Training Spiking Neural Networks with Accumulated Spiking Flow [7]

AAAI 2021

March 22, 2021

Content

Introduction

The spiking neural network (SNN) is emerging as a promising candidate for the third generation of neural networks. Compared with the previously-proposed artificial neural networks (ANNs), SNN has the advantage of temporal information processing capability, low power consumption, and high biological plausibility.

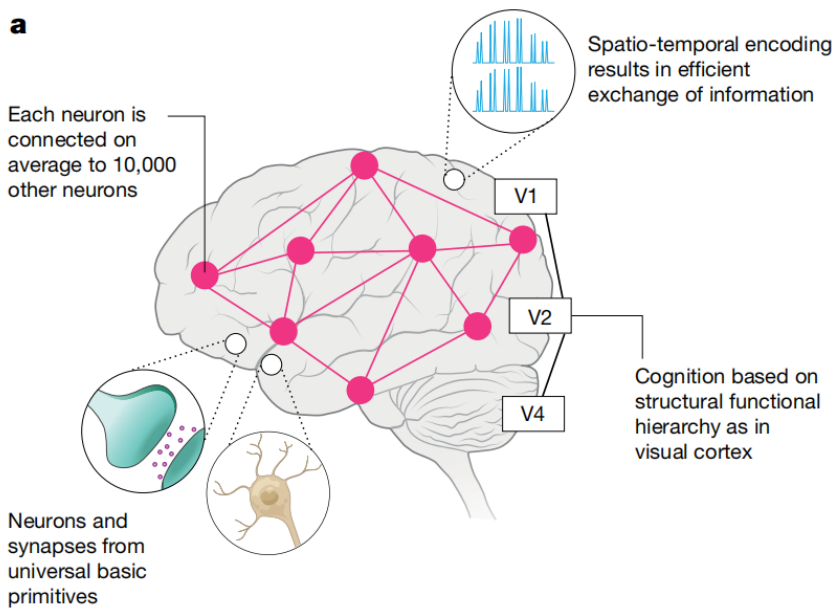

Figure 1: Inspiration of spiking neural networks[5]

Background

Spiking Neural Network

In spiking neural networks, the medium that carry information are spike trains. Inputs are encoded into input spike trains and output spike trains are translated into final result. Among them is the neurons that collect input spikes and fire output spikes.

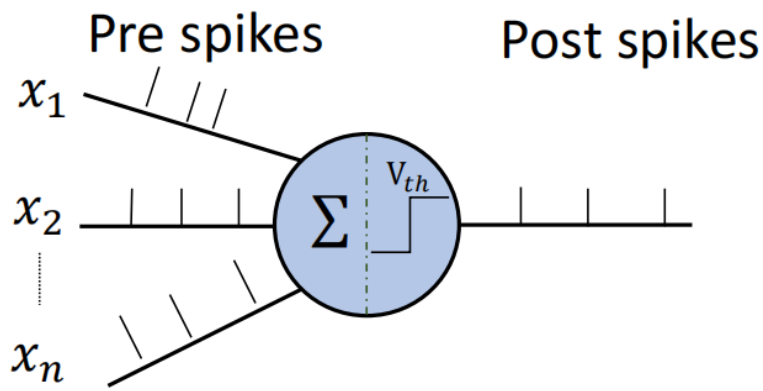

Figure 2: An overview of spiking neural networks

Background

Spiking Neural Network - Neurons

The neuron are responsible for receive spikes from previous layer and fire spikes to the next layer. The most most commonly used model is integrate-and-fire (IF) and leaky integrate-and-fire (LIF) neurons.

In a fully connected layer, The input current at time t is the weighted sum of pre-synaptic spikes:

$$I_i^n(t) = \sum_j w_{ij} s_j^{n-1}(t) \tag{1}$$

where n is the layer index, I^n is the input current, $s(t)$ is spike train.

The current are then accumulated into the membrane voltage V. For IF neurons, the membrane voltage will not decay:

$$V^n(t+1) = V^n(t) + I^n(t)$$

For LIF neurons, the voltage will decay with a time constant of τ:

$$V^n(t+1) = V^n(t) + (I^n(t) - (V^n(t) - V_{reset}))/\tau$$

Background

Spiking Neural Network - Neurons

Then, the neuron will emit an output spike if the membrane voltage exceed the threshold:

$$s^n(t) = \Theta(V^n(t) - V_{threshold}), \tag{2}$$

where $\Theta(x)$ is the Heaviside function

$$\Theta(x) = \begin{cases} 1 & x > 0, \\ 0 & x \leqslant 0, \end{cases} \tag{3}$$

and the membrane voltage is reset to V_{reset} if a spike is fired and remained the same if else:

$$V^n(t) = V^{reset}(t) \cdot s^n(t) + V^n(t) \cdot (1 - s^n(t)) \tag{4}$$

Traditional Training Methods

Supervised Training

Because the Heaviside function used in equatio 2 is not differentiable, the traditional back-propagation methods can not directly be applied on SNNs. There several methods that attempts to introduce BP into SNN.

Some of them use a differentiable surrogate function to simulate the Heaviside function in BP process, such as *Surrogate gradient learning in spiking neural networks* [3]

Others estimate the gradient directly, such as *BP-STDP* [6].

Traditional Training Methods

Supervised Training

Both of the previous-mentioned methods used a loss function that is defined as the mean square error between the average spike over time in the output layer and the target label Y:

$$L = \left\| Y - \sum_t s^N(t) \right\|_2^2$$

Combined with the input current function in equation 1, the gradient of the weight can be expressed as

$$\frac{\partial L}{\partial w_{ij}^n} = \sum_t \frac{\partial L}{\partial V_i^n(t)} s_j^{n-1}(t)$$

This means that the BP process will have an additional temporal complexity, which greatly increase the computational workload.

Methodology

To solve the above-mentioned problem, the paper proposed a method called *Accumulated Spiking Flow* (ASF) to save the additional temporal loop.

First, the neurons in the output are designed to not fired any more but only store the accumulated input current, and the loss function can then be defined as

$$L = \frac{1}{2} \left\| Y - \frac{V^N(T)}{T \cdot V_{th}} \right\|_2^2 \tag{5}$$

where Y is the target label, $V^N(T)$ is the accumulated voltage in the final T.

Methodology

In the forward process, several other elements for each neuron is also calculated:

$$\mathcal{FI}_i^n = \sum_t I_i^n(t)$$

$$\mathcal{FO}_i^n = \sum_t s_i^n(t)$$

$\mathcal{FI}$ is the total input current of the neuron and $\mathcal{FO}$ is the total count of output spikes. Combined with equation 1,

$$\begin{aligned} \mathcal{FI}_i^n &= \sum_t \sum_j w_{ij} s_j^{n-1}(t) \\ &= \sum_j w_{ij} \sum_t s_j^{n-1}(t) \\ &= \sum_j w_{ij} \cdot \mathcal{FO}_j^{n-1} \end{aligned} \tag{6}$$

Methodology

For the output layer, the proposed output $V^N(T)$ is just $\mathcal{FI}^N$. Combined with the loss function of equation 5 and equation 6, the gradient to $\mathcal{FO}^{N-1}$ can expressed as

$$\begin{aligned} \frac{\partial L}{\partial \mathcal{FO}_i^{N-1}} &= \sum_j \frac{\partial L}{\partial \mathcal{FI}_j^N} \cdot \frac{\partial \mathcal{FI}_j^N}{\partial \mathcal{FO}_i^{N-1}} \\ &= \sum_j \frac{1}{T \cdot V_{th}} \left(\frac{\mathcal{FI}_j^N}{T \cdot V_{th}} - Y_j \right) \cdot w_{ij}^N \end{aligned} \tag{7}$$

Methodology

For the other hidden layer, the linear relationship between $\mathcal{FI}^n$ and $\mathcal{FO}^n$ enables us the approximate the differential between them with their ratio:

$$\frac{\partial \mathcal{FO}^n}{\partial \mathcal{FI}^n} = \frac{\mathcal{FO}^n}{\mathcal{FI}^n} = \mathcal{S}^n$$

Then, the gradient to $\mathcal{FO}_i^n$ can be calculated as

$$\begin{aligned} \frac{\partial L}{\partial \mathcal{FO}_i^n} &= \sum_j \frac{\partial L}{\partial \mathcal{FO}_j^{n+1}} \cdot \frac{\partial \mathcal{FO}_j^{n+1}}{\partial \mathcal{FI}_j^{n+1}} \cdot \frac{\partial \mathcal{FI}_j^{n+1}}{\mathcal{FO}_i^n} \\ &= \sum_j \frac{\partial L}{\partial \mathcal{FO}_j^{n+1}} \cdot \mathcal{S}_j^{n+1} \cdot w_{ij}^N \end{aligned} \tag{8}$$

Methodology

With all the gradient with respect to $\mathcal{FO}$, the gradient to w can be calculated as

$$\begin{aligned}\frac{\partial L}{\partial w_{ij}^n} &= \sum_j \frac{\partial L}{\partial \mathcal{FO}_j^n} \cdot \frac{\partial \mathcal{FO}_j^n}{\partial \mathcal{FI}_j^n} \cdot \frac{\partial \mathcal{FI}_j^n}{w_{ij}^n} \\ &= \sum_j \frac{\partial L}{\partial \mathcal{FO}_j^n} \cdot \mathcal{S}_j^n \cdot \mathcal{FO}_i^{n-1}\end{aligned} \tag{9}$$

In practice, the ratio of $\mathcal{FO}$ and $\mathcal{FI}$ varies too much between samples, then the ratio is shared for the whole layer

$$\mathcal{S}^n = \frac{\sum_i \mathcal{FO}_i^n}{\sum_i \mathcal{FI}_i^n}$$

Experiment

Model	Network Structure	Accuracy
(Lee et al. 2016)	20C5-P2-50C5-P2-200FC-10FC	99.31%
(Wu et al. 2018)	15C5-P2-40C5-P2-300FC-10FC	99.42%
(Jin, Zhang, and Li 2018)	15C5-P2-40C5-P2-300FC-10FC	99.49%
(Zhang and Li 2019)	15C5-P2-40C5-P2-300FC-10FC	99.62%
(Lee et al. 2020)	20C5-P2-50C5-P2-200FC-10FC	99.59%
This work (IF)	20C5-P2-50C5-P2-200FC-10FC	**99.65%**
This work (LIF)	20C5-P2-50C5-P2-200FC-10FC	**99.60%**

* 20C5 denotes the convolution layer with 20 channels and the convolution kernel size is 5×5, P2 denotes average pooling layer whose kernel size is 2×2.

Figure 3: Comparison of different models on MNIST.

Experiment

Model	Accuracy
(Panda and Roy 2016)	70.16%
(Wu et al. 2019)	90.53%
(Rueckauer et al. 2017)	90.80%
(Lee et al. 2020)	90.95%
(Wu et al. 2020)	90.98%
This work (IF)	**91.35%**
This work (LIF)	**90.11%**

Figure 4: Comparison of different models on CIFAR-10.

Reference I

[1] Arnon Amir et al. "A low power, fully event-based gesture recognition system". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7243–7252.

[2] Hongmin Li et al. "Cifar10-dvs: an event-stream dataset for object classification". In: *Frontiers in neuroscience* 11 (2017), p. 309.

[3] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63.

[4] Garrick Orchard et al. "Converting static image datasets to spiking neuromorphic datasets using saccades". In: *Frontiers in neuroscience* 9 (2015), p. 437.

Reference II

[5] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. "Towards spike-based machine intelligence with neuromorphic computing". In: *Nature* 575.7784 (2019), pp. 607–617.

[6] Amirhossein Tavanaei and Anthony Maida. "BP-STDP: Approximating backpropagation using spike timing dependent plasticity". In: *Neurocomputing* 330 (2019), pp. 39–47.

[7] Hao Wu et al. "Training Spiking Neural Networks with Accumulated Spiking Flow". In: *ijo* 1.1 (2021).