Towards High-performance Spiking Transformers from ANN to SNN Conversion

MM '24: Proceedings of the 32nd ACM International Conference on Multimedia

https://github.com/h-z-h-cell/Transformer-to-SNN-ECMT

Zihan Huang
Peking University
Beijing, China
hzh@stu.pku.edu.cn

Xinyu Shi
Peking University
Beijing, China
xyshi@pku.edu.cn

Zecheng Hao
Peking University
Beijing, China
1900012989@pku.edu.cn

Tong Bu
Peking University
Beijing, China
putong30@pku.edu.cn

Jianhao Ding
Peking University
Beijing, China
djh01998@stu.pku.edu.cn

Zhaofei Yu*
Peking University
Beijing, China
yuzf12@pku.edu.cn

Tiejun Huang
Peking University
Beijing, China
tjhuang@pku.edu.cn

Towards High-performance Spiking Transformers from ANN to SNN Conversion

Zihan Huang
Peking University
Beijing, China

Xinyu Shi
Peking University
Beijing, China

Zecheng Hao
Peking University
Beijing, China

Tong Bu
Peking University
Beijing, China

Jianhao Ding
Peking University
Beijing, China

Zhaofei Yu*
Peking University
Beijing, China

Tiejun Huang
Peking University
Beijing, China

Abstract

CCS Concepts

Keywords

Abstract

Spiking neural networks (SNNs) show great potential due to their energy efficiency, fast processing capabilities, and robustness. There are two main approaches to constructing SNNs. Direct training methods require much memory, while conversion methods offer a simpler and more efficient option. However, current conversion methods mainly focus on converting convolutional neural networks (CNNs) to SNNs. Converting Transformers to SNN is challenging because of the presence of non-linear modules. In this paper, we propose an Expectation Compensation Module to preserve the accuracy of the conversion. The core idea is to use information from the previous T time-steps to calculate the expected output at time-step T. We also propose a Multi-Threshold Neuron and the corresponding Parallel Parameter normalization to address the challenge of large time steps needed for high accuracy, aiming to reduce network latency and power consumption. Our experimental results demonstrate that our approach achieves state-of-the-art performance. For example, we achieve a top-1 accuracy of 88.60% with only a 1% loss in accuracy using 4 time steps while consuming only 35% of the original power of the Transformer. To our knowledge, this is the first successful Artificial Neural Network (ANN) to SNN conversion for Spiking Transformers that achieves high accuracy, low latency, and low power consumption on complex datasets. The source codes of the proposed method are available at https://github.com/h-z-h-cell/Transformer-to-SNN-ECMT.

摘要：

1.SNN有诸多优势
2.构建SNN的两种方法，各有利弊（主说Conversion弊端）

- **Transformer Conversion的挑战：**
  非线性层/模块的存在

3.本文提出的方法1：Expectation Compensation Module
用前T个时间步计算第T个时间的期望输出
4.本文提出的方法2：Multi-Threshold Neuron
5.本文提出的方法3：Parallel Parameter Normalization
效果：减少SNN推理延迟和能耗
6.实验结果……
7. ?

Introduction

Para.1

1. What is SNN?
2. Spike-based neuron dynamics: **Sparse**
3. SNN is hard to train.

Para.2

Training Strategies:

1. Direct training: surrogate gradient or STDP
2. ANN2SNN: similarity between ANN ReLU(·) and firing rate;

More timesteps, More Precision

Para.3

Transformers are powerful, but no previous ANN2SNN works on such model.

- LayerNorm & GELU 👉require interaction between neurons **in the same layer.**
- Hard to apply piecewise quantization of individual neurons **?**
- Non-linear characteristics

Ideally, layerwise neurons are I.I.D maybe?

Introduction

Para.4

New Methods:

1.Expectation Compensation Module(ECM):

2.Multi-Threshold Neurons & Parallel Parameter normalization: **Irrelevant?**

Para.5

Summary of Contributions:

Related Works

Sec.1: ANN-SNN Conversion

1. Make output values in all layers positive:
   a. Add an abs() function (absolution value) layer after preprocessing phase that includes color transformation and spatial normalization. The abs() function ensures that the input values to the first convolution layer are all non-negative. This step can be omitted if the preprocessing method itself gives positive output (e.g., the original RGB values are used);
   b. Change the sigmoid activation function (after convolution) from tanh() to HalfRect(). HalfRect(x), also referred to as rectified linear units (ReLU), is defined as:

$$\text{HalfRect}(x) = \max(x, 0)$$

2015 Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition.

Algorithm 1: Model-Based Normalization

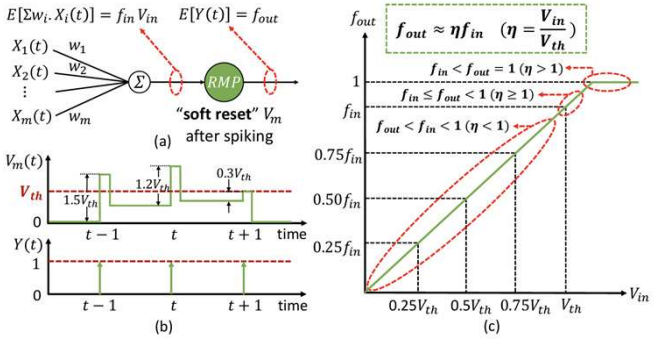
```
for layer in layers:
    max_pos_input = 0
    # Find maximum input for this layer
    for neuron in layer.neurons:
        input_sum = 0
        for input_wt in neuron.input_wts:
            input_sum += max(0, input_wt)
        max_pos_input = max(max_pos_input, input_sum)
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / max_pos_input
```


Algorithm 2: Data-Based Normalization

```
previous_factor = 1
for layer in layers:
    max_wt = 0
    max_act = 0
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            max_wt = max(max_wt, input_wt)
        max_act = max(max_act, neuron.output_act)
    scale_factor = max(max_wt, max_act)
    applied_factor = scale_factor / previous_factor
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / applied_factor
    previous_factor = scale_factor
```


2015 Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing

2017 Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification



$$r_i^1(t) = \begin{cases} a_i^1 r_{\max} \cdot \dfrac{V_{\text{thr}}}{V_{\text{thr}} + \epsilon_i^1} - \dfrac{V_i^1(t)}{t \cdot (V_{\text{thr}} + \epsilon_i^l)} & \text{reset to zero} \quad (5a) \\ a_i^1 r_{\max} - \dfrac{V_i^1(t)}{t \cdot V_{\text{thr}}} & \text{reset by subtraction.} \quad (5b) \end{cases}$$

2020 RMP-SNN: Residual Membrane Potential Neuron for Enabling Deeper High-Accuracy and Low-Latency Spiking Neural Network

I. Optimizing thresholds

2019 Going Deeper in Spiking Neural Networks: VGG and Residual Architectures

Algorithm 1: SPIKE-NORM

input : Input Poisson Spike Train *spikes*, Number of Time-Steps *#timesteps*
output: Weight-normalization / Threshold-balancing factors $v_{th,norm}[i]$ for each neural layer (*net.layer*[*i*]) of the network *net*

initialization $v_{th,norm}[i] = 0 \; \forall \; i = 1, ..., \#net.layer$;
// Set input of 1st layer equal to spike train
net.layer[1].*input* = *spikes*;
for $i \leftarrow 1$ **to** *#net.layer* **do**
  for $t \leftarrow 1$ **to** *#timesteps* **do**
    // Forward pass spike-train for neuron layer-*i* characterized by membrane potential *net.layer*[*i*].v_{mem} and threshold *net.layer*[*i*].v_{th}
    net.layer[*i*] : *forward*(*net.layer*[*i*].*input*[*t*]);
    // Determine threshold-balancing factor according to maximum SNN activation, max (*net.layer*[*i*].*weight* ∗ *net.layer*[*i*].*input*[*t*]) , where '∗' represents the dot-product operation
    $v_{th,norm}[i] = \max(v_{th,norm}[i], \max(net.layer[i].weight * net.layer[i].input[t]))$;
  end
  // Threshold-balance layer-*i*
  net.layer[*i*].$v_{th} = v_{th,norm}[i]$;
  // Record input spike-train for next layer
  net.layer[*i* + 1].*input* = *net.layer*[*i*] : *forward*(*net.layer*[*i*].*input*);
end

Algorithm 1 SATP

Input: Inference Images $\mathcal{X} = \{x_i\}_{i=1}^n$.
Output: Predicted label $C = \{c\}_{i=1}^n$ for each image.
Set the parameters m, λ, σ and η, initialize V_{th};
for *time t in* $[1, 2, \ldots, T]$ **do**
  for *layer l in layers* **do**
    for *neuron p in layer.neurons* **do**
      Calculate the input current $I(t)$ by Eq. (4);
      Calculate $\nabla v(t)$ by Eq. (2);
      if $V(t) > V_{th}(t)$ **then**
        Neuron p fires;
        Update the membrane potential $V(t)$ by Eq. (3);
        Calculate $\Delta V_{th}(t)$ by Eq. (7);
        Update $V_{th}(t) \leftarrow V_{th}(t) + \Delta V_{th}(t)$.
      end
    end
  end
  Collect the spikes from the output layer;
  Obtain the classification result at time t.
end

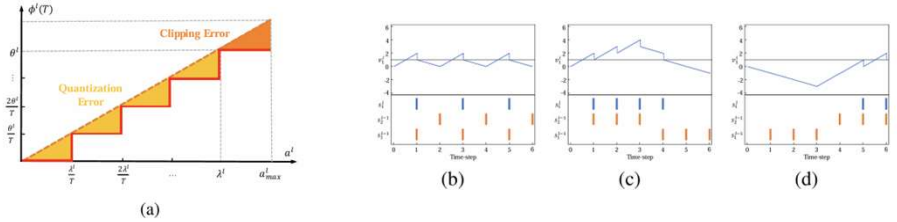
2023 Low Latency and Sparse Computing Spiking Neural Networks With Self-Driven Adaptive Threshold Plasticity

Related Works

Sec.1: ANN-SNN Conversion

II. Optimizing membrane potentials

2022 Optimized Potential Initialization for Low-latency Spiking Neural Networks



Optimal initialization of membrane potentials

Theorem 1. *The expectation of square conversion error (Eq. (13)) reaches the minimum value when the initial value* $\boldsymbol{v}^l(0)$ *is* $\boldsymbol{V}_{th}^l/2$*, meanwhile the expectation of conversion error reaches* 0*, that is:*

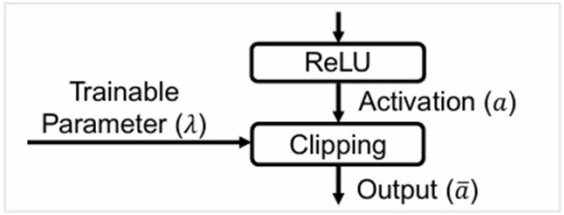
$$\arg\min_{\boldsymbol{v}^l(0)} E_{\boldsymbol{z}} \left\| f(\boldsymbol{z}) - f'(\boldsymbol{z}) \right\|_2^2 = \frac{\boldsymbol{V}_{th}^l}{2}, \tag{14}$$

$$E_{\boldsymbol{z}} \left(f(\boldsymbol{z}) - f'(\boldsymbol{z}) \right) \Big|_{\boldsymbol{v}^l(0)=\frac{\boldsymbol{V}_{th}^l}{2}} = \boldsymbol{0}. \tag{15}$$

2023 Reducing ANN-SNN Conversion Error through Residual Membrane Potential

III. Optimizing pre-conversion ANN structure

2021 TCL: an ANN-to-SNN Conversion with Trainable Clipping Layers

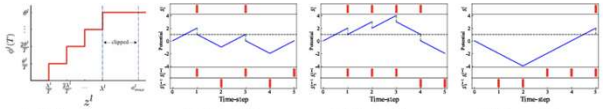


Rate Norm Layer

$$\theta_l = p_l \cdot \max(W_{l-1}\hat{\boldsymbol{r}}_{l-1} + \boldsymbol{b}_{l-1}),$$
$$\boldsymbol{z}_l = \text{clip}(W_{l-1}\hat{\boldsymbol{r}}_{l-1} + \boldsymbol{b}_{l-1}, 0, \theta_l),$$
$$\hat{\boldsymbol{r}}_l = \frac{\boldsymbol{z}_l}{\theta_l},$$

2021 Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks

2022 Optimal ANN-SNN Conversion for High-accuracy and Ultra-low-latency Spiking Neural Networks

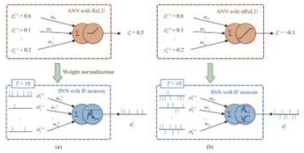


4.2 QUANTIZATION CLIP-FLOOR-SHIFT ACTIVATION FUNCTION

We propose the quantization clip-floor-shift activation function to train ANNs.

$$\boldsymbol{a}^l = \widehat{h}(\boldsymbol{z}^l) = \lambda^l \, \text{clip}\left(\frac{1}{L}\left\lfloor \frac{\boldsymbol{z}^l L}{\lambda^l} + \boldsymbol{\varphi} \right\rfloor, 0, 1\right).$$

2023 Symmetric-threshold ReLU for Fast and Nearly Lossless ANN-SNN Conversion



$$\boldsymbol{z}^l = \text{stReLU}(\boldsymbol{W}^l \boldsymbol{z}^{l-1} + \boldsymbol{b}^l)$$

$$\text{stReLU}(x) = \begin{cases} V_{th}, & \text{if } x \geq V_{th} \\ x, & \text{if } -V_{th} < x < V_{th} \\ -V_{th}, & \text{if } x \leq -V_{th}. \end{cases}$$

Related Works

Sec.1: ANN-SNN Conversion

III. Optimizing pre-conversion ANN structure

2023 A Unified Optimization Framework of ANN-SNN Conversion: Towards Optimal Mapping from Activation Values to Firing Rates

Proposed SlipReLU Activation Function. Following the unified optimization framework outlined in Sect. 3.1, our proposed SlipReLU activation function aims to minimize the mismatch between the tailored ANN and the target SNN, while also minimizing deviation from the regular ReLU,

$$\begin{aligned}\mathrm{SlipReLU}(\mathbf{z}^{(\ell)}) = c\theta^{(\ell)}\mathrm{clip}\left(\frac{\mathbf{z}^{(\ell)}}{\theta^{(\ell)}} + \boldsymbol{\delta}_1, 0, 1\right) \\ + (1-c)\theta^{(\ell)}\mathrm{clip}\left(\frac{1}{N}\left\lfloor \frac{N\mathbf{z}^{(\ell)}}{\theta^{(\ell)}} + \boldsymbol{\delta}\right\rfloor, 0, 1\right).\end{aligned} \tag{8}$$

B.3. Special Cases of the SlipReLU Activation Function

Here we list four different special cases of the proposed SlipReLU.

Threshold-ReLU When $c = 1$ and $\delta_1 = 0$, the SlipReLU becomes the threshold ReLU activation function which is studied in Deng & Gu (2021).

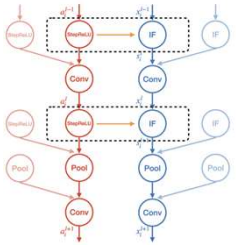
Shift-threshold-ReLU When $c = 1$ and $\delta_1 = -1/(2N)$, the SlipReLU becomes the shift-threshold ReLU activation function which is studied in Deng & Gu (2021).

Quantization clip-floor (QCF) When $c = 0$ and $\delta = 0$, the SlipReLU becomes the quantization clip-floor (QCF) activation function which is studied in Bu et al. (2021).

Quantization clip-floor-shift (QCFS) When $c = 0$ and $\delta = 1/2$, the SlipReLU becomes the quantization clip-floor-shift (QCFS) activation function which is studied in Bu et al. (2021).

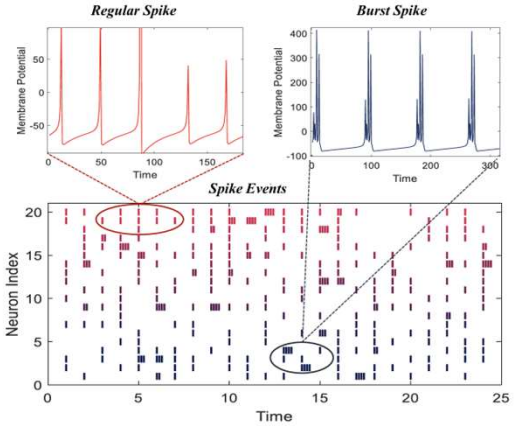
2023 A New ANN-SNN Conversion Method with High Accuracy, Low Latency and Good Robustness

$$\begin{aligned}\mathrm{StepReLU}(x, \gamma) &= \mathrm{ReLU}(x) + \gamma \\ &= \begin{cases} x + \gamma, & \text{if} \quad x > 0 \\ 0, & \text{if} \quad x \le 0 \end{cases}\end{aligned}$$



IV. Optimizing spiking neuronal models

2022 Efficient and Accurate Conversion of Spiking Neural Network with Burst Spikes



2022 Signed Neuron with Memory: Towards Simple, Accurate and High-Efficient ANN-SNNConversion

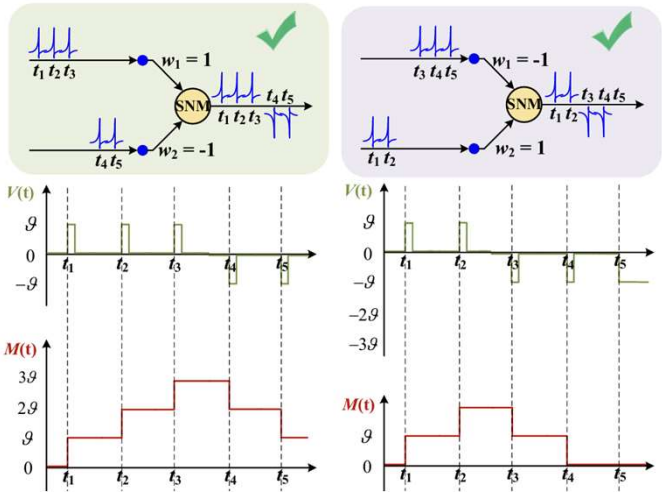
$$\tilde{v}_j^l(t) = v_j^l(t-1) + \sum_i w_{ij} s_i^{l-1}(t) + b_j, \tag{7}$$

$$\tilde{m}_j^l(t) = m_j^l(t-1), \tag{8}$$

$$s_j^l(t) = \begin{cases} \theta_j^l, & \tilde{v}_j^l(t) \ge \theta_j^l \\ -\theta_j^l, & \tilde{v}_j^l(t) \le -\theta_j^l \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \tilde{m}_j^l(t) > 0\,, \tag{9}$$

$$v_j^l(t) = \tilde{v}_j^l(t) - s_j^l(t), \tag{10}$$

$$m_j^l(t) = \tilde{m}_j^l(t) + s_j^l(t), \tag{11}$$

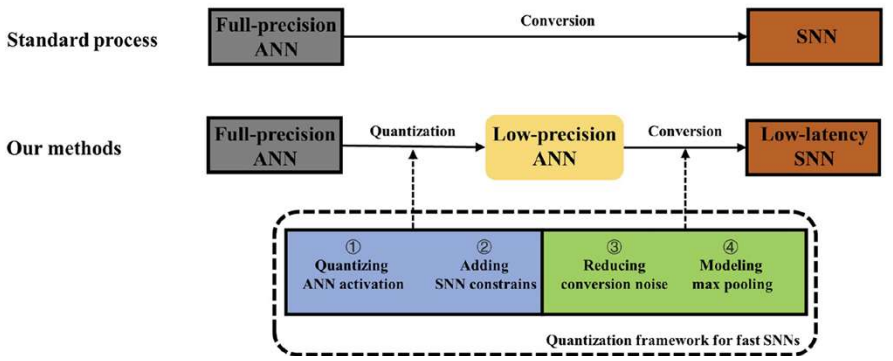


Related Works

Sec.1: ANN-SNN Conversion

IV. Optimizing spiking neuronal models

2022 Quantization Framework for Fast Spiking Neural Networks



V. Others:

*2024 Spatio-Temporal Approximation: A Training-Free SNN Conversion for **Transformers***

Definition 1 (Universal Group Operator). *Let $f : x \mapsto y$ defined on domain $x \in \mathcal{D}$ be a real continuous unary function. Its spiking universal group operator $\hat{f}$ comprises two fully connected (FC) layers surrounding a single hidden IF layer with N neurons, such that $\exists \epsilon > 0$ where for any spike input $\boldsymbol{x}_s$ with mean $\bar{\boldsymbol{x}}_s = x$, the output spikes $\boldsymbol{y}_s$ satisfy $\mathbb{E}\,|\bar{\boldsymbol{y}}_s - y| \leq \epsilon$.*

The input and output layers have weights $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^n$, and biases $\boldsymbol{b}_1 \in \mathbb{R}^n, b_2 \in \mathbb{R}$, respectively.

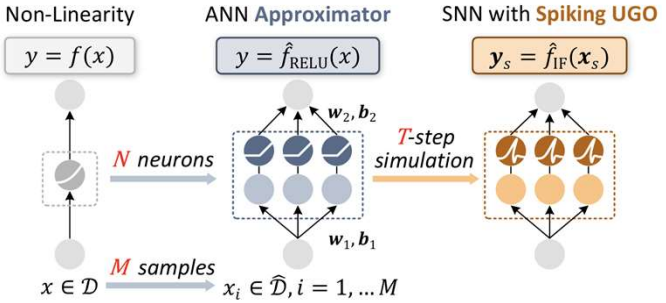


Figure 2: Spatial approximation process with UGO.

Sec.2: Direct Training for Transformer

Preliminaries

Sec.1: Theories of ANN-SNN Conversion

ANN Neuron: $$\boldsymbol{a}^l = \mathrm{ReLU}(\boldsymbol{W}^l \boldsymbol{a}^{l-1}) = \max(\boldsymbol{W}^l \boldsymbol{a}^{l-1}, 0),$$

IF Neuron:
$$\boldsymbol{m}^l(t) = \boldsymbol{v}^l(t-1) + \boldsymbol{W}^l \boldsymbol{x}^{l-1}(t),$$
$$\boldsymbol{s}^l(t) = H(\boldsymbol{m}^l(t) - \theta^l),$$
$$\boldsymbol{x}^l(t) = \theta^l \boldsymbol{s}^l(t),$$
$$\boldsymbol{v}^l(t) = \boldsymbol{m}^l(t) - \boldsymbol{x}^l(t). \quad \textit{reset-by-subtraction}$$

Recurrent IF Neuron: $$\boldsymbol{v}^l(t) - \boldsymbol{v}^l(t-1) = \boldsymbol{W}^l \boldsymbol{x}^{l-1}(t) - \boldsymbol{x}^l(t).$$

Recurrent IF Neuron summed through *T*:
$$\boldsymbol{v}^l(t) - \boldsymbol{v}^l(t-1) = \boldsymbol{W}^l \boldsymbol{x}^{l-1}(t) - \boldsymbol{x}^l(t).$$
$$\boldsymbol{v}^l(t-1) - \boldsymbol{v}^l(t-2) = \boldsymbol{W}^l \boldsymbol{x}^{l-1}(t-1) - \boldsymbol{x}^l(t-2).$$
$$\cdots\cdots$$
$$\frac{\boldsymbol{v}^l(T) - \boldsymbol{v}^l(0)}{T} = \frac{\boldsymbol{W}^l \sum_{i=1}^{T} \boldsymbol{x}^{l-1}(i)}{T} - \frac{\sum_{i=1}^{T} \boldsymbol{x}^l(i)}{T}.$$

Preliminaries

Sec.1: Theories of ANN-SNN Conversion

Recurrent IF Neuron summed through *T*:

$$\frac{v^l(T)-v^l(0)}{T}=\frac{W^l\sum_{i=1}^{T}x^{l-1}(i)}{T}-\frac{\sum_{i=1}^{T}x^l(i)}{T}.$$

$$\Phi^l(T)=\frac{\sum_{i=1}^{T}x^l(i)}{T}$$

$$\Phi^l(T)=W^l\Phi^{l-1}(T)-\frac{v^l(T)-v^l(0)}{T}.$$

Parameter normalization

$$W^l_{\mathrm{SNN}}=W^l_{\mathrm{ANN}}\frac{\lambda^{l-1}}{\lambda^l}.$$

Algorithm 1: Model-Based Normalization

```
for layer in layers:
    max_pos_input = 0
    # Find maximum input for this layer
    for neuron in layer.neurons:
        input_sum = 0
        for input_wt in neuron.input_wts:
            input_sum += max(0, input_wt)
        max_pos_input = max(max_pos_input, input_sum)
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / max_pos_input
```

Access all weights

Sum of weights(ignore neg. value)

Update *max_pos_input*

Algorithm 2: Data-Based Normalization

```
previous_factor = 1
for layer in layers:
    max_wt = 0
    max_act = 0
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            max_wt = max(max_wt, input_wt)
        max_act = max(max_act, neuron.output_act)
    scale_factor = max(max_wt, max_act)
    applied_factor = scale_factor / previous_factor
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / applied_factor
    previous_factor = scale_factor
```

Modifying Equation(9) and setting θ^l_j to 1 is equivalent to adjusting the firing threshold on the soft-reset neuron to λ^l

Methods

Sec.1 Error Analysis of Nonlinear Module in ANN-SNN Conversion

GeLU Softmax LN Attention matrix product?

assume that the outputs of layer $l-1$ in both ANNs and SNNs are identical, $a^{l-1} = \Phi^{l-1}(T) = \frac{\sum_{t=1}^{T} \boldsymbol{x}^{l-1}(t)}{T}$

Compare the outputs of layer l : a^l and Φ^l

An arbitrary non-linear module $F(\cdot)$, in ANN $a^l = F(a^{l-1}),$

The same arbitrary non-linear module $F(\cdot)$, in counterpart SNN $x^l(t) = F(x^{l-1}(t))$

Average output (0-T): $\Phi^l(T) = \frac{\sum_{t=1}^{T} x^l(t)}{T} = \frac{\sum_{t=1}^{T} F(x^{l-1}(t))}{T}.$

output in ANN: $a^l = F(a^{l-1}) = F\left(\frac{\sum_{t=1}^{T} x^{l-1}(t)}{T}\right)$

$$\frac{\sum_{t=1}^{T} F(x^{l-1}(t))}{T} \neq F\left(\frac{\sum_{t=1}^{T} x^{l-1}(t)}{T}\right)$$

Methods

Sec.2 Expectation Compensation Module

$$\frac{\sum_{t=1}^{T} F(x^{l-1}(t))}{T} \neq F\left(\frac{\sum_{t=1}^{T} x^{l-1}(t)}{T}\right)$$

output in ANN: $a^l = F(a^{l-1}) = F\left(\frac{\sum_{t=1}^{T} x^{l-1}(t)}{T}\right)$

$$\boldsymbol{a}^l = \Phi^l(T) = \frac{\sum_{i=1}^{T} \boldsymbol{x}^l(i)}{T}$$

General Expectation Compensation Module.

THEOREM 4.1. *Consider a non-linear layer l with a function F. In SNNs, the output of this layer at time t is denoted as $O^l(t)$. Let $S^l(T)$ be the cumulative sum of layer l outputs up to time T, given by $S^l(T) = \sum_{t=1}^{T} O^l(t)$. The expected output of the SNNs at time T is given by:*

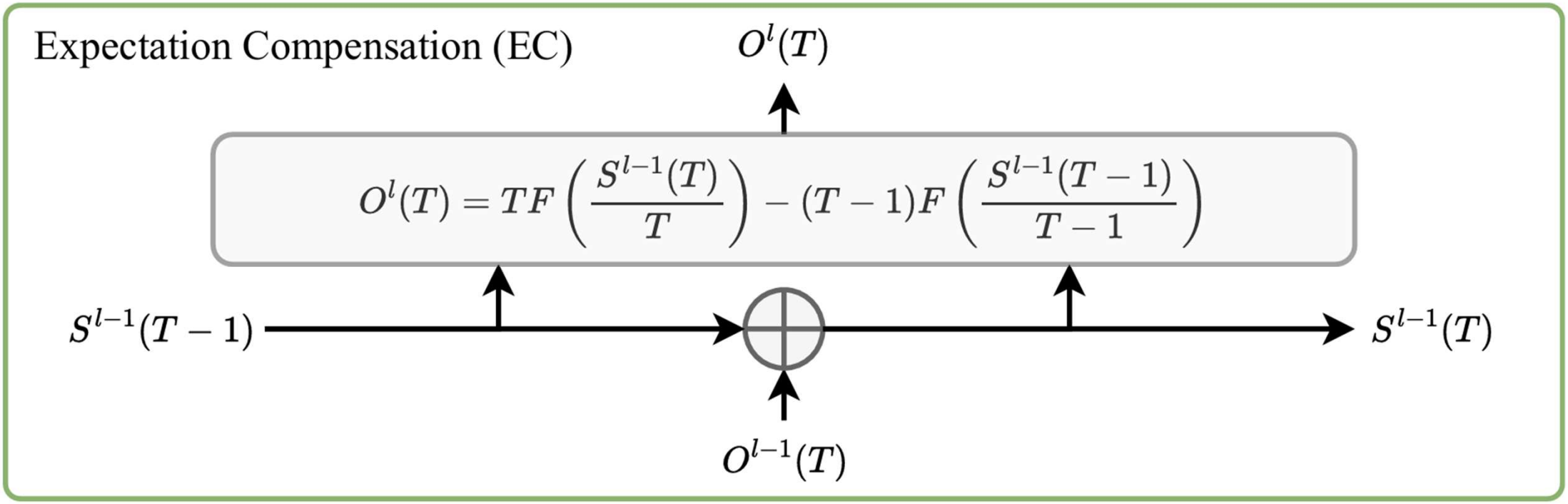
$$O^l(T) = TF\left(\frac{S^{l-1}(T)}{T}\right) - (T-1)F\left(\frac{S^{l-1}(T-1)}{T-1}\right). \quad (15)$$

$$\begin{aligned}
O^l(T) &= \sum_{t=1}^{T} O^l(t) - \sum_{t=1}^{T-1} O^l(t) \\
&= T\boldsymbol{a}_T^l - (T-1)\boldsymbol{a}_{T-1}^l \qquad T\boldsymbol{\phi}_T^l - (T-1)\boldsymbol{\phi}_{T-1}^l \\
&= TF(\boldsymbol{a}_T^{l-1}) - (T-1)F(\boldsymbol{a}_{T-1}^{l-1}) \\
&= TF\left(\frac{\sum_{t=1}^{T} O^{l-1}(t)}{T}\right) - (T-1)F\left(\frac{\sum_{t=1}^{T-1} O^{l-1}(t)}{T-1}\right) \\
&= TF\left(\frac{S^{l-1}(T)}{T}\right) - (T-1)F\left(\frac{S^{l-1}(T-1)}{T-1}\right)
\end{aligned}$$

Methods

Sec.2 Expectation Compensation Module

$$\boldsymbol{O}^{l}(T) = TF\left(\frac{\boldsymbol{S}^{l-1}(T)}{T}\right) - (T-1)F\left(\frac{\boldsymbol{S}^{l-1}(T-1)}{T-1}\right). \qquad (15)$$



Methods

Sec.2 Expectation Compensation Module

Expectation Compensation Module for Matrix Product.

THEOREM 4.2. *Consider a module for matrix product that receives two sets of spike inputs, denoted by $\boldsymbol{A}_{v_a}(t)$ and $\boldsymbol{B}_{v_b}(t)$. These inputs are generated by neurons A and B, respectively, and are characterized by multiple thresholds v_a and v_b, as described in Section 4.3.*

We can integrate the input by $\boldsymbol{A}(t) = \sum_{v_a} v_a \boldsymbol{A}_{v_a}(t)$ and $\boldsymbol{B}(t) = \sum_{v_b} v_b \boldsymbol{B}_{v_b}(t)$. Here, $\boldsymbol{A}(t)$ and $\boldsymbol{B}(t)$ are the sum matrices weighted by multiple thresholds v_a and v_b, respectively.

Let $\boldsymbol{S}_A(T) = \sum_{t=1}^{T} A(t)$ and $\boldsymbol{S}_B(T) = \sum_{t=1}^{T} B(t)$ represent the cumulative sum of inputs up to time T. We define $\boldsymbol{S}_K(T) = \boldsymbol{S}_A(T)\boldsymbol{S}_B(T)$. Then, the expected output at time T can be formulated as:

$$\boldsymbol{O}(T) = \frac{1}{T}\boldsymbol{S}_K(T) - \frac{1}{T-1}\boldsymbol{S}_K(T-1), \tag{16}$$

where $\boldsymbol{S}_K(T)$ can be calculated mainly using addition, as described by the following equation:

$$\boldsymbol{S}_K(T) = \boldsymbol{S}_K(T-1) + \boldsymbol{K}(T) \tag{17}$$

$$\begin{aligned}\boldsymbol{K}(T) = &\sum_{v_a, v_b} v_a v_b \boldsymbol{A}_{v_a}(T)\boldsymbol{B}_{v_b}(T) + \sum_{v_a} v_a \boldsymbol{A}_{v_a}(T)\boldsymbol{S}_B(T-1) \\ &+ \sum_{v_b} v_b \boldsymbol{S}_A(T-1)\boldsymbol{B}_{v_b}(T).\end{aligned} \tag{18}$$

$$\begin{aligned}\boldsymbol{A}_T &= \frac{\sum_{t=1}^{T} \boldsymbol{A}(t)}{T} \\ \boldsymbol{B}_T &= \frac{\sum_{t=1}^{T} \boldsymbol{B}(t)}{T} \\ \boldsymbol{O}_T &= \frac{\sum_{t=1}^{T} \boldsymbol{O}(t)}{T}\end{aligned}$$

approximate the value of ANNs using the mean value for the first T times in SNNs,

$$\begin{aligned}\boldsymbol{O}(T) &= \sum_{t=1}^{T} \boldsymbol{O}(t) - \sum_{i=t}^{T-1} \boldsymbol{O}(t) \\ &= T\boldsymbol{O}_T - (T-1)\boldsymbol{O}_{T-1} \\ &= T\boldsymbol{A}_T\boldsymbol{B}_T - (T-1)\boldsymbol{A}_{T-1}\boldsymbol{B}_{T-1} \\ &= T\frac{\sum_{t=1}^{T} \boldsymbol{A}(t)}{T}\frac{\sum_{t=1}^{T} \boldsymbol{B}(t)}{T} \\ &\quad - (T-1)\frac{\sum_{t=1}^{T-1} \boldsymbol{A}(t)}{T-1}\frac{\sum_{t=1}^{T-1} \boldsymbol{B}(t)}{T-1} \\ &= \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{A}(t)\sum_{t=1}^{T} \boldsymbol{B}(t) - \frac{1}{(T-1)}\sum_{t=1}^{T-1} \boldsymbol{A}(t)\sum_{t=1}^{T-1} \boldsymbol{B}(t) \\ &= \frac{1}{T}\boldsymbol{S}_A(T)\boldsymbol{S}_B(T) - \frac{1}{T-1}\boldsymbol{S}_A(T-1)\boldsymbol{S}_B(T-1) \\ &= \frac{1}{T}\boldsymbol{S}_K(T) - \frac{1}{T-1}\boldsymbol{S}_K(T-1)\end{aligned}$$

Methods

Sec.2 Expectation Compensation Module

Expectation Compensation Module for Matrix Product.

THEOREM 4.2. *Consider a module for matrix product that receives two sets of spike inputs, denoted by $\boldsymbol{A}_{v_a}(t)$ and $\boldsymbol{B}_{v_b}(t)$. These inputs are generated by neurons A and B, respectively, and are characterized by multiple thresholds v_a and v_b, as described in Section 4.3.*

We can integrate the input by $\boldsymbol{A}(t) = \sum_{v_a} v_a \boldsymbol{A}_{v_a}(t)$ and $\boldsymbol{B}(t) = \sum_{v_b} v_b \boldsymbol{B}_{v_b}(t)$. Here, $\boldsymbol{A}(t)$ and $\boldsymbol{B}(t)$ are the sum matrices weighted by multiple thresholds v_a and v_b, respectively.

Let $\boldsymbol{S}_A(T) = \sum_{t=1}^{T} A(t)$ and $\boldsymbol{S}_B(T) = \sum_{t=1}^{T} B(t)$ represent the cumulative sum of inputs up to time T. We define $\boldsymbol{S}_K(T) = \boldsymbol{S}_A(T)\boldsymbol{S}_B(T)$. Then, the expected output at time T can be formulated as:

$$\boldsymbol{O}(T) = \frac{1}{T}\boldsymbol{S}_K(T) - \frac{1}{T-1}\boldsymbol{S}_K(T-1), \tag{16}$$

where $\boldsymbol{S}_K(T)$ can be calculated mainly using addition, as described by the following equation:

$$\boldsymbol{S}_K(T) = \boldsymbol{S}_K(T-1) + \boldsymbol{K}(T) \tag{17}$$

$$\begin{aligned}\boldsymbol{K}(T) = &\sum_{v_a,v_b} v_a v_b \boldsymbol{A}_{v_a}(T)\boldsymbol{B}_{v_b}(T) + \sum_{v_a} v_a \boldsymbol{A}_{v_a}(T)\boldsymbol{S}_B(T-1) \\ &+ \sum_{v_b} v_b \boldsymbol{S}_A(T-1)\boldsymbol{B}_{v_b}(T).\end{aligned} \tag{18}$$

$$\begin{aligned}\overset{n\times m}{\boldsymbol{S}_K}(T) &= \overset{n\times p}{\boldsymbol{S}_A}(T)\overset{p\times m}{\boldsymbol{S}_B}(T) \\ &= (\sum_{t=1}^{T}\boldsymbol{A}(t))(\sum_{t=1}^{T}\boldsymbol{B}(t)) \\ &= (\boldsymbol{S}_A(T-1)+\boldsymbol{A}(T))(\boldsymbol{S}_B(T-1)+\boldsymbol{B}(T)) \\ &= \boldsymbol{S}_A(T-1)\boldsymbol{S}_B(T-1) + \boldsymbol{A}(T)\boldsymbol{B}(T) \\ &\quad + \boldsymbol{A}(T)\boldsymbol{S}_B(T-1) + \boldsymbol{S}_A(T-1)\boldsymbol{B}(T) \\ &= \boldsymbol{S}_K(T-1) + \sum_{v_a,v_b} v_a v_b \boldsymbol{A}_{v_a}(T)\boldsymbol{B}_{v_b}(T) \\ &\quad + \sum_{v_a} v_a \boldsymbol{A}_{v_a}(T)\boldsymbol{S}_B(T-1) + \sum_{v_b} v_b \boldsymbol{S}_A(T-1)\boldsymbol{B}_{v_b}(T) \\ &= \boldsymbol{S}_K(T-1) + \boldsymbol{K}(T).\end{aligned}$$

η_1 and η_2 are the firing rate of $A(T)$ and $B(T)$

$$ACs_{\text{SNN}}^{max} = \eta_1\eta_2 npm + \eta_1 npm + \eta_2 npm + 3nm$$

$$MACs_{\text{SNN}}^{max} = min(\eta_1, \eta_2)nm + \eta_1 nm + \eta_2 nm$$

$$ACs_{\text{SNN}}^{max} \gg MACs_{\text{SNN}}^{max}$$

Methods

Sec.2 Expectation Compensation Module

Multi-Threshold Neuron

If we only use the Expectation Compensation Module neuron communication will remain in a floating-point format.

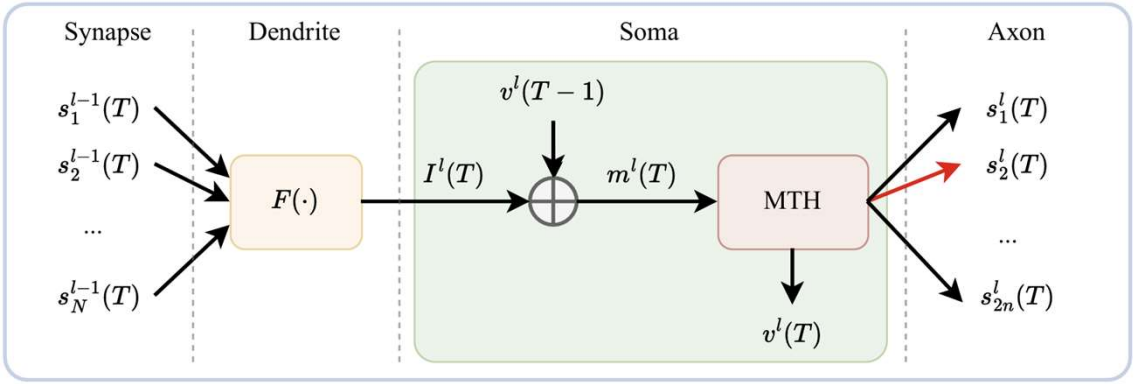
Introduce spiking neurons **before each linear layer and matrix product layer.**

Traditional SNN neurons takes several timesteps to generate at most one spike. **→Latency and power consumption**

Base threshold

$$\lambda_1^l = \theta_1^l, \lambda_2^l = 2\theta_1^l, ..., \lambda_n^l = 2^{n-1}\theta_1^l,$$

$$\lambda_{n+1}^l = -\theta_2^l, \lambda_{n+2}^l = -2\theta_2^l, ..., \lambda_{2n}^l = -2^{n-1}\theta_2^l,$$



输入电流 $I_j^l(t) = F_j^l(s_{,1}^{l-1}(t), ..., s_{,2n}^{l-1}(t)),$

膜电位更新 $m_j^l(t) = v_j^l(t-1) + I_j^l(t),$

脉冲输出 $s_{j,p}^l(t) = MTH_{\theta_1,\theta_2,n}(m_j^l(t))$

脉冲输出的总和 $x_j^l(t) = \sum_p s_{j,p}^l(t)\lambda_p^l,$

膜电位重置 $v_j^l(t) = m_j^l(t) - x_j^l(t).$

Methods

Sec.2 Expectation Compensation Module

Multi-Threshold Neuron

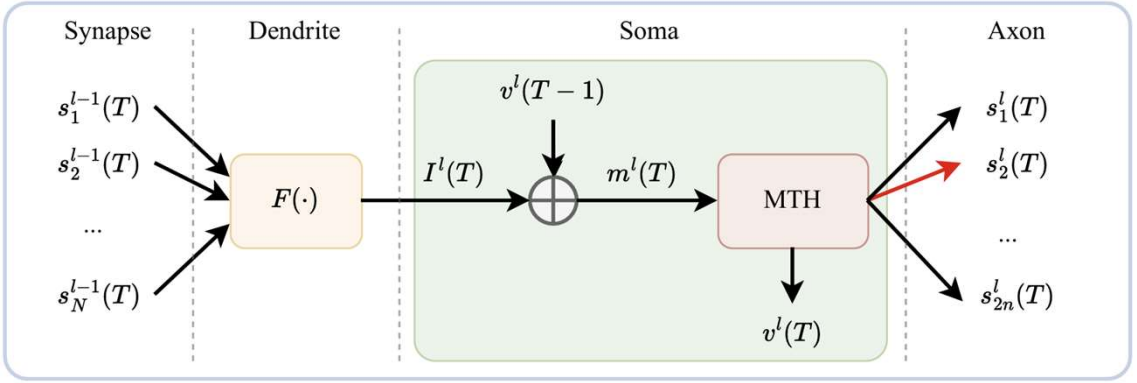
$$s^l_{j,p}(t) = MTH_{\theta_1,\theta_2,n}(m^l_j(t))$$

$MTH_{\theta_1,\theta_2,n}(x)$:

$$\begin{cases} \lambda^l_n - \frac{\lambda^l_1}{2} \le x : & s^l_{j,n}(t) = 1, \\ \lambda^l_{n-1} - \frac{\lambda^l_1}{2} \le x < \lambda^l_n - \frac{\lambda^l_1}{2} : & s^l_{j,n-1}(t) = 1, \\ \cdots & \cdots \\ \frac{\lambda^l_1}{2} \le x < \lambda^l_2 - \frac{\lambda^l_1}{2} : & s^l_{j,1}(t) = 1, \\ \text{未跨越阈值} \quad \frac{\lambda^l_{n+1}}{2} \le x < \frac{\lambda^l_1}{2} : & all\ the\ s^l_{j,p}(t) = 0, \\ \lambda^l_{n+2} - \frac{\lambda^l_{n+1}}{2} \le x < \frac{\lambda^l_{n+1}}{2} : & s^l_{j,n+1}(t) = 1, \\ \cdots & \cdots \\ \lambda^l_{2n} - \frac{\lambda^l_{n+1}}{2} \le x < \lambda^l_{2n-1} - \frac{\lambda^l_{n+1}}{2} : & s^l_{j,2n-1}(t) = 1, \\ x < \lambda^l_{2n} - \frac{\lambda^l_{n+1}}{2} : & s^l_{j,2n}(t) = 1. \end{cases}$$

正阈值部分

负阈值部分

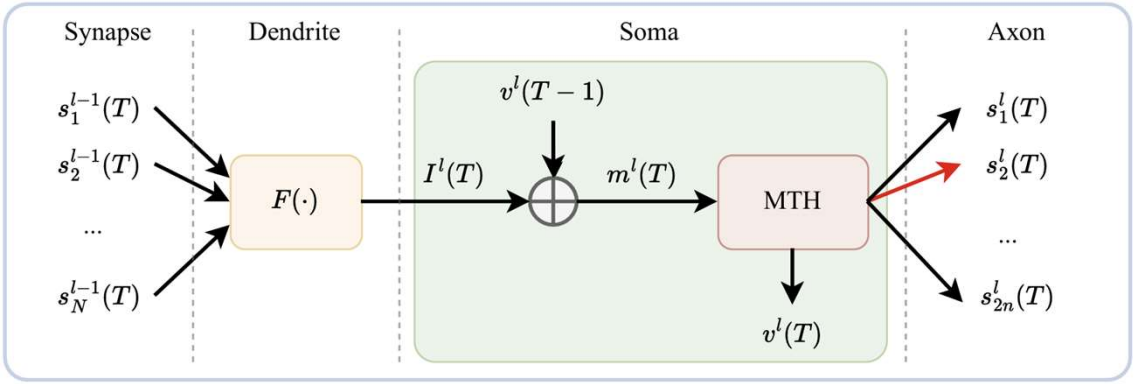


The results of experiments presented in Section 5.4 indicate that although this neuron has multiple thresholds, most of the spikes it generated are concentrated in θ_1 and $-\theta_2$. The spikes generated by other thresholds are minimal, which reduces energy consumption and inference latency.

Methods

Sec.2 Expectation Compensation Module

Parallel Parameter normalization for MT Neuron



extends the ANN weight to ***2n*** weights in the SNN **corresponding to 2n thresholds** of MT neurons

$$W^l_{\mathrm{SNN},p} = W^l_{\mathrm{ANN}} \frac{\lambda_p^{l-1}}{\lambda_1^l}$$

$$I_j^l(t) = \sum_{i,p} w^l_{ij\mathrm{SNN,p}} s_{i,p}^{l-1}(t)$$

$$\theta_{1,new} = 1, \theta_{2,new} = \eta$$

```python
def main(args):
    args.distributed = False
    print("{}".format(args).replace(', ', ',\n'))
    device = torch.device(args.device)
    cudnn.benchmark = True
    dataset_val, args.nb_classes = build_dataset(is_train=False, args=args)
    num_tasks = utils.get_world_size()
    global_rank = utils.get_rank()
    sampler_val = torch.utils.data.DistributedSampler(dataset_val, num_replicas=num_tasks, rank=global_rank, shuffle=False)
    data_loader_val = torch.utils.data.DataLoader(dataset_val,sampler=sampler_val,batch_size=int(args.batch_size),num_workers=args.num_workers,pin_memory=True,drop_last=False)
    model = create_model(args.model,pretrained=False,img_size=args.input_size,num_classes=args.nb_classes)
```

Algorithm 1 The conversion method using Expectation Compensation Module and Multi-Threshold Neuron(ECMT)

Input: Pre-trained Transformer ANN model $f_{\mathrm{ANN}}(\boldsymbol{W})$; Dataset D; Time-step T to test dataset; Threshold percent p.
Output: SNN model $f_{\mathrm{SNN}}(\boldsymbol{W}, \theta_1, \theta_2, v)$

step1: Obtain the base thresholds θ_1 and θ_2
for length of Dataset D **do**
  Sample minibatch data from D
  Run the data on f_{ANN} and static the activation values before linear and matrix product module at $p\%$ and $(1-p\%)$, setting them as θ_1 and $-\theta_2$ respectively.
end for
step2: Converted to SNN model
for module m in f_{ANN}.Module **do**
  if m is Linear Module **then**
    Add a Multi-Threshold Neuron before m
  else if m is Matrix Product **then**
    replace m by two Multi-Threshold Neurons followed by a Matrix Product EC Module
  else if m is Other Nonlinear Module **then**
    replace m by an EC Module
  end if
end for
Set the base thresholds of MT neurons to corresponding $\theta_1, -\theta_2$ and set the initial membrane potential v to 0.
f_{SNN} = Parallel Parameter normalization(f_{ANN})
return f_{SNN}

Experiments