generative RBM model was trained using 5000 images, without their labels as described in Section 3.3.2. The visible-to-hidden weights W, hidden layer biases b_h, and the visible layer biases c are optimized from the iterative training process over multiple Gibbs steps.

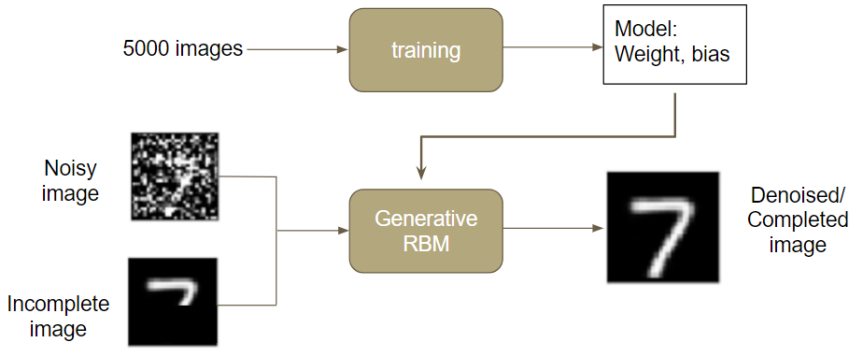

Figure 5.1: Generative model flow chart

The trained model can then be used to perform different types of tasks, such as denoising a noisy image, and completing an incomplete image. The entire process to perform discrimination is summarized in the flow chart in Figure 5.1. In the thesis, the generative model is used to complete 1000 incomplete images.

The incomplete images are quantized to $m.n$ bits (following the procedure

shown in Table 2.1), and individually fed into the visible layer, where each of the 784 visible units corresponds to a pixel in an incomplete image. The learned weights and biases were also quantized to $m.n$ bits. The value of each of the 441 hidden unit j were computed by summing the multiplication of each visible unit i with the weight w_{ji} over all the visible units, and then adding the bias of the hidden units b_j.

The hidden unit is fed into an approximation of the sigmoid activation function for hardware implementation and the output from the approximation is compared to a random number generated from the PRNG. If the output of the sigmoid approximation is greater than the random number, the hidden unit is activated. The binary output from the 441 hidden units are then fed back to the visible layer, and then into the approximate sigmoid function and compared to a random number generated by the PRNG to determine the activation. The process is repeated over a 1000 generation steps. The states of the visible neurons can be reconstructed into a 28x28 image, and with each generation step, the model would attempt to complete the image.

The results of the experiments on the Generative RBM is compared through the efficiency of the RBM using the two different methods of sigmoid approximations across the 5 quantization levels. The efficiency is dependent on the MMD values explained in Section 5.1.1 and the power consumption of the model shown in Section 5.1.2.

5.1 Efficiency of Generative RBM with FP

In this thesis, efficiency e for the generative model is defined to be:

$$e = \frac{(MMD_{req} - MMD)}{P_{consumption}}$$

where MMD is the maximum mean discrepancy between the MNIST dataset, described below, and the output data, and MMD_{req} is the maximum MMD of the system for the generated image to be just recognizable by a human expert. $P_{consumption}$ is the total power consumed by the generative model.

Note that the definition of efficiency can be modified depending on the purpose of the design and the preference of the designer.

5.1.1 MMD of FP Generative RBM

The relative accuracy of the generative model is determined by using a non-parametric test, the maximum mean discrepancy (MMD). MMD is a measure of the distance between mean elements μ in a Reproducing Kernel Hilbert Spaces (RKHS)[8]. In this thesis, we used the MMD test because in high-dimensional data spaces, it has proven to have outperform many traditional two-sample hypothesis testing such as the generalized Wald-Wolfowitz test, the generalized Kolmogorov-Smirnov (KS) test, the Hall-Tajvidi test, and the Biau-Gyorf test. MMD between the MNIST data, X, and the generated data, Y, can be computed by taking the square-root of the following equations.

$$\begin{aligned} MMD^2(X,Y) &= \Big(\sup_{||k||<1} \left(E[k(X,X')] - E[k(Y,Y')]\right) \Big)^2 \\ &= ||\mu_p - \mu_q||^2 \\ &= \frac{1}{m(m-1)} \sum_{X,X'} k(X,X') + \frac{1}{n(n-1)} \sum_{Y,Y'} k(Y,Y') - \frac{2}{mn} \sum_{X,Y} k(X,Y) \end{aligned}$$

where $k(X,X') = \exp(\frac{(X-X')^2}{2\sigma_{XX'}})$ and $\sigma_{XX'}$ is the standard deviation between X and X'. p and q are the distributions of X and Y respectively.

One common method to choose a good $\sigma_{XX'}$ is to use the median heuristic. To do that, we first need to compute the Euclidean distances between each pair of points on the original N-dimensional dataset X. The median value of the $\binom{N}{2}$ Eucildean distances will be a fairly suitable value for $\sigma_{XX'}$. In this thesis, we use the first 1000 images in the MNIST dataset, and obtain the Euclidean distances between all the $\binom{1000}{2}$ number of different pairs. The median value of the Euclidean distances was 33.

The value of MMD_{req} is defined to be the maximum MMD value at which a model can just generate a good enough reconstruction of the incomplete image to

be recognizable by a human. The value of MMD_{req} is a heuristic choice, chosen by manually examining quality of generated images for all classes. By trial-and-error, we determine that the MMD_{req} can be set to around MMD = 0.004.

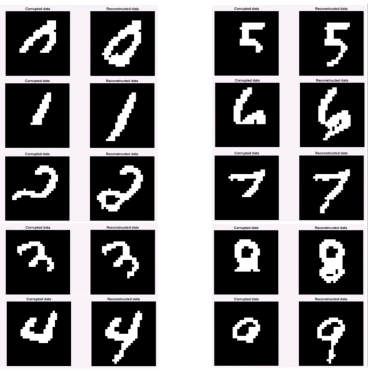

Figure 5.2: Image Outputs (right-hand columns) of Generative Model at MMD = 0.004

At MMD = 0.004, the generative model is able to do generate an output that is just acceptable as shown in Figure 5.2. The first and third column is the input incomplete image and the second and fourth columns are the output image after 1000 generation steps. The output images are not a perfect completion of the input images, but is doing a fair job in the completion process. Therefore, the MMD_{req} is set at 0.004.

5.1.2 Power Consumption of FP Generative RBM

A hardware implementation of the Generative RBM is programmed in Verilog and the block diagram of the Verilog code is shown in Figure 5.3. The power consumption of the Generative RBM is estimated from this Verilog implementation.

Similar to the Verilog block diagram for the discriminative model, there is a main block which consists of two big blocks. The first block (in light blue) simulate the flow from the visible layer to the hidden layer, and the second block (in light green) simulates the flow from the hidden layer to the visible layer. Within each of the two blocks, there is an accumulator, a sigmoid function, a Random Number Generator (PRNG) and a comparator. The entire design is quantized to

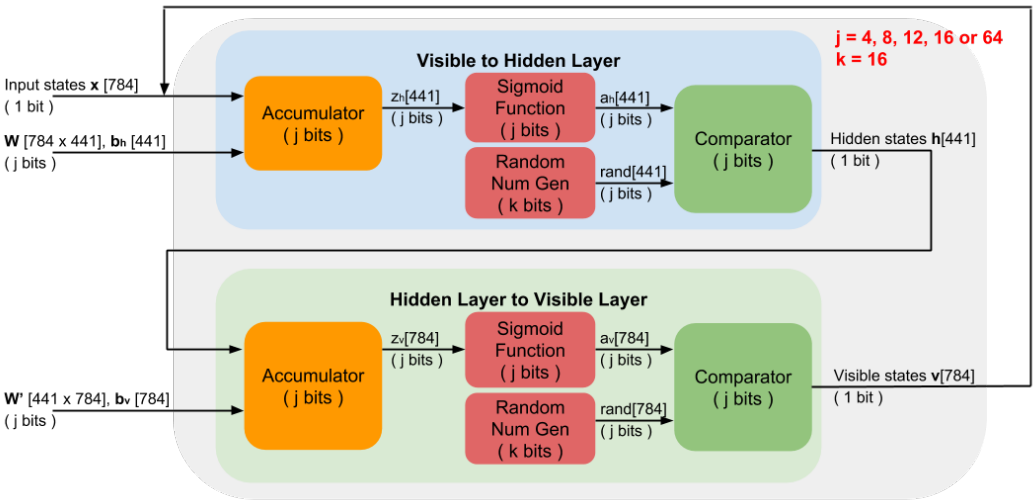

Figure 5.3: Block Diagram for Verilog implementation of Generative RBM

j bits, where $j = 4, 8, 12, 16, 64$, except for the PRNG that is set to k bits. The quantization is set to follow the $m.n$ infrastructure in Table 2.1.

Input states consists of the 784 pixels in an input image, and the quantized j-bit weight W and bias b_h are fed into the accumulator. The accumulator computes the sum of the product of the weights and input states and add the sum to the respective biases. The output z_h is then fed into the approximated sigmoid function. The output of the sigmoid function a_h is then compared to the output of the PRNG. The PRNG generates a random number of a fixed bitwidth of $k = 16$ bits and the number has to be zero-padded or truncated to j bits in order to be compared to the output of the sigmoid function. If $a_h > rand$, then the hidden state is spiked to 1. Otherwise, the hidden state is 0.

The 441 hidden states, the quantized j-bit weights from the hidden neurons to the visible neurons W' and bias of the visible layer b_v are fed into the accumulator in the second (light green) block. The process is similar to that of the top (light blue) block and the visible states are spiked accordingly. The visible states are then fed back to the top block as the new input states into the accumulator. This process is repeated for the desired number of MCMC generation steps.

The Generative RBM implemented in Verilog was verified according to the

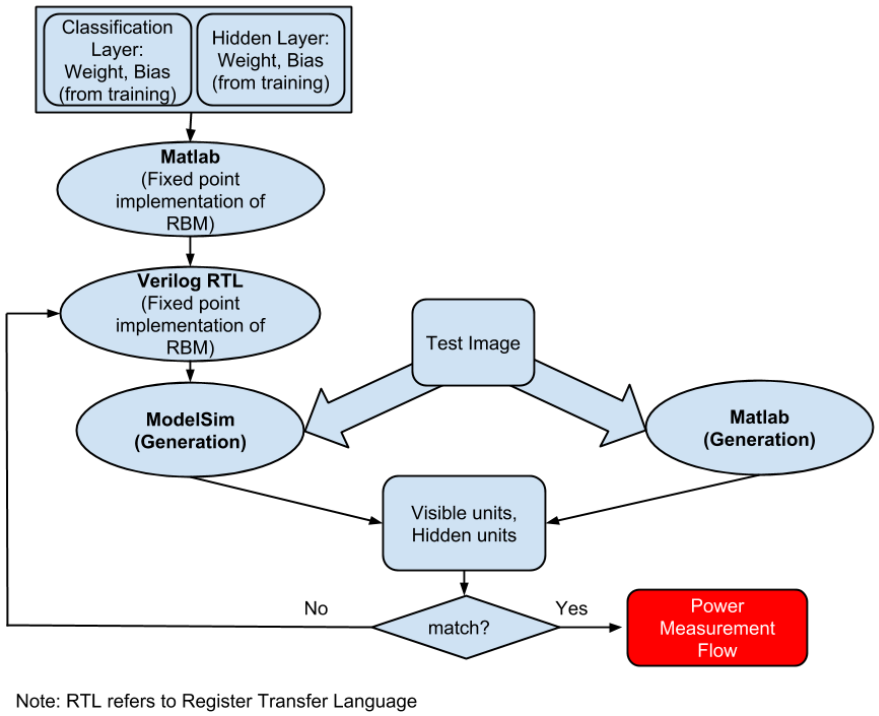

Figure 5.4: Procedure for Verilog implementation and verification of Generative RBM

procedure described in Figure 5.4. The values of the visible and hidden units from the ModelSim are compared with the values from Matlab simulation to ensure consistency between Verilog and Matlab simulations. If the values matches, power consumption of the model is measured as shown in Figure 4.4.

5.2 Sigmoid Approximation for Generative RBM

The generative task is more demanding demanding than the discriminative task. As a result, using the first order approximation of the sigmoid function can be inappropriate. Instead, using a higher order approximation of the sigmoid function can significantly increase performance in a generative model.

For the spatial approximation, we will use the 5-segment approximation of the sigmoid function in our experiment, and compare the results to a 3-segment approximation.

The selection of the temporal approximation is chosen at the optimal number of time steps Tw_0, and will be compared to the first-order temporal approximation with $Tw = 1$.

5.2.1 Selecting the most efficient temporal approximation

To tune the temporal approximation, we start by using the 64-bit Generative RBM model. First, the parameters for the temporal approximation are tuned according to the independent factor $Tw = 1, 2, 4, 6, 8, 10, 12, 14, 16$. The value of Tw is set and the rests of the parameters Vt, TM and $leak$ are tuned to ensure a good approximation of the sigmoid function. The sets of parameters are named from $P1$ to $P9$ as shown in Table 5.1.

Table 5.1: Tuning the temporal approximation using independent Tw values on 64-bit Temporal Approximation

Set	**Tw**	**Vt**	**TM**	**leak**	**MMD**$_{mean}$
P1	1	-510	10	0	0.000832
P2	2	50	10	450	0.000538
P3	4	500	10	430	0.000422
P4	6	350	11	300	0.000380
P5	8	500	11	250	0.000442
P6	10	700	11	240	0.000346
P7	12	870	11	225	0.000417
P8	14	510	12	165	0.000416
P9	16	645	12	160	0.000409

For each parameter set $P1$ to $P9$, the 64-bit Generative RBM is implemented with a temporal approximation of the sigmoid function as the activation function. The mean MMD values and power consumed by each simulation are computed. The mean MMD values are calculated by averaging the MMD values of 100 generated output from each input image. The box plot of the MMD values against parameter sets are shown in Figure 5.5. Efficiency of the model is computed and plotted against the parameter sets as shown in Figure 5.6.

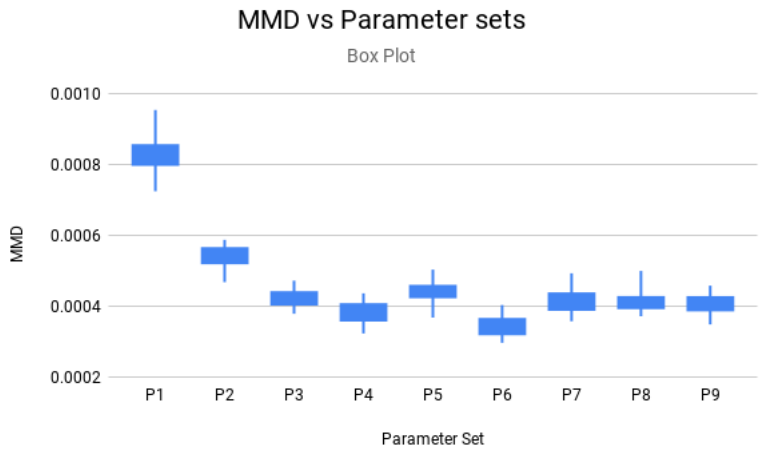

Figure 5.5: Box Plot of the spread of 100 MMD values for each parameter set

From the plot in Figure 5.6, the parameter set $P4$ gives the highest efficiency. The corresponding time step to the most efficient parameter set is $Tw = 6$. Using $Tw = 6$ and the predetermined scaling as shown in Table 5.2, the parameters Vt, TM and $leak$ are tuned to approximate the sigmoid function.

Table 5.2: Parameters for Temporal Approximation when Tw = 6

	Parameters (Tw = 6)	
Bitwidth	**Scaling Factor**	(Vt, TM, leak)
4-bit	2	5, 4, 3
8-bit	16	18, 7, 17
12-bit	16	18, 7, 17
16-bit	256	350, 11, 300
64-bit	256	350, 11, 300

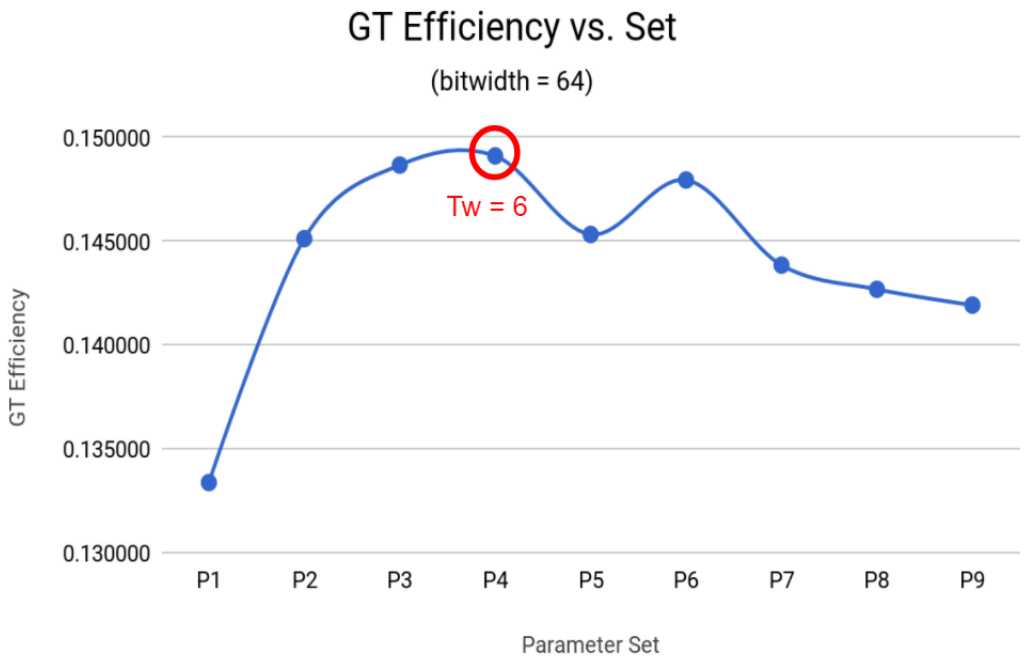

Figure 5.6: Image Output of Generative Model at MMD_{req}

5.3 Results of the Generative Model

5.3.1 MMD Results

For each bitwidth of the model, the values of the weights and biases are quantized according to the $m.n$ structure in Table 2.1, and the MMD values of the model using the first order spatial and temporal approximations of the sigmoid function are shown in the second and fourth columns in Table 5.3 respectively, and the MMD values of the model using the higher order spatial and temporal approximations are shown in the third and fifth columns respectively. The orders of the approximated sigmoid function is described in Section 3.4.1 and 3.4.2, and tuned according to Table 3.1 and 5.2.

MMD vs Bitwidth

In Table 5.3, the notation **GS** represents using the Generative RBM model with spatial approximation, and notation **GT** refers to using the Generative RBM

Table 5.3: MMD Results

	MMD (GS)		MMD (GT)	
Bitwidth	**3-segment**	**5-segment**	**TW = 1**	**TW = 6**
4-bit	0.009356	0.009122	0.009377	0.008278
8-bit	0.006593	0.003864	0.004118	0.003742
12-bit	0.004279	0.003607	0.003777	0.003753
16-bit	0.000844	0.000390	0.000801	0.000377
64-bit	0.000787	0.000400	0.000338	0.000380

model with temporal approximation. The general trends of the MMD of the model for both approximations are similar: the MMD value decreases, as the number of bit width increases.

MMD vs Order of Sigmoid Approximation

From Table 5.3, the MMD values of the 5-segment spatial approximation is evidently lower than the MMD values of the 3-segment spatial approximation. The MMD values of the temporal approximations at $Tw = 6$ is also generally lower than that of the $Tw = 1$ case. This result shows that the generative models are sensitive to the quality of the sigmoid approximation functions.

MMD vs Type of Sigmoid Approximation

The MMD values of temporal approximation is seen to be always lower than the MMD values of the spatial approximation, in both the first order approximation and the higher order approximation. This indicated that the quality of image generated is higher when temporal approximation is used.

5.3.2 Power Consumption Results

The power consumption of the models were again reduced through efficient clock gating methods, by activating the blocks only when necessary. As shown in Figure 5.7, the amount power consumed by the visible to hidden block and by

the hidden to visible block is almost the same. This is power efficient, since the number of connections in both blocks are the same at 784x441.

```
Dynamic Power Units = 1mW
-------------------------------------------------------
                                          Total
Hierarchy                                 Power    %
-------------------------------------------------------
Main                                          7.975 100.0
  rbmHV (H2VLayer)                            3.505  43.9
    add_50 (H2VLayer_DW01_inc_0)           5.54e-04   0.0
    adder_only (ap_adder_1)                   0.120   1.5
      add_13 (ap_adder_1_DW01_add_0)       8.93e-02   1.1
    sg (sigmoid_1)                         6.59e-02   0.8
      add_64 (sigmoid_1_DW01_add_2)        4.14e-03   0.1
      add_65 (sigmoid_1_DW01_inc_0)        2.27e-03   0.0
      rnd (RandomGenerator_1)              1.21e-02   0.2
  rbmVH (V2HLayer)                            3.483  43.7
    add_50 (V2HLayer_DW01_inc_0)           5.58e-04   0.0
    adder_only (ap_adder_0)                   0.104   1.3
      add_13 (ap_adder_0_DW01_add_0)       7.62e-02   1.0
    sg (sigmoid_0)                         4.44e-02   0.6
      add_64 (sigmoid_0_DW01_add_2)        3.71e-03   0.0
      add_65 (sigmoid_0_DW01_inc_0)        2.20e-03   0.0
      rnd (RandomGenerator_0)              7.57e-03   0.1
```

Figure 5.7: Power Breakdown for Generative RBM

The power consumption for the Generative models are then measured at different bitwidths, using the spatial and the temporal approximation respectively. The results are plotted and compared in Figure 5.8.

Power vs Bitwidth

The general trend of the power consumption of the model using all four approximations are similar. As the number of bits increases, the power consumption increases evidently.

Power vs Order of Sigmoid Approximation

The order of sigmoid approximation has no noticeable difference in power consumption in the case of the spatial approximations. However, in the case of the temporal approximations, the higher order approximation $Tw = 6$ consumes more power than the first order $Tw = 1$ approximation, due to the additional time steps.

Power vs Type of Sigmoid Approximation

The power consumed by temporal approximations are in general slightly higher than the power consumed by the spatial approximations, for both the first order and higher order approximations.

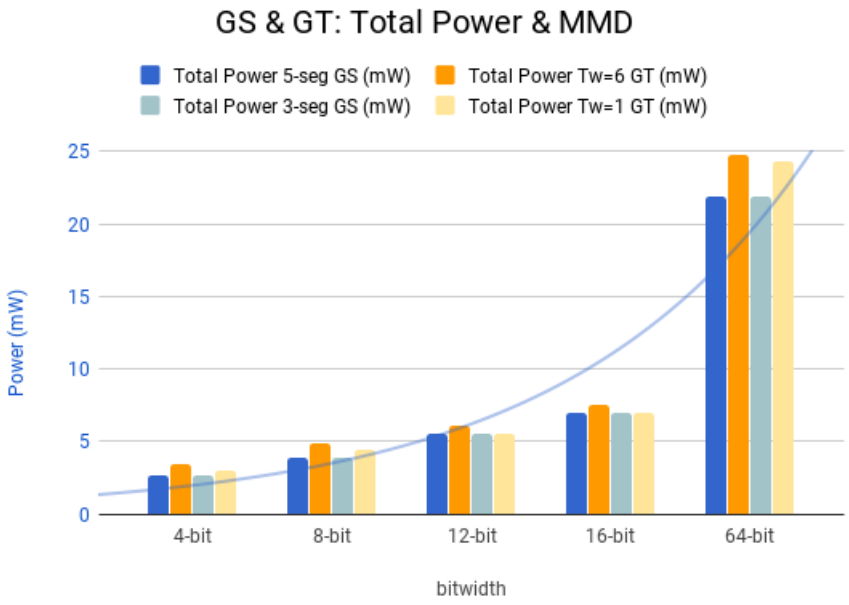

Figure 5.8: Power Consumption for Generative RBM

5.3.3 Efficiency

Using the MMD and power consumption results from the sections 5.3.1 and 5.3.2, the efficiency of the higher order spatial approximation and the temporal approximation used on the Discriminative RBM are computed. Figure 5.9 shows the combined total power versus MMD plots for the Generative RBM model using the first-order spatial approximation and the temporal approximation respectively. In general, as the number of bitwidth increases, the power consumed by the model increases significantly, while the MMD value decreases. Using our efficiency definition, the efficiency across all bitwidths for both approximations are computed and plotted in Figure 5.10.

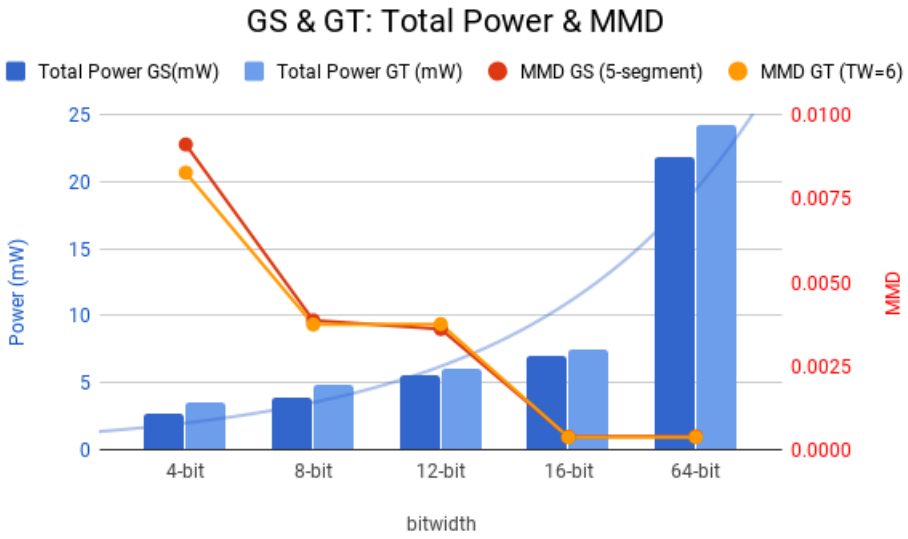

Figure 5.9: Power Consumption & MMD values for Generative model

Figure 5.10 shows that the efficiency of both first and higher order spatial and temporal approximations used on the Generative model to be highest at 16-bit quantization.

For the 3-segment spatial approximation, the efficiency is negative for 4, 8, and 12 bits implementations, and for the $Tw = 1$ temporal approximation, the efficiency is negative for 4 and 8 bits. For both higher-order approximations (5-segment spatial and $Tw = 6$ temporal approximations), the efficiency is negative only for the 4-bit implementation.

Comparing between the 3 and 5 segments spatial approximation, the 5-segment approximation gives a higher efficiency throughout all the bitwidths, especially at low bitwidths. The comparison between the $Tw = 1$ and $Tw = 6$ temporal approximations is similar. The efficiency of the higher-order approximation is higher than the first-order approximations for all bitwidth, and the difference is more significant at low bitwidths.

At low bitwidth, the temporal approximations generally outperform the spatial approximations in terms of efficiency. However, at higher bitwidths, the efficiency performance of both the spatial and temporal approximations are very similar.

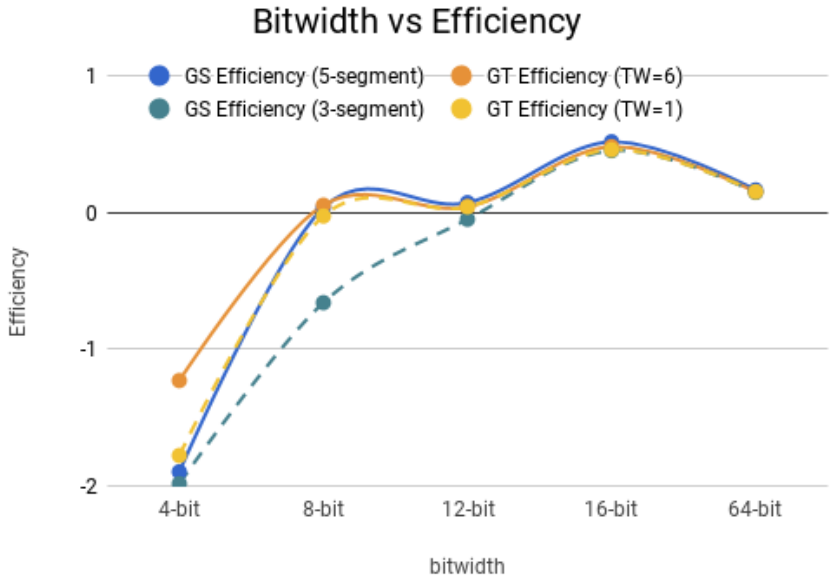

Figure 5.10: Efficiency of Generative RBM

5.4 Evaluation

The performance of the spatial and temporal approximations used in the Generative RBM are very similar at high bitwidths. At 4-bit, none of the approximations are sufficiently powerful to meet the requirement of the 0.004 maximium MMD values to perform a sufficiently decent generation. At high bitwidths, the performance of all the approximations are very similar.

However, if the designer has low power allowance and are able to sacrifice the MMD performance of the model, the higher order temporal approximation will be the best performing choice.

An advantage of using the temporal approximations is that it is easier to tune to alter order of approximation of the sigmoid function by adjusting the parameters. However, using a temporal approximation with high time steps can be detrimental to the level of power consumption by the RBM.

Chapter 6

Conclusions and Future Work

To study the performance-power efficiency of Gibbs samplers implemented with fixed point arithmetic and functional approximations, image generation and classification were performed using an RBM. Stochastic Gibbs sampling was done for learning and inferencing on an RBM. Test metrics were developed to compare the quality of the models, and power consumption were measured from the Verilog implementation of the RBM. The trade-offs between the two styles of activation function approximation, and among the different levels of quantization, were then identified.

Experimental results show that classification using the Discriminative RBM is less sensitive to the quality of the activation function approximations. Therefore, using the first-order approximations were sufficient. The performance of both spatial and temporal approximations are similar at high bitwidths, but the temporal sampler performs better at low bitwidths. From the Discriminative RBM results, we can conclude that it is most efficient to perform classification at 8 bits.

For the Generative RBM, it is shown that the quality of the activation function approximation is essential to the performance of the generation. Higher order sigmoidal approximation gives a higher efficiency especially at low bitwidths. In general, the temporal approximation performs better than spatial approximation at low bitwidth. From the Generative RBM results, we can conclude that it is most efficient to perform generation at 16 bits.

Since the temporal approximations are proven to outperform the spatial ap-

proximations at low bitwidths, and matches the performance of the spatial approximations at high bitwidths, designs are advised to select the temporal sigmoid approximations over the spatial approximations if the power constraint allows. Additonally, temporal implementations are more flexible to changes in the order of sigmoid approximations than spatial implementations.

A suggested future extension to the work reported in this thesis is to stack hidden layers to the RBM to form the DBN, and perform the same experiments on the DBN. Another possible extension on the hardware side, is to develop the design on the Field-Programable Gate Array (FPGA) for real-time application of image classification and generation tasks. The research can also be applied to a Bayesian Inference.

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$$

where $P(x) = \int P(x|\theta)\, P(\theta)\, d\theta$, which is a similar high dimensional complex computation that can be done using Gibbs sampling.

Bibliography

[1] Versa Technology. *How much Data will The Internet of Things (IoT) Generate by 2020?*. Web, 2015.

[2] M. Deutscher. *Cisco predicts Internet of Things will generate 500 zettabytes of traffic by 2019*. SiliconANGEL, Web, 2015.

[3] Haykin, Simon. *Neural Networks and Learning Machines*. Vol. 3. Upper Saddle River, NJ, USA: Pearson, 2009.

[4] Grinstead, Charles M. Snell, James Laurie. *Introduction to Probability*. American Mathematical Soc., 2012.

[5] Murphy, Kevin P. *Machine Learning A Probabilistic Perspective*. The MIT Press, 2015.

[6] Larochelle, Hugo. Bengio, Yoshua. *Classification using Discriminative Restricted Boltzmann Machines*. In Proceedings of the 25th international conference on Machine learning, pp. 536-543. ACM, 2008.

[7] Carreira-Perpinan, Miguel A. Hinton, Geoffrey E. *On Contrastive Divergence Learning*. Springer, Berlin, Heidelberg, 2012.

[8] Danafar, Somayeh. Rancoita, Paola M.V. Glasmachers, Tobias. Whittingstall, Kevin. Schmidhuber, Jurgen. *Testing Hypotheses by Regularized Maximum Mean Discrepancy*. arXiv preprint arXiv:1305.0423, 2013.

[9] Pedroni, Bruno U. Das, Srinjoy. Arthur, John V. Merolla, Paul A. Jackson, Bryan L. Modha, Dharmendra S. Kreutz-Delgado, Kenneth. Cauwenberghs, Gert. "Mapping Generative Models onto a Network of Digital Spiking Neurons". IEEE transactions on biomedical circuits and systems 10, no. 4: 837-854, 2016.

[10] Das, Srinjoy. Pedroni, Bruno Umbria. Merolla, Paul. Arthur, John. Cassidy, Andrew S. Jackson, Bryan L. Modha, Dharmendra. Cauwenberghs, Gert. Kreutz-Delgado, Ken. "Gibbs Sampling with Low-Power Spiking Digital Neurons". In Circuits and Systems (ISCAS), 2015 IEEE International Symposium on, pp. 2704-2707. IEEE, 2015.

[11] Neopane, Ojash. Das, Srinjoy. Arias-Castro, Ery. Kreutz-Delgado, Ken. *A nonparametric framework for quantifying generative inference on neuromorphic systems*. In Circuits and Systems (ISCAS), 2016 IEEE International Symposium on, pp. 1346-1349. IEEE, 2016.

[12] Marconi, Thomas. Theodoropoulos, Dimitris. Bertels, Koen. Gaydadjiev, Georgi. *A Novel HDL Coding Style to Reduce Power Consumption for Reconfigurable Devices*. In Field-Programmable Technology (FPT), 2010 International Conference on, pp. 295-299. IEEE, 2010.