Impact of time-discretization on the efficiency of continuous time Spiking Neural Networks

The effects of the time step size on the accuracy, sparsity and latency of the SNN.

Alexandru Gabriel Cojocaru

Supervisors: Nergis Tömen, Aurora Micheli

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Final project course: CSE3000 Research Project
Thesis committee: Nergis Tömen, Aurora Micheli, Lilika Markatou

Abstract

The increasing computational costs of training deep learning models have drawn more and more attention towards more power-efficient alternatives such as spiking neural networks (SNNs). SNNs are an artificial neural network that mimics the brain's way of processing information. These models can be even more power-efficient when run on specialized hardware like digital neuromorphic chips. These chips are designed to handle the unique processing needs of SNNs like sparse and event-driven computations. A lot of the state-of-the-art performance of SNNs in recent research has been achieved through supervised learning models that leverage intricate error backpropagation techniques. These models impose specific constraints on the network and rely on continuous time to facilitate the backpropagation process. However, this fix imposes a new challenge when converting these mechanisms on a neuromorphic chip. Because time is discrete on hardware numerical errors can be introduced as we can not calculate the infinitely precise value of variables depending on time. This work proposes a time discretization technique that allows for fast and stable backpropagation and analyses the effects of it on the efficiency of the SNN. Specifically, we look at sparsity, latency, and accuracy as the main factors that explain the power efficiency of the model. The experiments show that choosing a suitable time step size can improve sparsity while maintaining a high level of accuracy. However, too large values affect the network's ability to learn.

1 Introduction

In recent years, tremendous technological achievements have been seen in deep learning, with advancements in large language models such as GPT-4 [1] providing state-of-the-art results. On the other hand, with such fast advancement of technology in deep learning, the computational costs also increased to alarming levels. Consequently, it becomes increasingly important to make accurate models power efficient. For example, GPT-3 is estimated to consume roughly 190.000 kWh to train [2]. In contrast, the human brain operates within 12-20W of power. This is why a good place to look when it comes to efficiency is the human brain.

Spiking neural networks(SNNs) are artificial neural networks that more closely mimic natural neural networks. They incorporate into their model not only neuronal and synaptic states but also the concept of time. Spiking neurons handle information using distinct spatiotemporal events known as spikes, rather than continuous real-number values [3]. While SNNs lag on accuracy when compared to deep neural networks, they have proven to be more energy-efficient [4] especially when implemented on digital neuromorphic hardware such as Intel Loihi [5], IBM TrueNorth [6], BrainscaleS-2 [7] etc.

On neuromorphic hardware, energy is consumed when a signal is generated [8]. Therefore, the power consumption of neuromorphic devices is closely related to the number of spikes produced. This addresses the energy burden as the need for fewer spikes reduces the frequency of memory accesses [8] making *sparsity* of activations an important factor in analyzing the efficiency of an SNN model. However, sparse networks that only fire a small number of spikes also transmit less information creating a trade-off between energy consumption and accuracy that needs to be examined.

One approach to encoding information in a spiking neural network that allows for sparse networks is *Time-to-first-spike (TTFS)*. Inspired by the human visual system it is based on the idea that the first spikes of neurons must carry most of the information about input stimuli. In the Time-To-First-Spike (TTFS) coding scheme, a neuron's response to a stimulus is represented by the time it takes to emit its first spike after being stimulated. Recent research has shown that time-to-first-spike can achieve competitive results with great efficiency [9]. TTFS emphasizes the significance of individual spikes compared to rate-coded inputs, suggesting that fewer spikes are needed to achieve similar results.

Using inputs that encode information based on spike timing (latency-coded inputs) introduces another metric that helps assess the efficiency of spiking neural networks: *prediction latency*. Prediction latency refers to the ability of the SNN to make a fast prediction. This metric is crucial because low latency is highly desirable in various SNN applications and has been thoroughly studied in recent research [10] [11]. Minimizing the latency of an SNN can enhance its performance in real-time processing tasks. An SNN can achieve faster decision-making with earlier spikes, reducing energy consumption.

The standard TTFS encoding has one fundamental limitation. It does not allow multiple spikes per neuron. An increase in information gain and performance would be expected by relaxing this constraint, which has been proven in recent research [12] [13].

When examining various Spiking Neural Network models, it becomes apparent that unsupervised learning rules, such as Spike Time Dependent Plasticity[14], can be directly implemented on neuromorphic hardware. This allows for biologically plausible and energy-efficient training. However, unsupervised learning algorithms tend to underperform compared to the results obtained through supervised learning methods. State-of-the-art performance with SNNs is currently achieved by employing various *error backpropagation* techniques adapted from deep learning such as error backpropagation through spikes, backpropagation through time, [15] [9] [13] [12] etc.

Models such as BATS [12] or Fast&Deep [9] rely on specific assumptions that allow them to find exact values for the timing of spikes. They introduce a novel approach to backpropagating errors that achieves state-of-the-art results. However, since these systems depend on precise time calculations in a continuous space, it is unclear how easily they can be translated to digital neuromorphic hardware to enhance their power efficiency. This is because digital neuromorphic chips manage internal communication by leveraging the discrete

property of time, which conflicts with the nature of SNN models as BATS. As a result, we are interested in analyzing the effects of time discretization on these models.

In this research, we discretize the simulation time of a continuous time SNN model from the literature that leverages backpropagation through spikes. This creates a hardware-aware training process that can be used in the future to estimate the inference performance on neuromorphic hardware of different software-trained SNN models. More specifically we will answer the question "What is the impact of time discretization on the efficiency of an unconstrained SNN?" with the sub-questions:

- What is the impact of changing the time step on the accuracy of the SNN?
- What is the impact of changing the time step on the sparsity of the SNN?
- What is the impact of changing the time step on the latency of the SNN?

The rest of the report is structured as follows: Chapter 2 describes the BATS model and key equations that explain the training or inference process. Chapter 3 presents our approach to time discretization, plus the approach to backpropagation and the evaluation metrics. Chapter 4 explains the experimental setup and the results obtained for each dataset. In Chapter 5 an extensive analysis of the results is conducted and some conclusions are drawn regarding the behaviour of the network. Chapter 6 concludes the report and some directions for future work on this topic. Chapter 7 discusses ethical and societal concerns regarding the research conducted.

2 Methodology

In this section, we will outline the background required for the topic including the model used, some terminology, notations, and specifically, key equations that will be important to understanding the underlying mechanics of this SNN.

2.1 BATS model

This research continues over the BATS model[12]. This model has been chosen for two key reasons: its underlying ideology and its technical advantages. Firstly, BATS is a recent SNN model that calculates a closed-form solution for the timing of spikes, which is the main challenge of this research. Secondly, the implementation of this model has been written from scratch using CuPy[16] which allows for flexibility in the implementation, quick access to the memory and implicit parallelization due to CUDA kernels, making it an already efficient model.

2.2 Neuron mechanism

In BATS a CuBa (Current-Based) LIF (Leaky-Integrate-and-Fire) neuron with a soft reset of the membrane potential is used. The membrane potential represents the neuron's current state that changes over time based on incoming signals. The value of the membrane potential follows a system of linear ordinary differential equations :

$$\frac{du^{(l,j)}}{dt} = -\frac{1}{\tau}u^{(l,j)}(t) + g^{(l,j)}(t) - \vartheta\delta\left(u^{(l,j)}(t) - \vartheta\right) \quad (1)$$

$$\frac{dg^{(l,j)}}{dt} = -\frac{1}{\tau_s}g^{(l,j)}(t) + \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \delta\left(t - t_z^{(l-1,i)}\right) \quad (2)$$

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

We denote $u^{(l,j)}$ as the membrane potential of the j-th neuron in the layer. $n^{(l,j)}$ represents the number of spikes from the j-th neuron in layer l and N^l represents the number of neurons in the l-th layer. $g^{(l,j)}$ represents the post-synaptic effect. Each spike received by the neuron $n^{(l,j)}$ implies an increase in $g^{(l,j)}$ by a value $w_{i,j}^{(l)}$. $\delta(x)$ is the Dirac delta function that satisfies $\int_{-\infty}^{+\infty}\delta(x)\,dx = 1$. Throughout this paper, we will refer to post-synaptic and pre-synaptic which refer to connections to the adjacent layers. We denote by τ and τ_s the membrane and synaptic time constants that control the decay over time of the membrane potential and the post-synaptic current respectively.

The spike response model(SRM) [3] is a generalization of the LIF neuron that defines the membrane potential as a linear sum of postsynaptic potentials caused by spike arrivals. The membrane potential is defined as a function of time as follows:

$$u^{(l,j)}(t) = \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \epsilon\left(t - t_z^{(l-1,i)}\right) - \\ - \sum_{z=1}^{n^{(l,j)}} \eta\left(t - t_z^{(l,j)}\right) \quad (4)$$

By integrating we find the following post-synaptic potential (PSP) and refractory kernels ($\epsilon(t)$ respectively $\eta(t)$). PSP defines the response of a neuron to a stimulus and the refractory kernel defines the reset behaviour.

$$\epsilon(t) = \Theta(t)\frac{\tau\tau_s}{\tau - \tau_s}\left[\exp\left(\frac{-t}{\tau}\right) - \exp\left(\frac{-t}{\tau_s}\right)\right] \quad (5)$$

$$\eta(t) = \Theta(t)\vartheta\exp\left(\frac{-t}{\tau}\right) \quad (6)$$

Where theta is the Heavyside step function.

$$\Theta(x) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

By taking $\tau = 2\tau_s$ the BATS model shows that the spike time of the neurons must satisfy the following polynomial :

$$0 = -a_k^{(l,j)}\exp\left(-\frac{t_k^{(l,j)}}{\tau}\right)^2 + b_k^{(l,j)}\exp\left(-\frac{t_k^{(l,j)}}{\tau}\right) - c_k^{(l,j)}$$

Where:

$$a_k^{(l,j)} = \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right)\exp\left(\frac{t_z^{(l-1,i)}}{\tau_s}\right)$$

$$b_k^{(l,j)} = \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right)$$
$$\exp\left(\frac{t_z^{(l-1,i)}}{\tau}\right) - \frac{\vartheta}{\tau} \sum_{z=1}^{n^{(l,j)}} \Theta\left(t_k^{(l,j)} - t_z^{(l,j)}\right) \exp\left(\frac{t_z^{(l,j)}}{\tau}\right)$$

$$c_k^{(l,j)} = \frac{\vartheta}{\tau}$$

Which solves to:

$$t_k^{(l,j)} = \tau \ln\left[\frac{2a_k^{(l,j)}}{b_k^{(l,j)} + x_k^{(l,j)}}\right] \tag{8}$$

Where

$$x_k^{(l,j)} = \sqrt{\left(b_k^{(l,j)}\right)^2 - 4a_k^{(l,j)} c_k^{(l,j)}} \tag{9}$$

2.3 Backpropagation Derivations

To calculate the loss we define the following function:

$$\mathcal{L} := \frac{1}{2} \sum_{j=1}^{N^{(o)}} \left(y_j - n^{(o,j)}\right)^2 \tag{10}$$

Where y_j is the spike count target. The goal is to minimize the difference between the output spikes and their corresponding targets.

Having the closed-form solution for a spike time allows us to calculate all the partial derivatives necessary for proper backpropagation through spikes. First, we look at the total weight change between two neurons. We define $\delta_k^{(l,j)}$ as the error received by spike k of the neuron j in layer l. The change of weight between the pre-synaptic neuron i and the post-synaptic neuron j of the layer l is defined as a sum of all errors applied to their corresponding spike derivatives:

$$\Delta w_{i,j}^{(l)} = \sum_{k=1}^{n^{(l,j)}} \frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}} \frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} = \sum_{k=1}^{n^{(l,j)}} \delta_k^{(l,j)} \frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} \tag{11}$$

Where

$$\frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} = \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right) \cdot$$
$$\cdot \left[f_k^{(l,j)} \exp\left(\frac{t_z^{(l-1,i)}}{\tau_s}\right) - h_k^{(l,j)} \exp\left(\frac{t_z^{(l-1,i)}}{\tau}\right)\right] \tag{12}$$

Is obtained by calculating the partial derivative of equation 8 with respect to the weight $w_{i,j}^{(l)}$ and:

$$f_k^{(l,j)} := \frac{\partial t_k^{(l,j)}}{\partial a_k^{(l,j)}} = \frac{\tau}{a_k^{(l,j)}} \left[1 + \frac{c}{x_k^{(l,j)}} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right)\right]$$

$$h_k^{(l,j)} := \frac{\partial t_k^{(l,j)}}{\partial b_k^{(l,j)}} = \frac{\tau}{x_k^{(l,j)}}$$

Therefore the weight $w_{i,j}^{(l)}$ can be updated using the gradient descent algorithm : $w_{i,j}^{(l)} = w_{i,j}^{(l)} - \lambda \Delta w_{i,j}^{(l)}$

To derive the spike error $\delta_k^{(l,j)}$ BATS splits it into two different error sources: inter-neuron and intra-neuron errors. Inter-neuron errors are generated from the dependencies between post-synaptic and pre-synaptic spikes because of the synaptic connections. Intra-neuron errors are generated due to the membrane potential reset mechanism after firing.

$$\delta_k^{(l,j)} := \frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}}$$
$$= \underbrace{\sum_{i=1}^{N^{(l+1)}} \sum_{z=1}^{n^{(l+1,i)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l+1,i)}} \frac{\partial t_z^{(l+1,i)}}{\partial t_k^{(l,j)}}}_{\text{iner-neuron error}} + \underbrace{\sum_{z=k+1}^{n^{(l,j)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l,j)}} \frac{\partial t_z^{(l,j)}}{\partial t_k^{(l,j)}}}_{\text{intra-neuron error}}$$
$$= \phi_k^{((l,j))} + \mu_k^{(l,j)} \tag{13}$$

Where $\phi_k^{((l,j))}$ is inter-neuron error and $\mu_k^{(l,j)}$ represents intra-neuron error.

The rest of the derivation of the closed form error has been omitted for conciseness. For more information on the specifics of the derivations consult the BATS paper [12]. However, note that the inter-neuron error is different for the output layer. More specifically:

$$\phi_k^{(o,j)} := \frac{\partial \mathcal{L}}{\partial n^{(o,j)}} = y_j - n^{(o,j)}$$

3 Adapting BATS to discrete time

This section describes our contribution to the model including the approach to time discretization, the changes to the backpropagation method and the evaluation metrics.

3.1 Time discretization

As shown in Equation 8 the spiking time has a closed-form solution which allows us to calculate the precise spike timing. Because of that in the implementation of the model, an event-driven mechanism is used: computations happen only when a spike is generated from a neuron. This allows the network to have low latency and improves its power efficiency. However, due to the requirement to derive the precise spike timing moving this model to a neuromorphic chip can present increased numerical errors. We will maintain the event-driven approach to adapt the model to discrete-time and delay the spike timings to match the discrete times based on a set step size. We choose this approach for two reasons. Firstly, adhering to the event-driven approach facilitates efficient training of diverse networks with adaptable step sizes. Secondly, modifying the fundamental methodology of the model can result in network instability and the introduction of errors, which complicate the traceability of time discretization effects. This consistency with the original model allows isolating the impact of time discretization.

Assume $t_k^{(l,j)}$ the time of the k-th spike of neuron j-th from layer l. The discretized time spike equivalent to $t_k^{(l,j)}$ has the following property :

$$0 \equiv \tilde{t}_k^{(l,j)} \pmod{\Delta t}$$

Where Δt is the discrete time step.

In a discrete environment, spikes cannot occur **before** their corresponding continuous events. While a neuron might miss a spike at the exact continuous time, it can fire it after a delay that aligns with the discrete spike timing. Because of that $\tilde{t}_k^{(l,j)} \geq t_k^{(l,j)}$. More explicitly:

$$\tilde{t}_k^{(l,j)} = t_k^{(l,j)} + \Delta t - \operatorname{mod}\left(t_k^{(l,j)}, \Delta t\right)$$

Consider the spike time discretizing function

$$d(t_k^{(l,j)}) = t_k^{(l,j)} + \Delta t - \operatorname{mod}\left(t_k^{(l,j)}, \Delta t\right) = \tilde{t}_k^{(l,j)} \quad (14)$$

This function is non-differentiable with respect to $t_k^{(l,j)}$ which means the derivatives with respect to the spike timing in the original BATS model should be recalculated for the discrete spike times. To avoid recalculation we can use methods from deep learning that allow backpropagation on a non-differentiable function.

3.2 Straight-Through-Estimator

In deep learning there has been a lot of research done on different ways to counter the non-derivability of activation functions when doing backpropagation either through surrogate gradients or different biased estimators [17] [18]. For our experiments, which utilize a network with one hidden layer, a straightforward and effective method is to employ a straight-through estimator[18]. The straight-through estimator is a biased estimator that has been shown to work extremely efficiently on networks with a small number of layers [18]. We will use it to approximate the non-differentiable time spike discretizing function, Equation 14, as an identity function during backpropagation.

$$\frac{\partial \tilde{t}_k^{(l,j)}}{\partial t_k^{(l,j)}} \approx 1 \quad (15)$$

$$\frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}} \approx \frac{\partial \mathcal{L}}{\partial \tilde{t}_k^{(l,j)}} \cdot \frac{\partial \tilde{t}_k^{(l,j)}}{\partial t_k^{(l,j)}}$$

Equation 15 simplifies this to :

$$\frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}} \approx \frac{\partial \mathcal{L}}{\partial \tilde{t}_k^{(l,j)}} \quad (16)$$

The estimator will be applied as follows. First, the discrete spike times are used in the forward pass to calculate the output and propagate the post-synaptic effects. During the backwards pass, the non-discretized spike times are utilized to compute the gradients.

3.3 Evaluation Metrics

As stated in the introduction the main focus of this research is to analyse the effects of time discretization on the efficiency of the model presented in 2.

The first metric is *accuracy*. While we look at the power efficiency of a model we can not overlook the change in accuracy as we want to preserve state-of-the-art performance as we want the resulting SNNs to be usable. Accuracy will be measured using a test dataset that will be separate from the training dataset since we want to avoid overfitting on a specific set and we want to test the inference power of the model on data that it has not processed before.

The second metric is *sparsity*. Sparsity in the context of a neural network directly correlates to how many inactive neurons are in the network. When it comes to efficiency in an SNN we know computations happen when spikes happen. Because of that by achieving high sparsity and maintaining a high level of accuracy, we can perform fewer calculations while having similar performance. For our research, we will look at the average spike count per neuron in the hidden and output layers as a measure of sparsity, with a lower spike count being more beneficial. Average spike count is calculated as $\frac{\text{spikes per neuron}}{\text{number of neurons in layer}}$.

The last metric is *prediction latency*. Prediction latency refers to the network's ability to make predictions quickly. Latency is important for power efficiency for several reasons. Firstly, resolving a task more quickly allows the network to enter an idle state sooner conserving power. Secondly, since the model used for this research relies on temporal coding, low latency allows for transmitting important information through earlier spikes. This can reduce the need for continued spiking to reinforce the signal and will increase sparsity. We calculate the prediction latency for the true class as $\frac{\text{number of spikes for the output label}}{\text{target count of spikes}}$

4 Experimental Setup and Results

The following section outlines the experimental setup, the metrics, and how the results were gathered. After each dataset, there will be a discussion on the results.

4.1 Experimental Setup

Several experiments have been conducted to evaluate the errors introduced by time discretization, focusing on the training process. The specific hyperparameters have been tuned in the BATS paper [12] and have not been altered for two reasons. Firstly this approach can provide a fair comparison between the discrete time model and the original. Secondly, it was observed through experiments that the network is extremely sensible to hyperparameter changes specifically in the target spike count, the distribution of the weight initialization and the output thresholds for each layer.

The Δt choices for the experiment have been selected considering the simulation time of MNIST datasets which in our case is 0.2 seconds, with all input spikes being in range [0, 0.1](s). For every metric, the results will be averaged over the batches.

4.2 MNIST

The first set of experiments has been performed on the MNIST[19] dataset as it is one of the most commonly used image classification tasks for benchmarking machine learning models. The MNIST dataset contains 60000 training samples and 10000 test samples. All the experiments consisted of a fully connected SNN with a layer configuration of 784-800-10 neurons. A mini-batch gradient descent approach was used with a training batch size of 50 and a test batch size of

100. The learning rate for the gradient descent has been set to 0.003, and the Adam optimizer has been used to perform the gradient descent. The target spike count for the true class has been set to 15, and for the false class, 3. τ_s has been set to 0.13 for both hidden and output layers and the threshold to 0.2 and 1.3 for the output layer and the hidden layer respectively. The weight distribution is U(-1, 1) as in the original paper. Five experiments were conducted over 10 epochs to evaluate sparsity and accuracy. Three experiments with 2 epochs were performed to measure latency, given the stability of the network. The average and standard deviation of the results are presented in Figures 1 - 4.

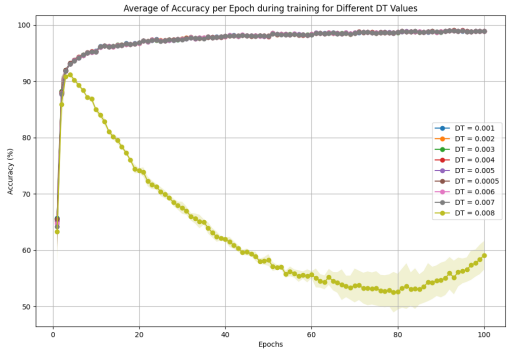

Figure 1: Accuracy% on training data for different Δt on the MNIST dataset

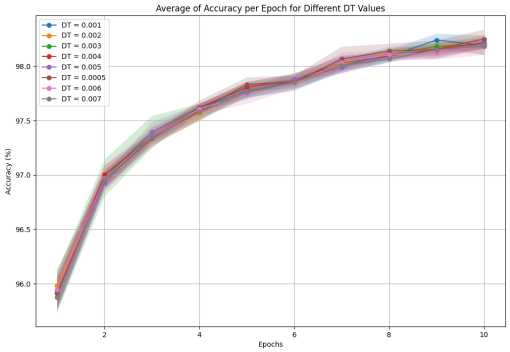

Figure 2: Accuracy% on test data for different Δt on the MNIST dataset

For a time step (Δt) size value larger or equal to 0.009 the network is not learning, and accuracy constantly decreases over epochs. On the contrary, for a Δt value below 0.008, accuracy remains stable for training and test batches, but sparsity is affected. This result hints that choosing a suitable timestep size can increase sparsity while maintaining a high accuracy. Regarding latency, the prediction confidence for the network trained using Δt = 0.0005 is dominant, implying that the latency is higher with a larger timestep.

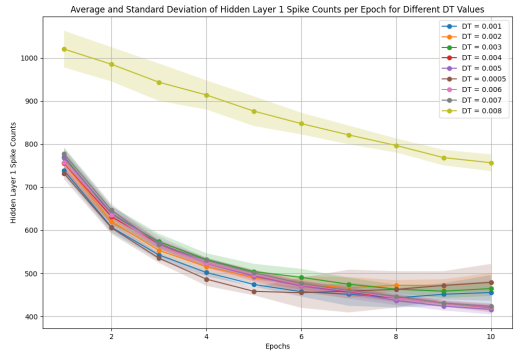

Figure 3: Spike count in hidden layer for different Δt on the MNIST dataset

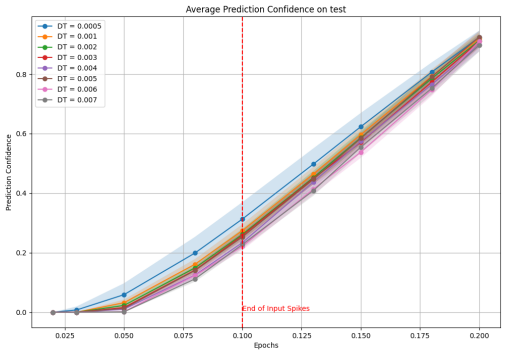

Figure 4: Prediction confidence over time for different Δt on the MNIST dataset

4.3 EMNIST

Introduced in 2017 EMNIST[20] is an extension of MNIST that includes a broader range of handwritten characters adding letters totalling 62 different classes. We will be using Balanced EMNIST which contains around 131000 samples.

To train a network for EMNIST, we will use the same hyperparameter values as for MNIST, as both datasets have been similarly used in BATS. This is logical given their similarities, with the primary difference being the input processing and the samples available in each dataset. The network architectures differ in the number of output layer neurons as EMNIST has more classes than MNIST. For the following experiments, we have also integrated the time continuous BATS model tagged as 'DT = 0' in the Figures.

Figure 6 shows that the continuous time network has the highest accuracy on training and test datasets. The difference between the best-performing and worst-performing Δt that converges is of $\approx$ 2% accuracy. However, it is not the most sparse network as shown in Figures 7. For Δt values larger than 0.008 the network starts losing in accuracy but does learn, in contrast to the previous dataset. Prediction latency does not seem to be affected by the time step size.

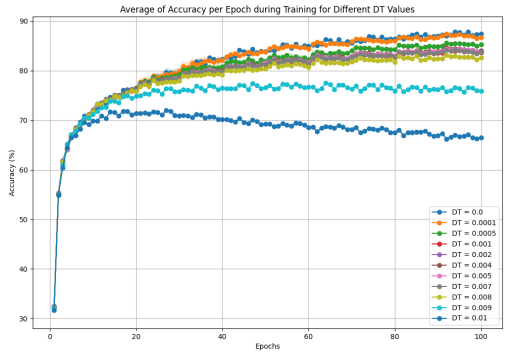

Figure 5: Accuracy% on training data for different Δt on the EMNIST dataset

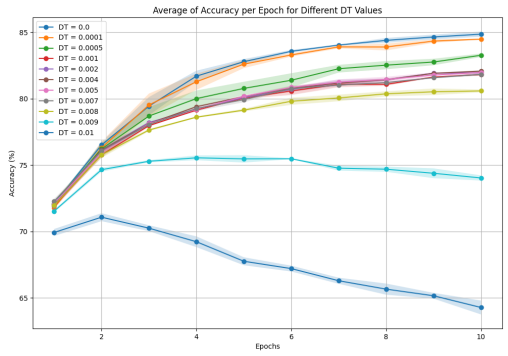

Figure 6: Accuracy% on test data for different Δt on the EMNIST dataset

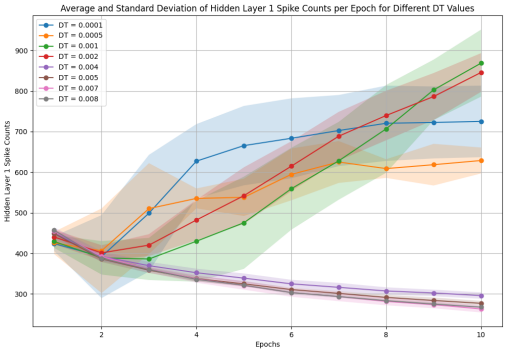

Figure 7: Spike count in hidden layer for different Δt on the EMNIST dataset

4.4 Fashion MNIST

The last dataset used is Fashion MNIST[21], a dataset that contains 28x28 grayscale images of 70,000 fashion products from 10 categories. Since this dataset represents a more difficult image recognition task, a more complex network architecture has been proposed in BATS. We will be using a 784x400x400x10 network. The training batch size is 5 and the test batch size is 100. The learning rate has been set to 0.005, with a decay factor of 0.5 every 10 epochs. The target spike count for the true class has been set to 15, and for the false class, 3. τ_s has been set to 0.13 for both the hidden layers and output layer. The threshold has been set to 0.13 and 0.45 for the hidden layers and 0.7 for the output layer. The weight distribution is U(-1, 1) as in the original paper. Two experiments were conducted over 10 epochs to evaluate accuracy, sparsity and latency. The average and standard deviation of the results are presented in Figures 9 - 12.

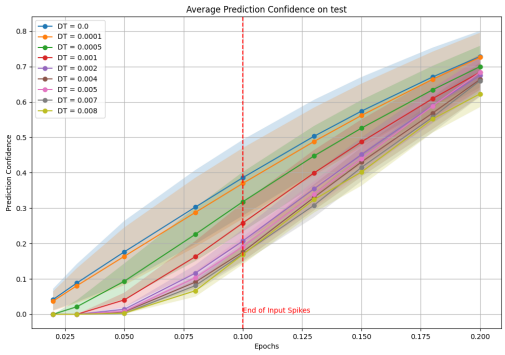

Figure 8: Prediction confidence over time for different Δt on the EMNIST dataset

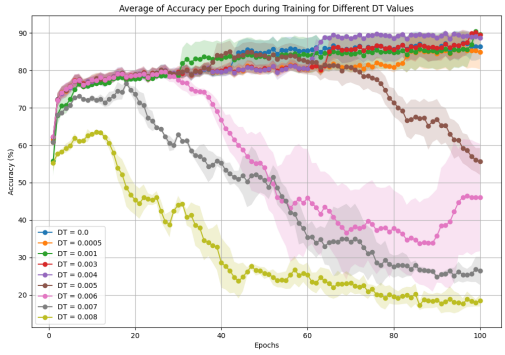

Figure 9: Accuracy% on training data for different Δt on the EMNIST dataset

The results indicate that the network accuracy decreases for $\Delta t = 0.005$ contrary to the other two datasets. The highest test accuracy is achieved for $\Delta t = 0.003$, but the difference between continuous BATS and this is negligible as seen from the standard deviation. Prediction latency seems to have a linear decrease with the time step size.

5 Discussion

The 3 datasets chosen are similar in the preprocessing of data and the input spikes which allows for a fair comparison be-

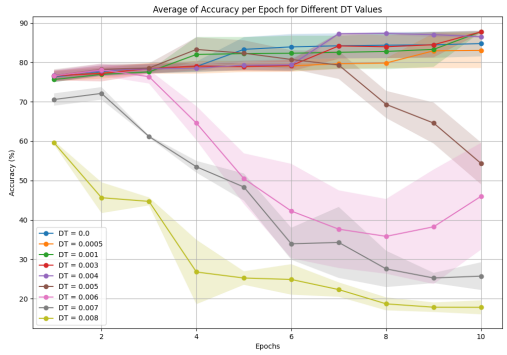

Figure 10: Accuracy% on test data for different Δt on the Fashion MNIST dataset

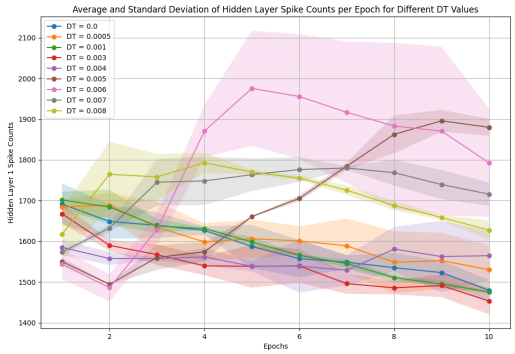

Figure 11: Spike count in hidden layer for different Δt on the Fashion MNIST dataset

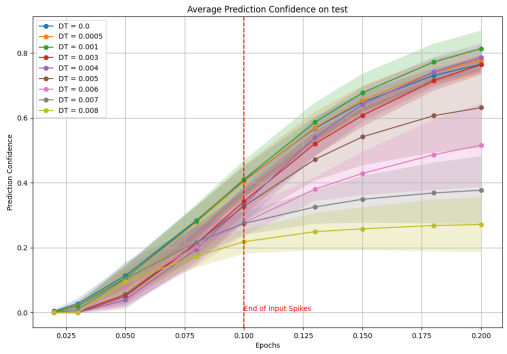

Figure 12: Prediction confidence over time for different Δt on the Fashion MNIST dataset

tween the performance of our networks on each of them. EMNIST is a more difficult version of MNIST that is being solved with the same network architecture and similar parameters. This allows us to generalize behaviour seen in the MNIST dataset. Due to its difficulty, Fashion MNIST is solved using a different network architecture with 2 hidden layers. This can be useful for our experiments as we would like to inspect the effects of multiple layers on the STE approach.

The initial hypothesis that sparsity can be improved by choosing a suitable timestep while not sacrificing accuracy has been verified by Figures 2 3 for MNIST, 6 7 for EMNIST and 10 11 for Fashion MNIST. This result suggests that power efficiency can be improved by setting a suitable timestep for the training process.

To further emphasize the sparsity-accuracy trade-off Figure 13 plots the accuracy of SNNs trained on different Δt values on the EMNIST dataset over the total average spike count in the hidden and output layers. This contains the points that form the Pareto Frontier, which, are points that do not have a better solution in *both* aspects. This plot does not answer the sparsity-accuracy trade-off as different settings can have different needs for the required spike count or lower bound of accepted accuracy. However, it provides a basis for our hypothesis that choosing a suitable timestep can improve or decrease this trade-off.

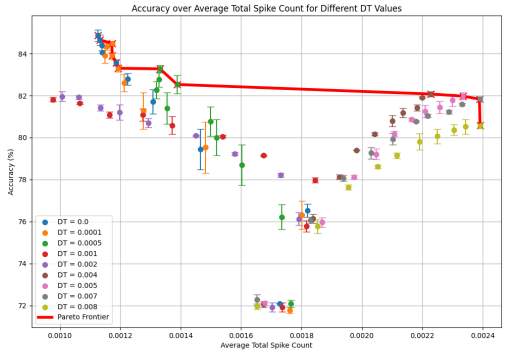

Figure 13: Accuracy % on the test set for EMNIST dataset over the inverse of the average total(hidden and output layers) spike count. The red line crosses the points on the Pareto Frontier.

On the other hand, setting the value of Δt too high, can lead to a decrease in accuracy making the network unable to learn, shown in Figures 1, 9. This can happen due to the following reasons. Firstly, the larger the timestep size the later the spikes happen and there is an information loss for spikes close to the end of the simulation time. This information loss depends on the behaviour of the network and how many spikes are close to the end of the time window. However, it should increase as the timestep is larger, for a highly populated input with late spikes. Secondly, the Straight-Through Estimator argues that the difference between continuous spikes and their corresponding discrete spike times is small. However, this difference increases on average as the timestep increases and numerical errors are introduced in the gradients that do not allow our network to learn. This happens because the delta of the gradients is inaccurate, leading to inconsistent weight changes in the network. This hypothesis can be verified by looking at the results for the Fashion MNIST dataset. The Straight-Through Estimator is biased.

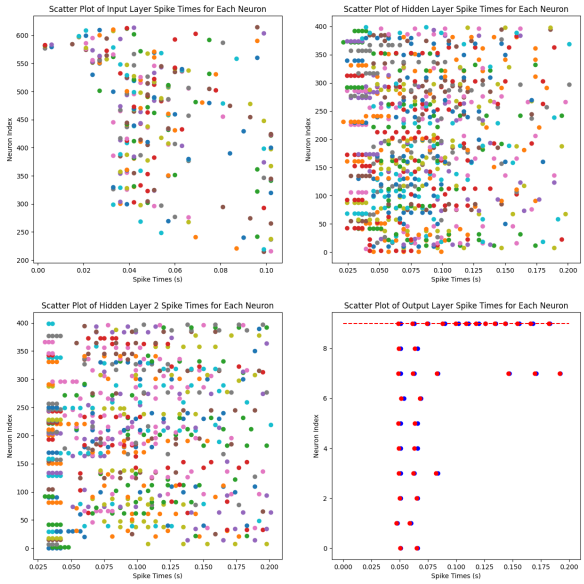

Figure 14: The spiking activity for an SNN trained on the Fashion MNIST dataset using $\Delta t = 0.003$ inferencing on one input image. In the output layer, the red horizontal dotted line marks the true label and the red dots mark continuous spikes while the blue ones mark discrete spikes.

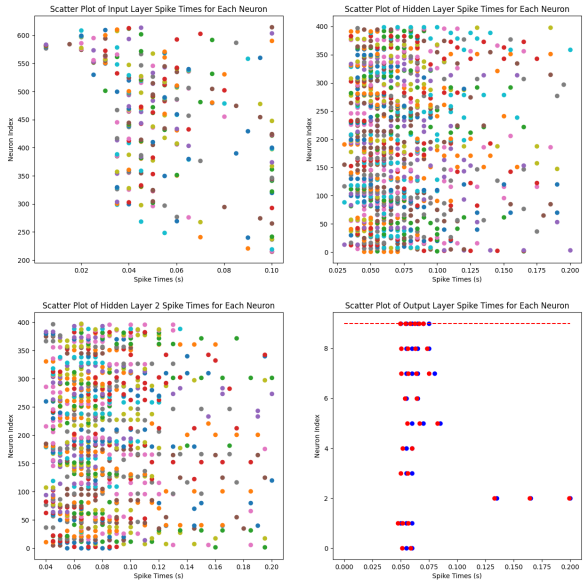

Figure 15: The spiking activity for an SNN trained on the Fashion MNIST dataset using $\Delta t = 0.005$ inferencing on one input image.

This suggests that the numerical errors increase as we introduce multiple layers in the network. This explains why for the Fashion MNIST dataset the upper bound for Δt where the network can learn is lower. Even if the spiking times in the input are in the same range as for the other two datasets and the behaviour in the hidden layers is similar (shown in Figures 14 15), the network is not able to learn for $\Delta t = 0.005$.

In all the experiments latency seems to have a direct correlation to the timestep size, as the bigger the timestep the more it takes for the network to reach a certain level of accuracy.

6 Conclusions and Future Work

This report proposes a hardware-aware training process through time discretization and analyses its effects on the power efficiency of a continuous time SNN model. More specifically we look at the BATS model [12] and modify its spike timings to match a discrete time step size. Using the Straight-through estimator we present a quick fix for the nonderivability of the time discretization function to pass the backpropagation step. An extensive analysis of SNNs trained with different time step size (Δt) values has been conducted using three image recognition datasets: MNIST, EMNIST and Fashion MNIST. The analysis shows that choosing a suitable time step can lead to more sparse networks that achieve a similar level of accuracy as the continuous time model. However, the time step size has an upper bound regarding its ability to converge to a high accuracy and choosing a large value can lead to numerical errors. This behaviour has been explained through issues with the biased estimator. This study represents a starting step in bridging the gap between software-implemented models and their counterparts on hardware devices. Below are some recommendations on the next steps regarding this topic.

Firstly, the datasets used in this analysis are all static and do not present a temporal dimension. Inspecting the performance of networks trained on different time step sizes for such a dataset would be useful. The SNN can not perform the task easily especially as the timestep size increases, because such datasets have intrinsic temporal patterns that are hard to detect. Some work has already been done to adapt the Spiking Heidelberg Digits [22] dataset to the discretized BATS framework within the project repository. However, a complex hyperparameter tuning process is required and due to its increasing difficulty, some data augmentation is recommended. Secondly, as explained in the Discussion section, some numerical errors are introduced due to the backpropagation approach. This implies that to enhance the performance of an event-driven discrete time model as the one used in this paper an extensive analysis on the impact of the backpropagation method needs to be conducted. Due to the errors introduced by the backpropagation, the effects of time discretization can not be fully isolated. Lastly, the spike timing derivative estimator is biased and relies on the difference between the discrete and continuous variables to be small. A surrogate gradient approach can approximate the derivative between the spike timings more accurately and will lower the gradient errors.

7 Responsible Research

In this section, we outline the aspects concerning responsibility in research such as the reproducibility of experiments, privacy and ethical issues regarding the data processed.

In terms of reproducibility, the repository that contains the implementation of the time discretization, as well as the experiment pipelines, is available on the Gitlab Server[1]hosted by TU Delft, with the possibility of uploading the work on a fully public repository. For each experiment, the cupy and numpy seeds have been saved allowing the experiments to be reproduced by running the training process using the same seeds for the initial weight distribution. The difference in batch selection should be negligible concerning the average outcomes. This research has been conducted on publicly available datasets that do not include any personal information. Training a neural network on these datasets does not present any privacy concerns.

However, this research addresses a significant societal issue. The increased computational costs of deep learning models pose a serious problem. The training emissions of GPT-3 are estimated to be equivalent to 305% of the carbon emissions of a full passenger jet flying between San Francisco and New York[23]. Therefore, it is becoming increasingly important to assess the energy efficiency of deep learning models alongside their accuracy.

In terms of writing, certain paragraphs have been augmented using a large language model for different writing purposes with prompts such as "Please reword this sentence [...]" or "Give me a synonym for [...]".

A Appendix

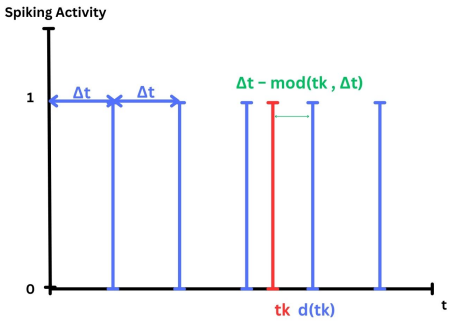

Figure 16: Visualization of the discretization function

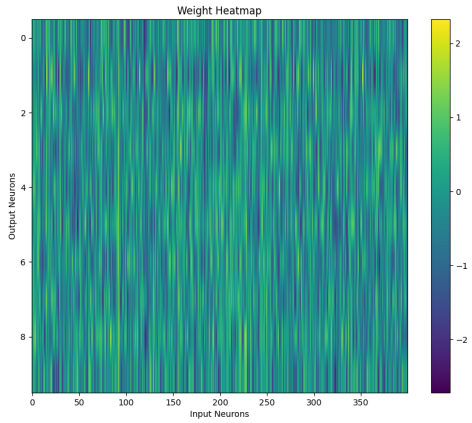

Figure 17: Heatmap of the weight values for the second hidden layer in a network trained with $\Delta t = 0.003$ for the Fashion MNIST dataset

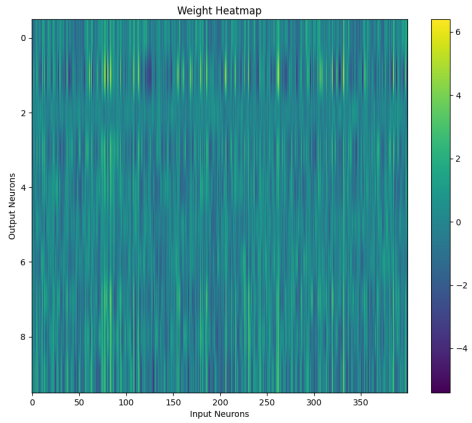

Figure 18: Heatmap of the weight values for the second hidden layer in a network trained with $\Delta t = 0.006$. The similar low values hint at vanishing gradient problems.

References

[1] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian,

[1]https://gitlab.ewi.tudelft.nl/cse3000/2023-2024-q4/Tomen_Micheli/agcojocaru-Adapting-unconstrained-spiking-neural-networks-to

Table 1: Hyperparameters for experiments on the datasets

Hyperparameter	**MNIST**	**EMNIST**	**Fashion MNIST**
Layer Configuration	784-800-10	784-800-47	784-400-400-10
Training Batch Size	50	50	5
Test Batch Size	100	100	100
Learning Rate	0.003	0.003	0.0005
Optimizer	Adam	Adam	Adam
Target Spike Count (True Class)	15	15	15
Target Spike Count (False Class)	3	3	3
τ_s (Hidden & Output Layers)	0.13	0.13	0.13
Threshold (Output Layer)	0.2	1.3	0.7
Threshold (Hidden Layer)	1.3	0.2	0.13
Threshold (Hidden Layer 2)	-	-	0.45
Weight Distribution	U(-1, 1)	U(-1, 1)	U(-1, 1)
Epochs (Accuracy & Sparsity)	10	10	10
Epochs (Latency)	10	10	10
Number of Experiments (Accuracy & Sparsity)	5	3	2
Number of Experiments (Latency)	3	3	2

J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, and et al., "Gpt-4 technical report," tech. rep., OpenAI, 2023.

[2] A. F. Wolff, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," *arXiv preprint*, 2020.

[3] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[4] M. Dampfhoffer, T. Mesquida, *et al.*, "Are snns really more energy-efficient than anns? an in-depth hardware-aware study," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 2, pp. 242–256, 2023.

[5] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. B. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[6] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[7] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, "The brainscales-2 accelerated neuromorphic system with hybrid plasticity," *Frontiers in Neuroscience*, vol. 16, p. 795876, 2022.

[8] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[9] J. Göltz, L. Kriener, A. Baumbach, *et al.*, "Fast and energy-efficient neuromorphic deep learning with first-spike times," *Nature Machine Intelligence*, vol. 3, pp. 823–835, 2021.

[10] N. Rathi and K. Roy, "Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 6, pp. 3174–3182, 2023.

[11] Q. Meng, M. Xiao, S. Yan, Y. Wang, Z. Lin, and Z.-Q. Luo, "Training high-performance low-latency spiking neural networks by differentiation on spike representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12444–12453, June 2022.

[12] F. Bacho and D. Chu, "Exploring tradeoffs in spiking neural networks," *Neural Computation*, vol. 35, no. 10, pp. 1627–1656, 2023.

[13] W. Zhang and P. Li, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[14] F. Liu, W. Zhao, Y. Chen, Z. Wang, T. Yang, and L. Jiang, "Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training," *Frontiers in Neuroscience*, vol. 15, p. 1413, 2021.

[15] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, pp. 1412–1421, 2018.

[16] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[17] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural Computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[18] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.

[21] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[22] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 2744–2757, 2022.

[23] D. Patterson, J. Gonzalez, Q. V. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *arXiv preprint arXiv:2104.10350*, 2021.