Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exploring Multi-bit Spike Trains in Neuromorphic Computing

Bachelor Thesis

Yi Zhu

Friday 10th January, 2025

Advisors: Dr. G. Kwasniewski, P. Okanovic, Prof. Dr. T. Hoefler

Department of Computer Science, ETH Zürich

Abstract

Spiking Neural Networks (SNNs) represent an emerging paradigm in machine learning, inspired by biological neurons, with potential for greater energy efficiency compared to traditional Artificial Neural Networks (ANNs). Unlike ANNs, which use continuous floating-point outputs, SNNs produce discrete (binary) spikes when a neurons membrane voltage exceeds a threshold. In this thesis we propose a novel firing model where neurons fire at multiple spiking levels (multi-bit spike trains) when their membrane potentials reach corresponding thresholds. The project explores whether such models can enable more efficient training or inference by facilitating higher communication bandwidth between neurons.

Contents

Chapter 1

Introduction

Artificial neural networks (ANNs) have been widely used in machine learning and artificial intelligence. The computational principle behind ANNs is mostly matrix multiplications which can be efficiently implemented on modern hardware like GPUs. Although such operations can be excellently parallelized and accelerated, the energy consumption of ANNs is still high.

On the other hand, human brains are much more energy-efficient than ANNs. One of the reasons for this energy efficiency is the difference between the neuron models used in ANNs and the biological neurons in the human brain. Unlike ANNs, which use floating-point numbers for precise computation, biological neurons communicate with each other using spike trains and rely on temporal information.

To model such biological neurons and deploy them in artificial neural networks, spiking neural networks (SNNs) have been proposed. The mostly used neuron model in SNNs is the leaky integrate-and-fire (LIF) neuron model [6]. In this model, a neuron's membrane potential is increased by incoming spikes and decreased by a leak term. When the membrane potential reaches a threshold, the neuron fires a spike and resets its membrane potential. The communication between neurons in SNNs is done by sending spikes, which are binary events.

In this thesis, we propose a novel neuron model where neurons can fire at multiple spiking levels. Instead of firing a single spike when the membrane potential reaches a threshold, neurons can fire multiple spikes at different levels. We call this model the multi-bit spike train model. The motivation behind this model is to increase the communication bandwidth between neurons and enable more efficient training or inference in SNNs. We explore the properties of this model and compare it with the traditional LIF neuron model.

We experiment on various datasets and tasks with the multi-bit spike train model and show that it can achieve better performance than the classic 1-bit spike train model in the most cases. Finally, we also present a energy consumption model of the multi-bit spike train model and show that it can be more energy-efficient than the LIF neuron model assuming certain hardware implementations.

Chapter 2

Background

2.1 Leaky Integrate-and-Fire Neuron Model

2.1.1 Dynamics of the Membrane Potential

A leaky integrate-and-fire (LIF) neuron is a simple model of a neuron that captures the essential dynamics of a neuron. The LIF model is described by the differential equation:

$$\tau \frac{\mathrm{d}U(t)}{\mathrm{d}t} = -U(t) + I_{\mathrm{in}}(t) \tag{2.1}$$

where $U(t)$ is the membrane potential of the neuron, τ is the time constant of the neuron, and $I_{\mathrm{in}}(t)$ is the input current to the neuron. The neuron fires a spike when the membrane potential reaches a threshold U_{th}. The membrane potential is then reset to a reset potential U_{reset}.

The equation above has an approximate solution given by:

$$U[t] = \beta U[t-1] + (1-\beta) I_{\mathrm{in}}[t] \tag{2.2}$$

where $\beta = e^{-\Delta t/\tau}$, Δt is the time step, and $I_{\mathrm{in}}[t]$ is the input current at time t, defined by the following equation:

$$I_{\mathrm{in}}[t] = W \cdot X[t] \tag{2.3}$$

where $X[t]$ is the input spike train at time t, and W is the weight matrix.

Since the weights W are learnable parameters, one often merges the weights with the coefficient $(1-\beta)$. In the end, one obtains the following equation:

$$U[t] = \beta U[t-1] + W \cdot X[t] - S_{\mathrm{out}}[t] \cdot \theta \tag{2.4}$$

where $S_{\mathrm{out}}[t]$ is the output spike train at time t, and θ is the threshold of the neuron. By subtracting $S_{\mathrm{out}}[t] \cdot \theta$ from the equation, one resets the membrane potential (soft reset) when the neuron fires a spike.

2.1.2 Spiking Mechanism

Usually the firing model of a neuron is very simple:

$$S_{\text{out}}[t] = \begin{cases} 1 & \text{if } U[t] \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

This is a heaviside step function, which is not differentiable.
For the simplicity of analysis, one often considers $x := U[t] - \theta$ and $y := S_{\text{out}}[t]$. At each time step, the membrane potential $U[t]$ is evaluated through

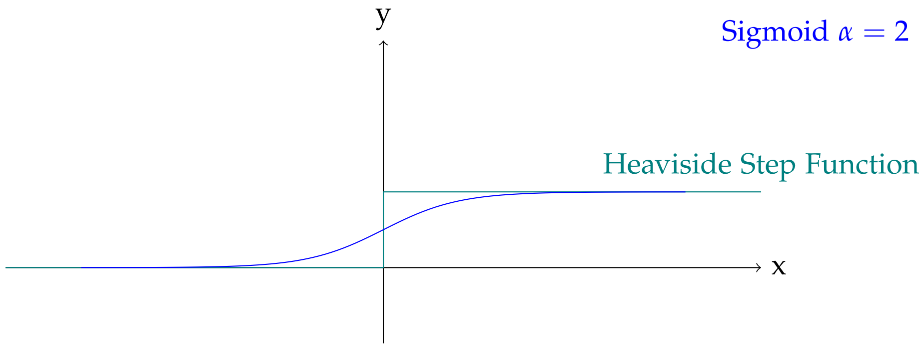


Figure 2.1: Comparison of the Heaviside Step Function and the Sigmoid Function

the Heaviside step function, and the output is a binary value of 0 or 1, illustrated in Figure 2.1. The array of these binary values in the time domain is called the spike train of the neuron.
After the neuron fires a spike, the membrane potential is reset to the reset potential. There are two ways to reset the membrane potential: hard reset and soft reset. In the hard reset, the membrane potential is immediately reset to the reset potential. In the soft reset, the membrane potential is subtracted by the threshold when the neuron fires a spike. Although the hard reset is more biologically plausible and more efficient in terms of computation, the soft reset is more popular in practice because it often delivers better performance in training.

2.2 Training of Spiking Neural Networks

2.2.1 Constructing Spiking Neural Networks

There is in principle no ground-level limitation to the construction of spiking neural networks than the construction of traditional artificial neural networks. One of the most popular methods to construct a spiking neural network is to use the leaky integrate-and-fire (LIF) neuron node to replace the ReLU activation function in the ANNs.

There are also other techniques to replace some components of the ANNs with certain tweaks for the SNNs (e.g. variants of batch normalization and attention mechanisms), but we will not discuss them in this thesis.

2.2.2 Backpropagation through Time

Although our brains are likely not trained by backpropagation, the gradient-based optimization is still the most popular and reliable method to train the neural networks.

Backpropagation through time (BPTT) is a method used to train recurrent neural networks (RNNs) by unfolding the network in time and applying backpropagation. The same method can be applied to train SNNs, as they are very similar to RNNs.

There are also other methods to train SNNs, like SLAYER [8] and EXODUS [2] which utilize the vectorized model of the SNNs. However, we will focus on the temporal model of the SNNs in this thesis.

2.2.3 Surrogate Gradient

Gradient-based optimization requires the activation function to be differentiable. However, the heaviside step function is not. Therefore, one often uses another differentiable function to approximate the heaviside step function in the backpropagation algorithm, e.g. the sigmoid function (see Figure 2.1):

$$S'_{\text{out}}[t] = \frac{1}{1 + e^{-\alpha \cdot x}} \tag{2.6}$$

It turns out that SNNs can tolerate such approximations, and the performance is quite reasonable.

Chapter 3

Multi-bit Spike Train Model

3.1 Graded Spikes

Now we consider the firing model with graded spikes, namely our multi-bit spike train model. We divide the range of $[0, \theta]$ into $2^n - 1$ intervals where n is the number of bits used to encode one single spike. Then once the membrane potential reaches the threshold for a certain interval, the neuron fires a spike with the corresponding intensity described as follows:

$$S_{\text{out}}[t] = \begin{cases} 0 & \text{if } U[t] < \frac{1}{2^n-1} \cdot \theta \\ \frac{i}{1^n-1} & \text{if } \frac{i}{2^n-1} \cdot \theta \leq U[t] < \frac{i+1}{2^n-1} \cdot \theta, i \in [1, 2^n-2] \\ 1 & \text{if } U[t] \geq \theta \end{cases} \tag{3.1}$$

When $n = 1$, the multi-bit spike train model becomes the binary spike train model.
Again we focus on the case $x := U[t] - \theta$ and $y := S_{\text{out}}[t]$, the spike function is now a step function with $2^n - 1$ steps, illustrated in Figure 3.1.

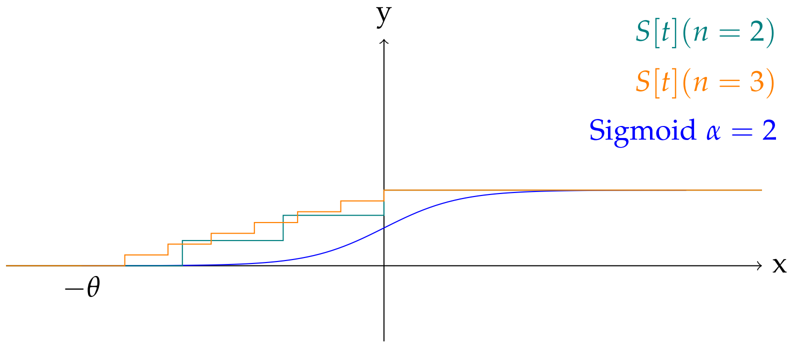


Figure 3.1: Comparison of the Multi-bit Spike Train Model and the Sigmoid Function

Intuitively this firing model enables higher communication bandwidth between neurons. Although biologically, neurons can only fire binary encoded spikes, the spike trains can be rate encoding or temporal encoding. The multi-bit spike train model is a generalization where the graded spikes can be interpreted as rate encoding and their temporal positions can be interpreted as temporal encoding.

3.2 Shifted Surrogate Function

The multi-bit spike train model leads to the problem that the surrogate function no longer approximates the spike function well. One can tell from gap between the multi-bit spike train model and the sigmoid function in Figure 3.1 that the gradients derived from the sigmoid function can no longer reflex the gradients of the multi-bit spike train model accurately. A simple solution is to shift the sigmoid function accordingly. The following equation describes the shifted sigmoid function:

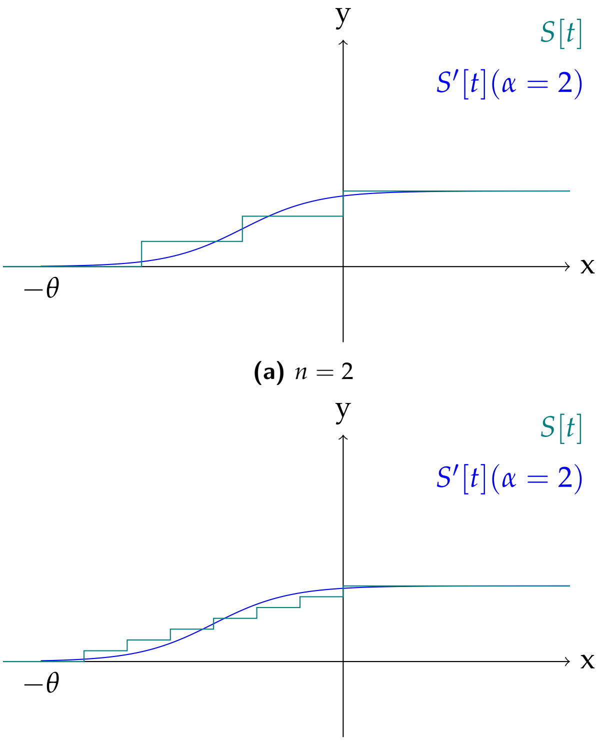
$$S'_{\text{out}}[t] = \frac{1}{1 + \exp(-\alpha \cdot (x + \frac{2^{n-1}-1}{2^n-1} \cdot \theta))} \tag{3.2}$$

This technique centers the sigmoid function relatively to the multi-bit spike train model, illustrated in Figure 3.2. It improves the accuracy of the resulting networks.

3.3 Implementation

We use the SNN framework SpikingJelly which is based on PyTorch to implement the multi-bit spike train model. The firing mechanism is implemented as a forward path of the surrogate function in SpikingJelly. One just needs to override the Heaviside step function with the multi-bit spike train model. The core implementation is shown in Figure 3.3.

Noticing that the input x is already centered to zero by SpikingJelly, so we apply the shift directly to the sigmoid function.



(a) $n = 2$

(b) $n = 3$

Figure 3.2: Comparison of the Shifted Sigmoid Function and the Multi-bit Spike Train Model

```
@torch.jit.script
def multi_level(x: torch.Tensor, n: int, threshold: float):
    l = int(2**n)-1
    r = (x >= 0).float()
    for i in range(1, l):
        r += ((x >= -float(i)/l * threshold) ^ (x >= -float(i
            -1)/l * threshold)) * float(l-i)/l
    return r.to(x)

class sigmoid(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, alpha, n, threshold):
        shift = (2**(n-1) -1) / (2**n-1) * threshold
        if x.requires_grad:
            ctx.save_for_backward(x+shift)
            ctx.alpha = alpha
            ctx.n = n
            ctx.threshold = threshold
        return multi_level(x, n, threshold)

    @staticmethod
    def backward(ctx, grad_output):
        return sigmoid_backward(grad_output, ctx.
            saved_tensors[0], ctx.alpha, ctx.n, ctx.threshold)

class Sigmoid(SurrogateFunctionBase):
    def __init__(self, alpha=4.0, spiking=True, n=1,
        threshold=1.0):
        super().__init__(alpha, spiking, n, threshold)

    @staticmethod
    def spiking_function(x, alpha, n, threshold):
        return sigmoid.apply(x, alpha, n, threshold)

    @staticmethod
    def backward(grad_output, x, alpha, n, threshold):
        shift = (2**(n-1) -1) / (2**n-1) * threshold
        return sigmoid_backward(grad_output, x+shift, alpha,
            n, threshold)[0]
```

Figure 3.3: Implementation of the Multi-bit Spike Train Model in SpikingJelly

Chapter 4

Experiments

We implement the multi-bit spike train model with SpikingJelly [4] using its PyTorch backend, and compare it with the traditional 1-bit spike train model on various tasks and datasets. We investigate the convergence, accuracy, firing rate, and quantizability of the multi-bit spike train model.

We mainly use the Fashion MNIST dataset [9] and a convolutional neural network (CNN) to set up the experiments of an image classification task. We will also show the results also hold on more complex datasets like CIFAR-10 [5].

In the following sections, the input images from Fashion MNIST are first converted to spike trains using the Poisson encoding method. The CNN has a structure of $28 \times 28 - 8c5 - 2a - 16c5 - 2a - 120 - 10o$, visually illustrated in Figure 4.1.

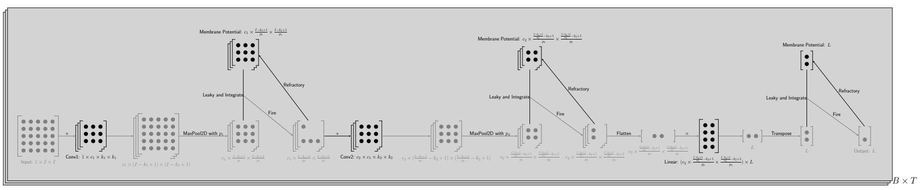
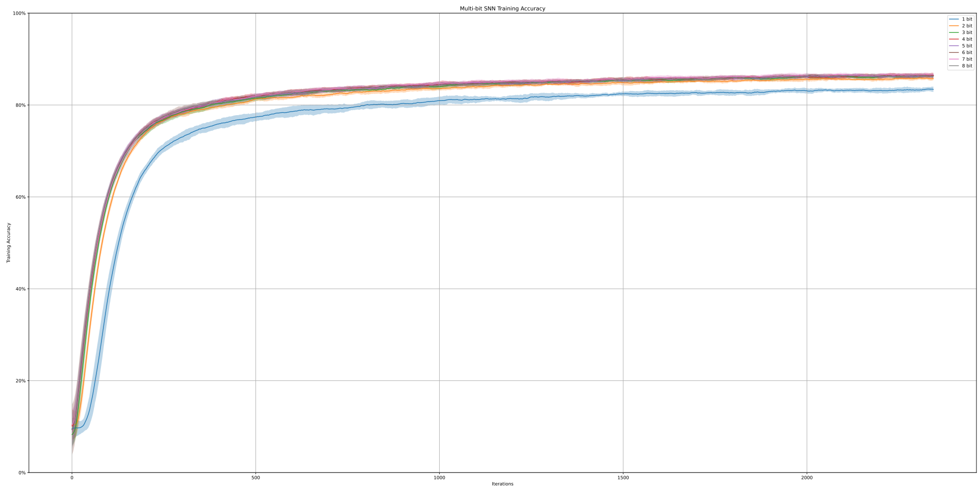


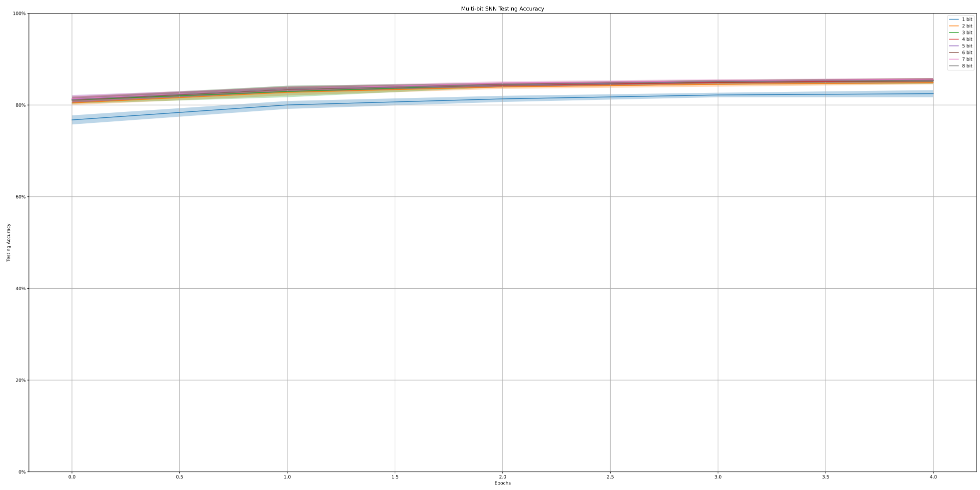
Figure 4.1: The CNN structure used in the experiments with Fashion MNIST dataset

4.1 Convergence & Accuracy

The first noticeable difference between the multi-bit spike train model and the 1-bit spike train model is the convergence speed (see Figure 4.2). Even just by increasing the bit width of the spike train from 1 to 2, the convergence speed of the network is significantly improved.



(a) Training Accuracy (smoothed with a window size of 100)

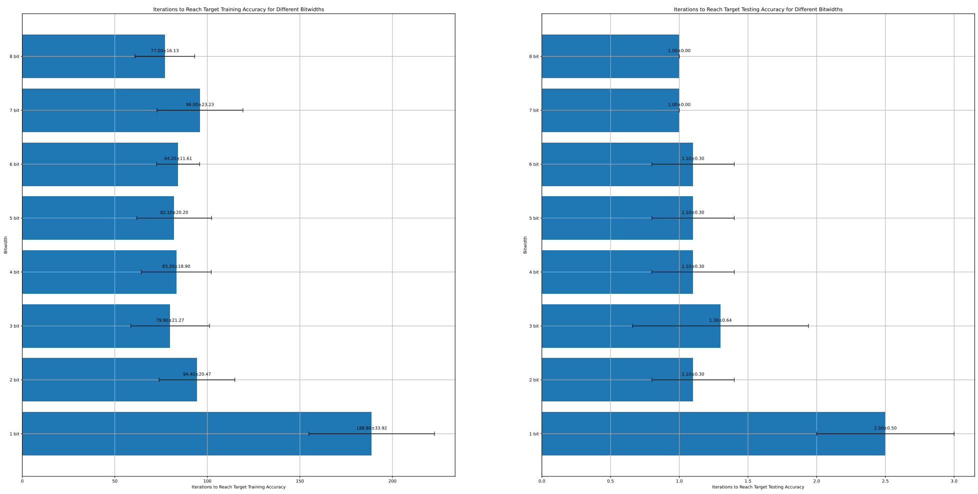


(b) Test Accuracy

Figure 4.2: Comparison of the Convergence Speed of 1-bit to 8-bit Spike Train Model, repetition of the experiment 10 times

It may not seem to be much graphically, but the improvement in convergence speed can lead to significant reduction in the training time if considering a fixed target accuracy, especially considering the diminishing return of the training time with respect to the target accuracy.

Here we set the target accuracy to be 80%. The training time of the multi-bit spike train model is reduced by around 50%, shown in Figure 4.3.



(a) Iterations to reach 80% training accuracy **(b)** Epochs to reach 80% testing accuracy

Figure 4.3: Comparison of the Training Time of 1-bit to 8-bit Spike Train Model, repetition of the experiment 10 times

Such great improvement however also comes at certain cost:

- The improvement in convergence speed diminishes as the bit width of the spike train increases. The improvement from 2-bit to 3-bit or even higher bit width is not as significant as the improvement from 1-bit to 2-bit.

- The more complicated the model is, the harder is it to train the multi-bit spike train model. One can easily run into the problem of overfitting, for example in the case of DVS gesture recognition task.

Due to the faster convergence speed, the multi-bit spike train model can achieve better accuracy than the 1-bit spike train model given a fixed number of iterations or epochs for the most cases (see Figure 4.4).

In general, such behaviors can be shown on MNIST [3], NMNIST [7], Fashion MNIST, DVS gesture recognition [1], and CIFAR-10 datasets.

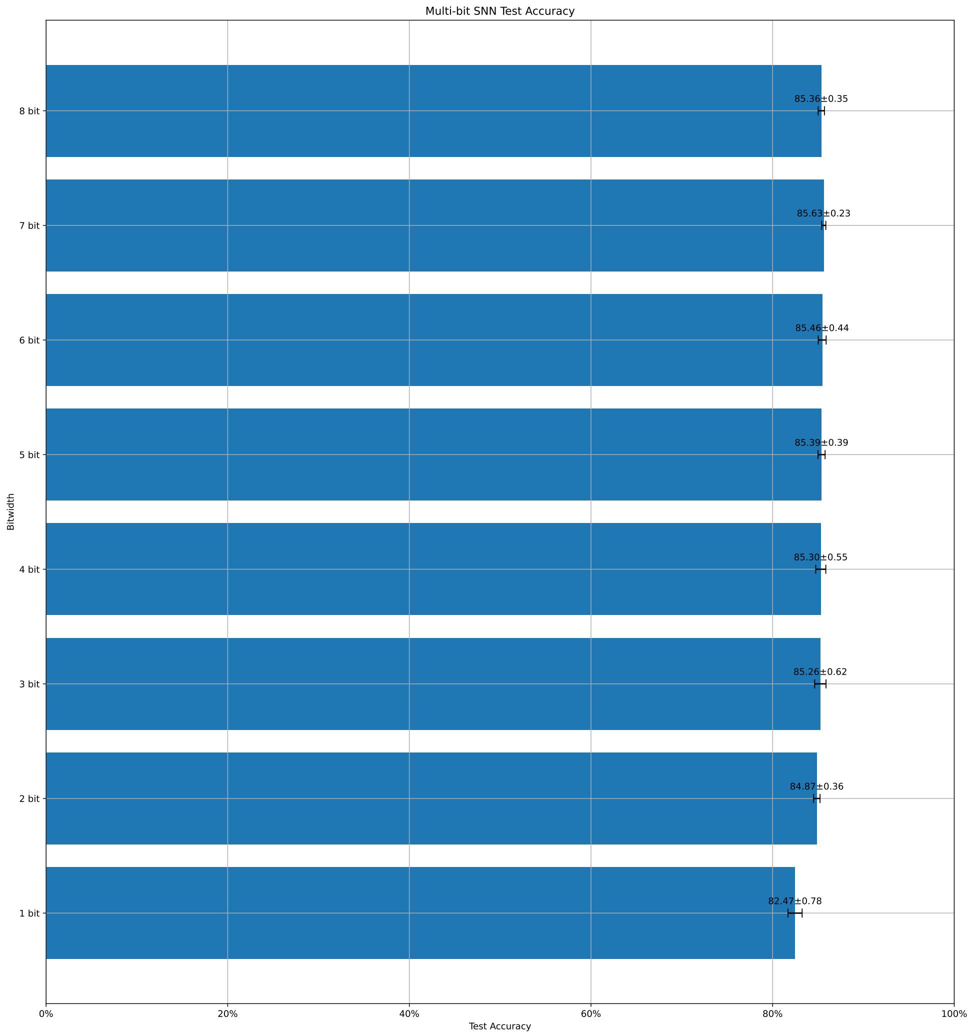
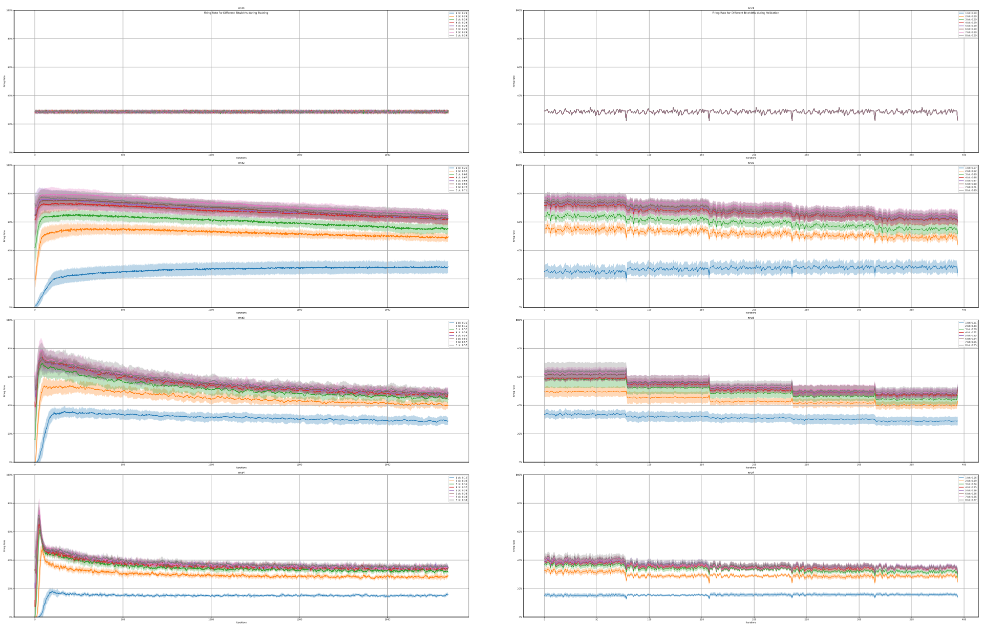


Figure 4.4: Comparison of the Final Accuracy of 1-bit to 8-bit Spike Train Model after 5 epochs, repetition of the experiment 10 times

4.2 Firing Rate

SNNs are known for their sparsity in the firing rate of the neurons. Here we notice that the multi-bit spike train model has a higher firing rate than the 1-bit spike train model (see Figure 4.5), as tradeoff for the improvement in convergence speed.



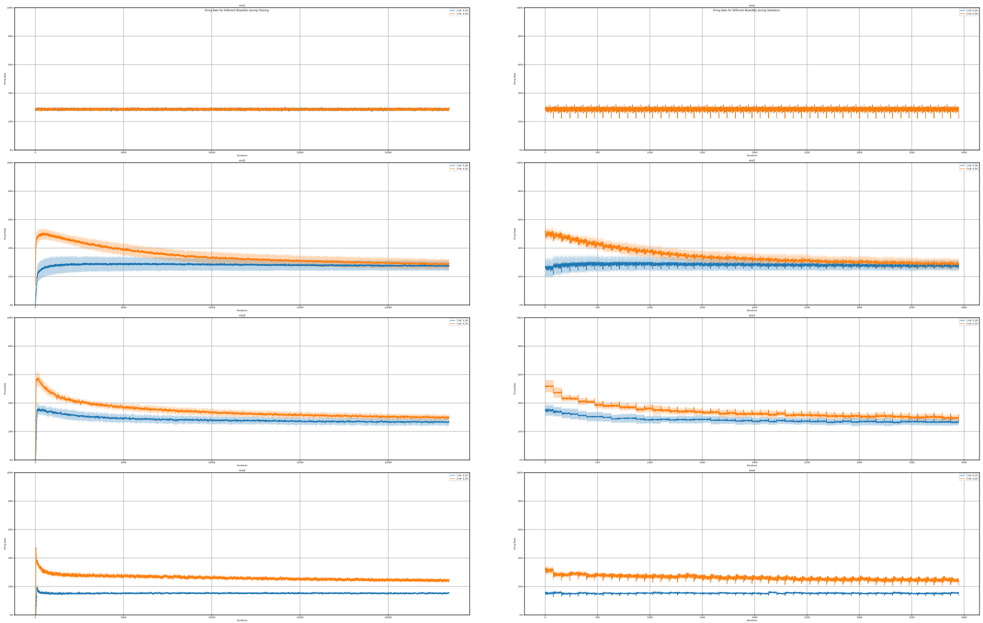
(a) Training Firing Rate **(b)** Test Firing Rate

Figure 4.5: Comparison of the Firing Rate of 1-bit to 8-bit Spike Train Model: repetition of the experiment 10 times, "nnz[i]" means the number of non-zero elements in the i-th measuring point, corresponding to the position from the inputs to the output layer, "nnz1" is the input data

One can notice that the firing rate of the multi-bit spike train model peaks at the very beginning of the training session, when the model converges rapidly. The firing rate then decreases as the training progresses. If we extend the training session, the firing rate of the multi-bit spike train model tends to converge to the firing rate of the 1-bit spike train model. Such process takes however a long time (see Figure 4.6).

In the end, the firing rate of the multi-bit spike train model is comparable to the 1-bit spike train model (see Figure 4.7).

Such behavior is more visible on simpler datasets like MNIST. For now, we do not have any explanation for this phenomenon.



(a) Training Firing Rate **(b)** Test Firing Rate

Figure 4.6: Analog to figure 4.5, but with a training session of 50 epochs instead of 5 epochs, only the 1-bit and 2-bit spike train models

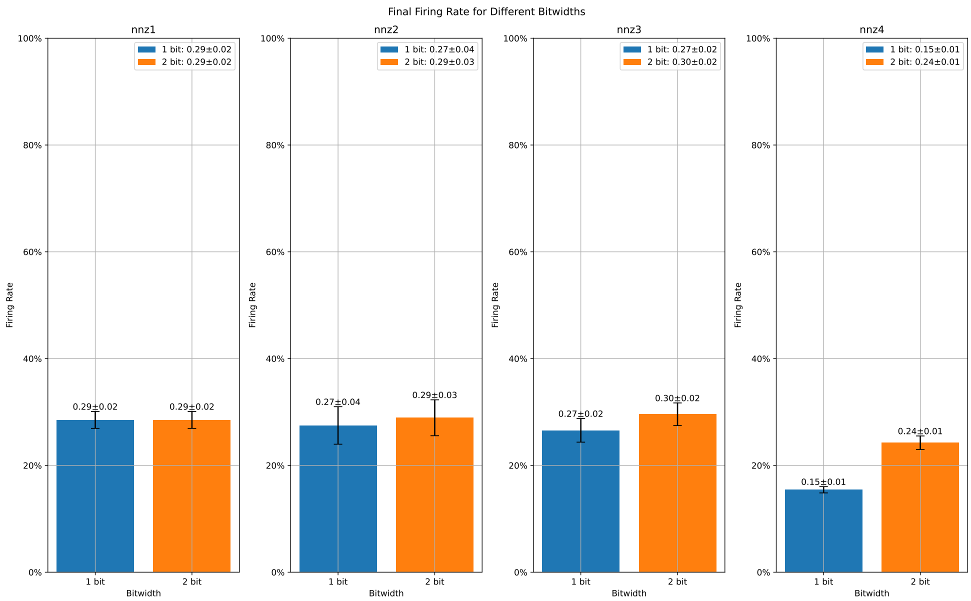
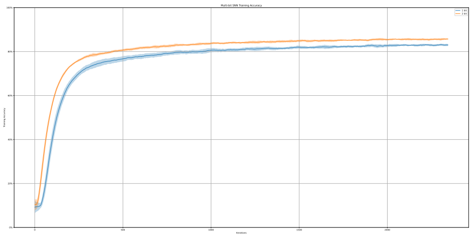


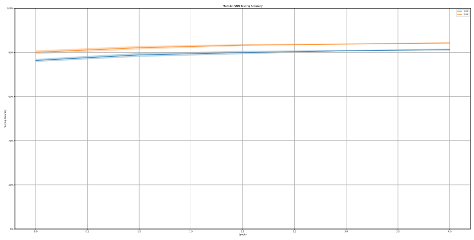
Figure 4.7: Comparison of the Final Firing Rate of 1-bit to 8-bit Spike Train Model after 50 epochs, only the 1-bit and 2-bit spike train models

4.3 Quantizability

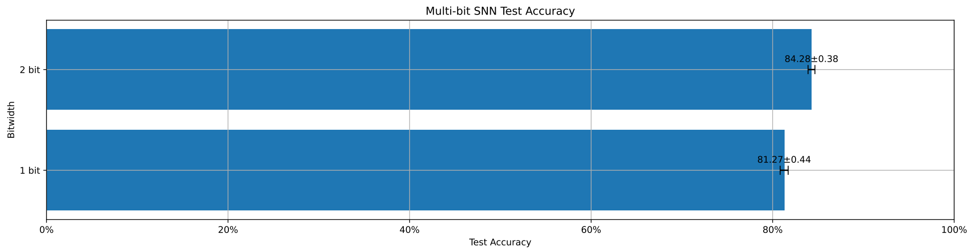
It is well known that SNNs are easy to quantize. There are studies showing that the weights and activations of SNNs can be quantized to 1-bit or 2-bit without significant loss in accuracy. Here we show that both the multi-bit spike train model and the 1-bit spike train model can be trained with `bf16` and quantized to `int8` without significant loss in accuracy.



(a) Training Accuracy with half precision (smoothed with a window size of 100)



(b) Test Accuracy with half precision



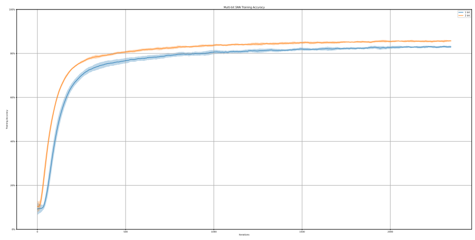
(c) Final Accuracy with half precision after 5 epochs

Figure 4.8: Comparison of the Convergence Speed of 1-bit to 8-bit Spike Train Model with half precision, repetition of the experiment 10 times

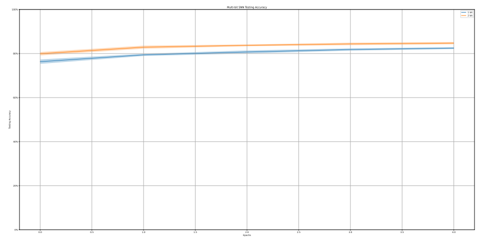
As shown in Figure 4.8, both SNN models converge well as if they were trained with `float32`. Meanwhile the multi-bit spike train model has its advantage in the convergence speed and accuracy preserved.

Before quantizing the models to `int8`, we first apply quantization-aware training to both SNN models. While regular ANNs may encounter various problems, both SNN models here converge well. We also see that the multi-bit spike train model has a faster convergence speed and better accuracy than the 1-bit spike train model (Figure 4.9).

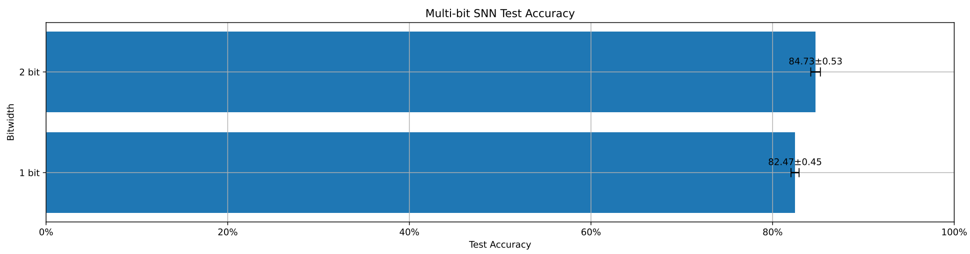
We then quantize the weights and biases to `int8` using the PyTorch quantization API. The quantization of the LIF layer is not supported at the moment. We evaluate the quantized models on the test set. The final accuracy is barely affected by the quantization (see Figure 4.9c).



(a) Training Accuracy with quantization-aware training (smoothed with a window size of 100)



(b) Test Accuracy with quantization-aware training



(c) Final Accuracy with quantization-aware training after 5 epochs

Figure 4.9: Comparison of the Convergence Speed of 1-bit to 8-bit Spike Train Model with quantization-aware training, repetition of the experiment 10 times

Chapter 5

Evaluation

5.1 Energy Consumption

One of the main motivations for SNNs is their energy efficiency compared to ANNs on specialized hardware. Products like Loihi from Intel and TrueNorth from IBM have proved the potential of SNNs by utilizing the asynchronous communication via spikes. In the real-world scenario, one often uses accelerators like GPUs to train SNNs and deploy them on specialized hardware to achieve fast training and energy efficent inference.

Here we present an energy consumption model for the multi-bit spike train model and compare it with the 1-bit spike train model. We consider the unique properties of various hardware implementations and give the energy consumption of the multi-bit spike train model relative to the 1-bit spike train model.

5.1.1 Training Energy Consumption on GPUs

On GPUs, low precision spikes are generally not very meaningful, as the hardware is not designed for such level mixed precision operations. Often the spikes are represented as 32-bit floating point numbers, which can be computed with the weights with the same precision. And the popular SNN frameworks like snnTorch and SpikingJelly do not utilize the sparsity of the spike trains. So here, the firing rate and the bit width of the spike train do not affect the energy consumption of the training phase.

The only factors that matter are the number of iterations required to reach a certain accuracy and the number of time steps that the network is simulated, assuming fixed network topology and batch size.

This allows us to create a simple, yet effective energy consumption model for the training phase on the GPUs. Let T_i denote the number of time steps,

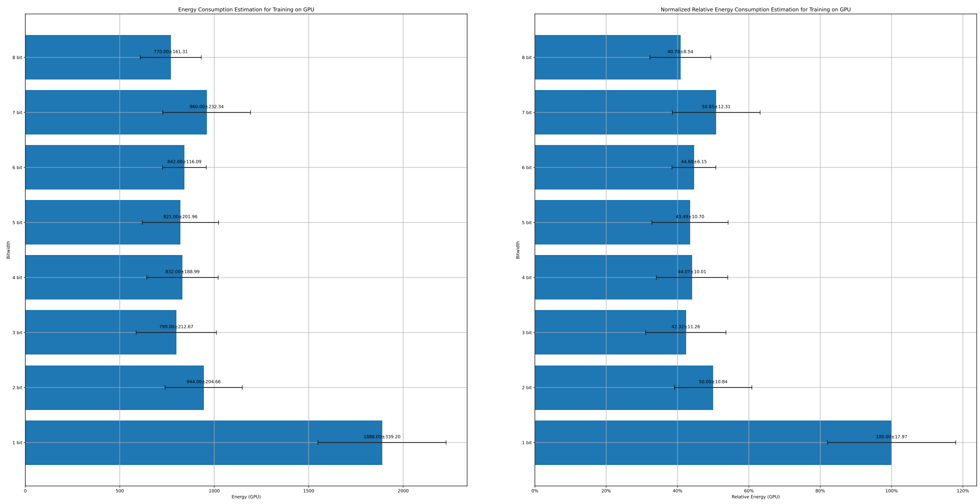
S_i denote the number of iterations required to reach a certain accuracy, and $E_{\text{train}-i}$ denote the energy consumption. Then we have:

$$E_{\text{train}-i} = T_i \cdot S_i \cdot c \tag{5.1}$$

where as c is a constant factor that depends on the hardware and the software used, however remains the same across different bit widths of the spike train.

As noticed in Section 4.1, one requires fewer iterations to reach a certain accuracy with the multi-bit spike train model. Here we focus on the energy consumption of the 2-bit spike train model, as it does not increase the firing rate as much as the other higher bit width models while still providing a significant improvement in convergence speed and accuracy.

Based on the results in Section 4.3, we can estimate the energy consumption of the 2-bit spike train model relative to the 1-bit spike train model directly by comparing the number of iterations required to reach a certain accuracy which is around $50.00 \pm 10.84\%$ in this case (see Figure 5.1).



(a) Energy Consumption Estimation, Unit in Constant c

(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure 5.1: Training Energy Consumption Estimation on GPUs for Fashion MNIST Dataset

5.1.2 Inference Energy Consumption on Neuromorphic Chips

A widely adopted energy estimation model for neuromorphic chips is the following:

$$E_{\text{inference}} = F \cdot fr \cdot E_{\text{AC}} \cdot T \tag{5.2}$$

where F is the number of floating point operations required to simulate the network, fr is the firing rate of the neurons, E_{AC} is the energy consumption of accumulation, and T is the number of time steps.

It is however hard to evaluate the energy consumption of the multi-bit spike train model on neuromorphic chips like Loihi and TrueNorth, as they do not support multi-bit spikes natively. Although it may be possible to encode the multi-bit spikes into multiple spikes with different intensities, the energy consumption of such encoding is very expensive due to the high cost of the synchronization barrier between the time steps.

A viable option is to consider hardware like Intel Loihi 2 which supports graded spikes up to 32-bit precision. We consider the case of Intel Loihi 2 and make the following assumptions:

1. There is no difference in the energy consumption for the number of bits used to encode the spikes.
2. The energy consumption for a floating point operation is always E_{MAC} (Multiply-Accumulate).

Since the payload is not variable in this case, the energy consumption of the multi-bit spike train model is directly proportional to the firing rate given fixed maximum floating point operations and time steps. Analog as the equation 5.2, we have the following estimation for the i-bit spike train model:

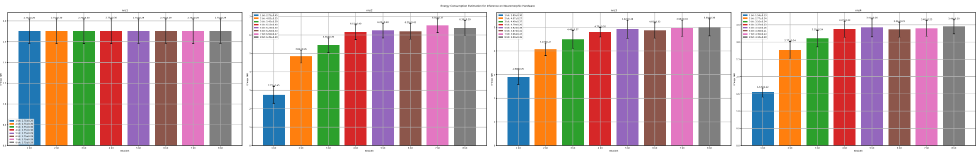
$$E_{\text{inference}-i} = F \cdot fr_i \cdot E_{\text{MAC}} \cdot T_i \tag{5.3}$$

We can estimate the energy consumption of the 2-bit spike train model relative to the 1-bit spike train model by comparing the firing rate of the neurons. As expected, the energy consumption of the multi-bit spike train model is higher than the 1-bit spike train model, as the firing rate of the neurons is higher (see Figure 5.2).

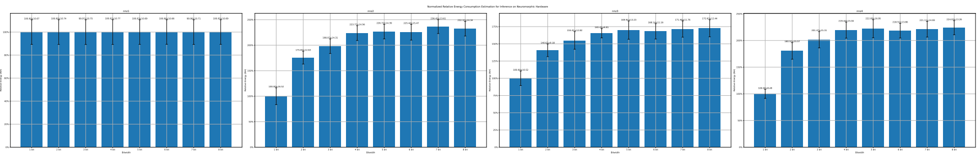
5.1.3 Tradeoffs

One can tell that the energy consumption of the multi-bit spike train model has no direct advantage over the 1-bit spike train model on neuromorphic chips, as the firing rate of the multi-bit spike train model tends to be higher than the 1-bit spike train model. Moreover, one can also argue that the energy consumption from E_{MAC} is due to the graded spikes, which is not an issue with binary spikes.

We consider the energy consumption of the multi-bit spike train model in general as an opportunity to enable tradeoffs. If the inference is not the bottleneck of the application, then one can rely on the fast convergence speed of the multi-bit spike train model during training. If the inference is the bottleneck, then one can choose to train the multi-bit spike train model for



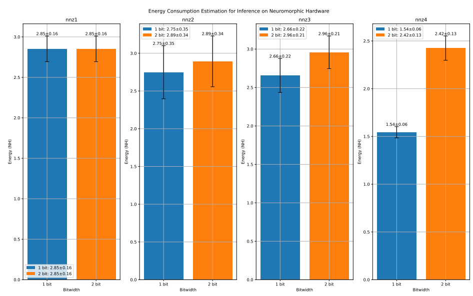
(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\mathrm{MAC}}$



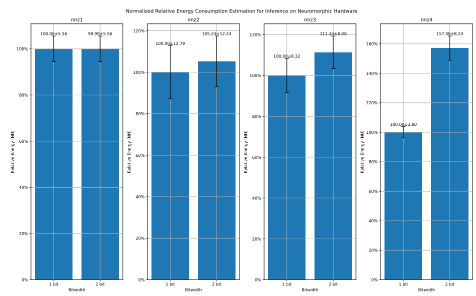
(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure 5.2: Inference Energy Consumption Estimation on Intel Loihi 2 for Fashion MNIST Dataset

longer time to achieve a firing rate that is comparable to the 1-bit spike train model (see Figure 5.3). Such tradeoffs are not possible with the 1-bit spike train model.



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\mathrm{MAC}}$



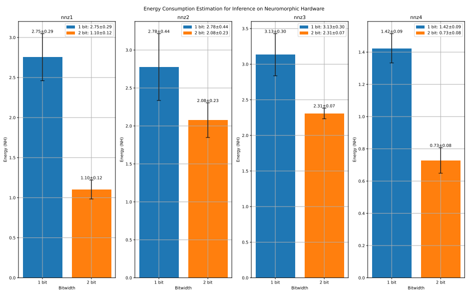
(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model

Figure 5.3: Inference Energy Consumption Estimation on Intel Loihi 2 for Fashion MNIST Dataset with 50 Training Epochs

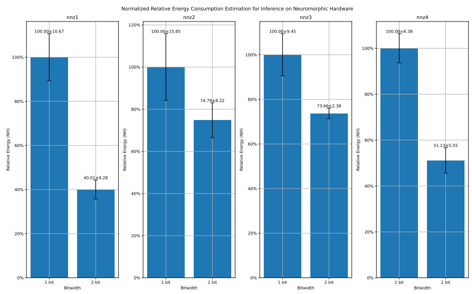
And more interestingly, if one is satisfied with the accuracy of the 1-bit spike train model, then one can choose to train the multi-bit spike train model for fewer time steps. This can enable higher efficiency in both training and inference.

We take again the example of Fashion MNIST dataset, while $T = 10$ is a good choice for both the 1-bit and 2-bit spike train model, we can reduce the time steps to $T = 4$ for the 2-bit spike train model and still achieve a comparable accuracy (see 5.4).

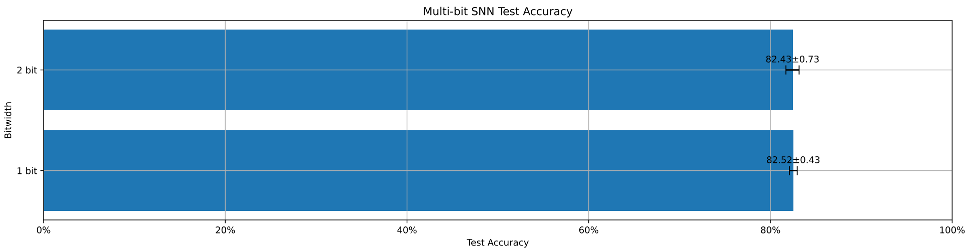
This can lead to a significant reduction in the energy consumption. It may not be reflected as a direct advantage in the energy consumption model



(a) Energy Consumption Estimation, Unit in Parameters $F \cdot E_{\text{MAC}}$



(b) Normalized Energy Consumption Estimation Relative to 1-bit Spike Train Model



(c) Accuracy Comparison

Figure 5.4: Inference Energy Consumption Estimation on Intel Loihi 2 and Test Accuracy for Fashion MNIST Dataset with 10 Time Steps for 1-bit Spike Train Model and 4 Time Steps for 2-bit Spike Train Model

mentioned above 5.1.2, but in practice, it should bring significant benefits, as most of the neuromorphic chips are designed to utilize the asynchronous communication via spikes, so they do not have a central clock system to synchronize the time steps very efficiently, and the cost for the synchronization barrier is very high. As reference, the latency per tile hop on Intel Loihi is at most around 6.5 ns where as the latency for the synchronization barrier is from 113-465 ns.

5.2 Performance

In the section 5.1.1, we claim that the energy consumption on the GPUs are not affected by the firing rate and the bit width of the spike train in theory. However, in practice, with the increase of the bit width of the spike train, the training process slows down. This can be caused by the inefficient implementation of the multi-bit spike train model and the low level optimization on operations of sparse matrix multiplication. There is sufficient room for improvement, e.g. by utilizing the sparsity of the spike trains or completely switching to the vectorized model instead of the temporal model instead, which is shown to be more efficient with learning algorithms like SLAYER and EXODUS.

Chapter 6

Concluding Remarks

6.1 Conclusion

In this thesis we have developed a noval spike train model for spiking neural networks (SNNs) that uses multi-bit spikes to encode the information. We implement such model using an SNN framework, SpikingJelly, based on PyTorch. And we have shown that the multi-bit spike train model can significantly improve the convergence speed and accuracy of the network compared to the traditional 1-bit spike train model while preserving other characteristics of the 1-bit spike train model such as high quantizability and low energy consumption. We consider the tradeoffs the multi-bit spike train model can bring in terms of energy consumption and performance important for maximizing the efficiency of SNNs on various hardware platforms.

6.2 Future Work

The multi-bit spike train model is a promising direction for the development of SNNs. However, there are still many open questions and challenges that need to be addressed in the future. Here we list some of the possible future work:

- **Optimization of the multi-bit spike train model:** The current implementation of the multi-bit spike train model is not optimized for performance. One can consider using just-in-time (JIT) compilation to improve the performance of the model.
- **Investigation of the overfitting problem:** The multi-bit spike train model is more complicated than the 1-bit spike train model, which can lead to overfitting. One can investigate the overfitting problem and propose solutions to mitigate it.

- **Extension to other tasks and datasets:** The experiments in this thesis are mainly focused on image classification tasks. One can extend the multi-bit spike train model to other tasks and datasets to evaluate its performance.
- **Implementation on neuromorphic chips:** The multi-bit spike train model is designed to be hardware-friendly in theory. It would be more convincing if one can implement the model on neuromorphic chips like Intel Loihi 2 to evaluate its performance on specialized hardware.
- **Investigation of the energy consumption model:** The energy consumption model presented in this thesis is a simple estimation. One can investigate the energy consumption of the multi-bit spike train model more thoroughly and propose a more accurate model. Ideally, one can also measure the energy consumption of the multi-bit spike train model after implementing it on neuromorphic chips.

Appendix A

Accuracy of the Multi-Bit Spike Train Model

Appendix B

Firing Rate in Different Positions of the Multi-Bit Spike Train Model

Bibliography

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7388–7397, 2017.

[2] Felix Christian Bauer, Gregor Lenz, Saeid Haghighatshoar, and Sadique Sheik. Exodus: Stable and efficient training of spiking neural networks. *arXiv preprint arXiv:2205.10242*, 2022.

[3] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[4] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothe Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40):eadi1480, 2023.

[5] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[6] Louis Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J. of Physiol. and Pathology*, 9:620–635, 1907.

[7] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, 2015.

[8] Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1419–1428. Curran Associates, Inc., 2018.

[9] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.